This watermark does not appear in the registered version - http://www.clicktoconvert.com

UNIT I

LESSON – 1

CONSTANTS & VARIABLES

1.0 AIMS AND OBJECTIVES

In this lesson we are going to learn the character set, Tokens, Keywords, Identifiers, Constants and Variables of C programming language.

After reading this lesson, we should be able to

- identify C tokens
- know C key words and identifiers
- write constants and variables of C language

1.1 INTRODUCTION

A programming language is designed to help certain kinds of *data process* consisting of numbers, characters and strings to provide useful output known as *information*. The task of processing of data is accomplished by executing a sequence of precise instructions called *program.*

1.2 CHARACTER SET

C characters are grouped into the following categories.

1. Letters
2. Digits
3. Special Characters
4. White Spaces

Note: The compiler ignores white spaces unless they are a part of a string constant.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Letters
Uppercase A….Z
Lowercase a…..z
Digits
All decimal digits 0…..9

Special characters

, Comma	& Ampersand
. Period	^ Caret
; Semicolon	* Asterisk
: Colon	- Minus
? Question mark	+ Plus sign
‘ Apostrophe	< Less than
“ Quotation mark	> Greater than
! Exclamation	(Left parenthesis
| Vertical Bar	) Right parentheses
/ Slash	[Left bracket
\ Back slash	] Right bracket
~ Tilde	{ Left brace
_ Underscore	} Right brace
$ Dollar sign	# Number sign
% Percent sign	

White Spaces

- Blank Space
- Horizontal Tab
- Carriage Return
- New Line
- Form Feed

Tri-graph Characters

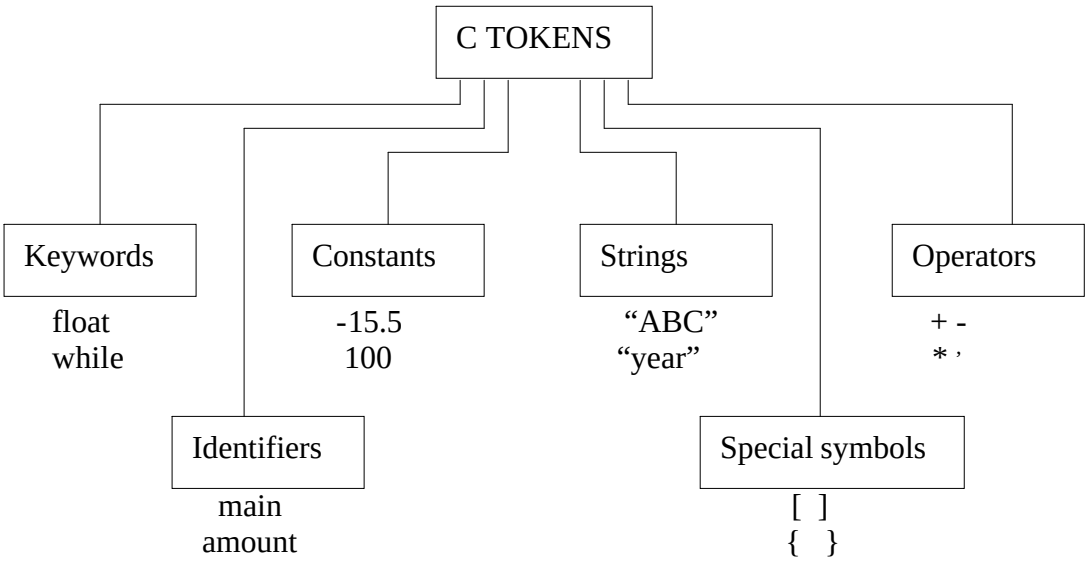
Many non-English keyboards do not support all the characters. ANSI C introduces the concept of “Trigraph” sequences to provide a way to enter certain characters that are not available on some keyboards.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Trigraph Sequence	Translation
??=	# Number sign
??(	[Left bracket
??)	] Right bracket
??<	{ Left brace
??>	} Right brace
??!	| Vertical bar
??/	\ Back slash
??'	^ Caret
??-	~ Tilde

1.3 C TOKENS

In C programs, the smallest individual units are known as *tokens.*



1.4 KEYWORDS AND IDENTIFIERS

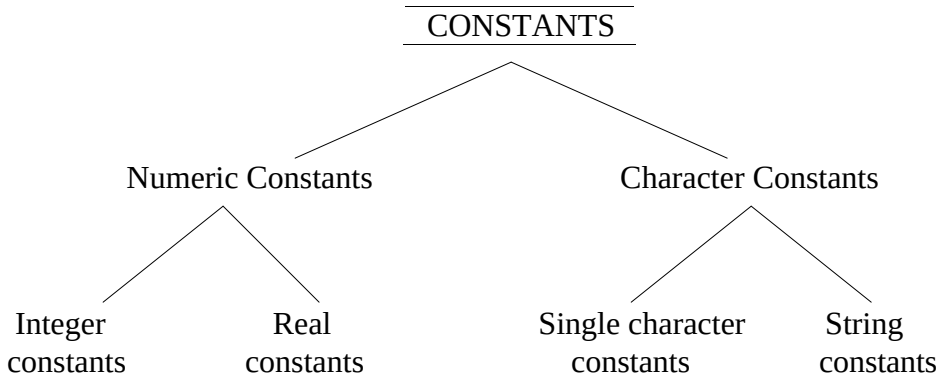
Every C word is classified as either a *keyword* or an *identifier.*
All keywords have fixed meanings and these meanings cannot be changed.
Eg: auto, break, char, void etc.,

Identifiers refer to the names of variables, functions and arrays. They are user-defined names and consist of a sequence of letters and digits, with a letter as a first character. Both uppercase and lowercase letters are permitted. The underscore character is also permitted in identifiers.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1.5 CONSTANTS

Constants in C refer to fixed values that do not change during the execution of a program.



Integer Constants

An integer constant refers to a sequence of digits, There are three types integers, namely, *decimal, octal,* and *hexa decimal*.

Decimal Constant

Eg:123,-321 etc.,

Note: Embedded spaces, commas and non-digit characters are **not** permitted between digits.

Eg: 1) 15 750 2)$1000

Octal Constant

An *octal* integer constant consists of any combination of digits from the set 0 through 7, with a leading 0.

Eg: 1) 037 2) 0435

Hexadecimal Constant

A sequence of digits preceded by 0x or 0X is considered as *hexadecimal* integer. They may also include alphabets A through F or a through f.

Eg: 1) 0X2 2) 0x9F 3) 0Xbcd

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Program for representation of integer constants on a 16-bit computer.

```
/*Integer numbers on a 16-bit machine*/
main()
{
  printf("Integer values\n\n");
  printf("%d%d%d\n",32767,32767+1,32767+10);
  printf("\n");
  printf("Long integer values\n\n");
  printf("%ld%ld%ld\n",32767L,32767L+1L,32767L+10L);
}
```

OUTPUT

```
  Integer values
    32767  -32768  -32759
  Long integer values
    32767  32768  32777
```

Real Constants

Certain quantities that vary continuously, such as distances, heights etc., are represented by numbers containing functional parts like 17.548.Such numbers are called *real* (or *floating point)*constants.

Eg:0.0083,-0.75 etc.,

A real number may also be expressed in *exponential or scientific notation.*

Eg:215.65 may be written as 2.1565e2

Single Character Constants

A single character constants contains a single character enclosed within a pair of *single* quote marks.

Eg: '5'
'X'
';'

String Constants

A string constant is a sequence of characters enclosed in *double* quotes. The characters may be letters, numbers, special characters and blank space.

Eg:"Hello!"
"1987"
"?….!"

Backslash Character Constants

C supports special backslash character constants that are used in output functions. These character combinations are known as *escape sequences.*

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Constant	Meaning
'\a'	audible alert
'\b'	backspace
'\f'	form feed
'\n'	new line
'\0'	null
'\v'	vertical tab
'\t'	horizontal tab
'\r'	carriage return

Check your progress

Ex 1) Write a few numeric constants and character constants.

1.6 VARIABLES

Definition:

A *variable* is a data name that may be used to store a data value. A variable may take different values at different times of execution and may be chosen by the programmer in a meaningful way. It may consist of letters, digits and underscore character.

Eg: 1) Average
2) Height

Rules for defining variables

- They must begin with a letter. Some systems permit underscore as the first character.
- ANSI standard recognizes a length of 31 characters. However, the length should not be normally more than eight characters.
- Uppercase and lowercase are significant.
- The variable name should not be a keyword.
- White space is not allowed.

Check your progress

Ex 2) Write a few meaningful variable names you think .

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1.7 LET US SUM UP

In this lesson we have

- described character set
- learnt the C tokens
- studied about constants and variables of C

1.8 LESSON -END ACTIVITIES

Try to find the answers for the following exercises on your own.

1) Describe the character set of C language
2) What do you understand by C tokens?
3) Differentiate Keywords and Identifiers
4) Describe the constants of C language with examples

1.9 MODEL ANSWERS TO CHECK YOUR PROGRESS

(Answers vary)
Ex -1) 12 - Integer Constant
23.5 - Real Constant
"Amount" – String constant
'S' – Single character constant

Ex-2) total_salary , final_amount, discount are a few meaningful variable names.

1.10 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON – 2

DATA TYPES

2.0 AIMS AND OBJECTIVES

In this lesson we are going to learn about the data types and declaration of variables. Also we are to learn about assigning values to variables and reading data from keyboard.

After learning this lesson, we should be able to

- understand Primary data types
- know how to declare variables
- assign values to variables
- read data from key board

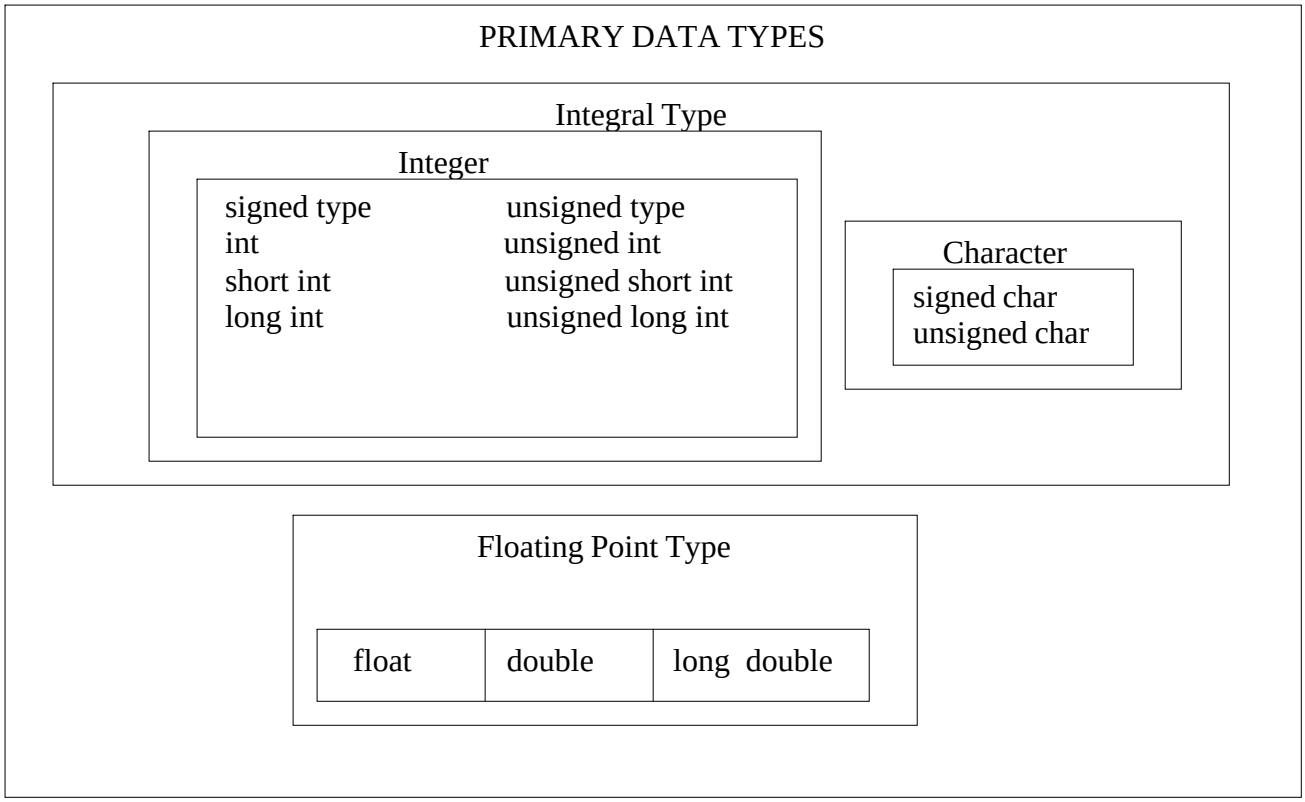
2.1 INTRODUCTION

ANSI C supports four classes of data types.

1. Primary or Fundamental data types.
2. User-defined data types.
3. Derived data types.
4. Empty data set.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

2.2 PRIMARY DATA TYPES



Integer Types

Type	Size (bits)	Range
int or signed int	16	-32,768 to 32767
unsigned int	16	0 to 65535
short int	8	-128 to 127
unsigned short int	8	0 to 255
long int	32	-2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295

Floating Point Types

Type	Size(bits)	Range
float	32	3.4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E+308
long double	80	3.4E-4932to 1.1E+4932

Character Types

Type	Size (bits)	Range
char	8	-128 to 127
unsigned char	8	0 to 255

This watermark does not appear in the registered version - http://www.clicktoconvert.com

2.3 DECLARATION OF VARIABLES

The syntax is

Data-type v1,v2…..vn;

Eg:1.**int** count;
2.**double** ratio, total;

User-defined type declaration

C allows user to define an identifier that would represent an existing **int** data type.
The general form is

typedef type identifier;

Eg: 1) **typedef int** units;
2) **typedef float** marks;

Another user defined data types is enumerated data type which can be used to declare variables that can have one of the values enclosed within the braces.

enum identifier {value1,value2,……valuen};

Declaration of storage class

Variables in C can have not only data type but also storage class that provides information about their locality and visibility.

```
/*Example of storage class*/
int m;
main()
{
  int i;
  float bal;
  ……
  ……
  function1();
}
function1()
{
  int i;
  float sum;
  ……
  ……
}
```

Here the variable **m** is called the global variable. It can be used in all the functions in the program.

The variables **bal, sum and i** are called local variables. Local variables are visible and meaningful only inside the function in which they are declared.

There are four storage class specifiers, namely, auto, static, register and extern.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

2.4 ASSIGNING VALUES TO VARIABLES

The syntax is

```
Variable_name=constant
```

Eg:1) int a=20;
2) bal=75.84;
3) yes='x';

C permits multiple assignments in one line.
Example:

```
initial_value=0;final_value=100;
```

Declaring a variable as constant

Eg: 1) **const int** class_size=40;

This tells the compiler that the value of the int variable class_size must not be modified by the program.

Declaring a variable as volatile

By declaring a variable as volatile, its value may be changed at any time by some external source.

Eg:1) **volatile int** date;

Check Your Progress

Ex 1) declare a few variables and initialize them

2.5 READING DATA FROM KEYWORD

Another way of giving values to variables is to input data through keyboard using the **scanf** function.

The general format of **scanf** is as follows.

```
scanf("control string",&variable1,&variable2,….);
```

The ampersand symbol **&** before each variable name is an operator that specifies the variable name's *address.*

Eg: 1) **scanf("%d",&number);**

This watermark does not appear in the registered version - http://www.clicktoconvert.com

2.6 DEFINING SYMBOLIC CONSTANTS

We often use certain unique constants in a program. These constants may appear repeatedly in a number of places in the program. One example of such a constant is 3.142, representing the value of the mathematical constant **"pi"**.We face two problems in the subsequent use of such programs.

1. Problem in modification of the programs.
2. Problem in understanding the program.

A constant is defined as follows:

#define symbolic-name value of constant

Eg: 1) **#define pi** 3.1415
2) #**define** pass_mark 50

The following rules apply to a **#define** statement which define a symbolic constant

- Symbolic names have the same form as variable names.
- No blank space between the sign **'#'** and the word **define** is permitted
- **'#'** must be the first character in the line.
- A blank space is required between **#define** and *symbolic name* and between the *symbolic name* and the *constant.*
- **#define** statements must not end with the semicolon.
- After definition, the symbolic name should not be assigned any other value within the program by using an assignment statement.
- Symbolic names are NOT declared for data types. Their data types depend on the type of constant.
- **#define** statements may appear *anywhere* in the program but before it is referenced in the program.

Check Your Progress

Ex 2) Write a few symbolic constants

2.7 LET US SUM UP

In this lesson, we leant about

- primary Data Types
- declaration of Variables
- assigning values to variables
- reading data from keyword
- defining symbolic constants

This watermark does not appear in the registered version - http://www.clicktoconvert.com

2.8 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own

1) List out the four classes of data types supported by ANSI C.
2) Sketch out the Primary Data Types
3) Explain the method of declaring and assigning values to variables.
4) What is the role of Symbolic constants in C?

2.9 MODEL ANSWERS TO CHECK YOUR PROGRESS

[Answers vary]

Ex – 1 1) int stu_num = 50;
2) float minbal = 500;
3) char test = ‘s’;

Ex-2 1) #define STUNUM 50
2) #define CHECKVAL 1

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON – 3

OPERATORS

3.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about the various operators of C language that include among others arithmetic, relational and logical operators.

After reading this lesson, we should be able to understand

- arithmetic operators
- relational operators
- logical, assignment operators
- increment, decrement, conditional operators
- bitwise and special operators.

3.1 OPERATORS OF C

C supports a rich set of operators. Operators are used in programs to manipulate data and variables. They usually form a part of the mathematical of logical expressions.

C operators are classified into a number of categories. They include:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and Decrement operators
6. Conditional operators
7. Bitwise operators
8. Special operators

This watermark does not appear in the registered version - http://www.clicktoconvert.com

3.2 ARITHMETIC OPERATORS

The operators are
+ (Addition)
- (Subtraction)
* (Multiplication)
/ (Division)
% (Modulo division)

Eg: 1) a-b 2) a+b 3) a*b 4) p%q

The modulo division produces the remainder of an integer division.
The modulo division operator cannot be used on floating point data.

Note: C does not have any operator for *exponentiation*.

Integer Arithmetic

When both the operands in a single arithmetic expression are integers, the expression is called an *integer expression* , and the operation is called *integer arithmetic.*

During modulo division the sign of the result is always the sign of the first operand.

That is

```
-14 %  3  = -2
-14 % -3 =  -2
 14 % -3 =  2
```

Real Arithmetic

An arithmetic operation involving only real operands is called *real arithmetic.* If **x and y** are floats then we will have:

1) x = 6.0 / 7.0 = 0.857143
2) y = 1.0 / 3.0 = 0.333333

The operator % cannot be used with real operands.

Mixed-mode Arithmetic

When one of the operands is real and the other is integer, the expression is called *a mixed-mode arithmetic* expression and its result is always a real number.

Eg: 1) 15 / 10.0 = 1.5

3.3 RELATIONAL OPERATORS

Comparisons can be done with the help of *relational operators*. The expression containing a relational operator is termed as a *relational expression.* The value of a relational expression is either *one* or *zero.*

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1) < (is less than)
2) <= (is less than or equal to)
3) > (is greater than)
4) >= (is greater than or equal to)
5) = = (is equal to)
6) != (is not equal to)

3.4 LOGICAL OPERATORS

C has the following three *logical operators.*

&& (logical **AND**)
|| (logical **OR**)
! (logical **NOT**)

Eg: 1) if(age>55 && sal<1000)
2) if(number<0 || number>100)

3.5 ASSIGNMENT OPERATORS

The usual assignment operator is '='.In addition, C has a set of 'shorthand' assignment operators of the form, v **op** = exp;

Eg:1.x += y+1;

This is same as the statement
x=x+(y+1);

Shorthand Assignment Operators

Statement with shorthand operator	Statement with simple assignment operator
a + =1	a = a + 1
a - = 1	a = a – 1
a *= n + 1	a = a * (n+1)
a /= n + 1	a = a / (n+1)
a %= b	a = a % b

3.6 INCREMENT AND DECREMENT OPERATORS

C has two very useful operators that are not generally found in other languages. These are the *increment* and *decrement* operator:

++ and --

The operator ++ adds 1 to the operands while – subtracts 1.It takes the following form:

++m; or m++
--m; or m—

This watermark does not appear in the registered version - http://www.clicktoconvert.com

3.7 CONDITIONAL OPERATOR

A ternary operator pair "?:" is available in C to construct conditional expression of the form:

```
exp1 ? exp2 : exp3;
```

Here *exp1* is evaluated first. If it is true then the expression *exp2* is evaluated and becomes the value of the expression. If *exp1* is false then *exp3* is evaluated and its value becomes the value of the expression.

Eg:1)
```
if(a>b)
    x = a;
else
    x = b;
```

Check Your Progress

Ex 1) List the arithmetic operators of C

---

2) What is the answer of -5 % 2 ?

---

3) If x =10 , then , x+=5 evaluates to ----------

3.8 BITWISE OPERATORS

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right
~	One's complement

3.9 SPECIAL OPERATORS

C supports some special operators such as

- Comma operator
- Size of operator
- Pointer operators(& and *) and
- Member selection operators(. and ->)

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The Comma Operator

The comma operator can be used to link the related expressions together. A comma-linked list of expressions are evaluated *left* to *right* and the value of *right-most* expression is the value of the combined expression.

Eg: value = (x = 10, y = 5, x + y);

This statement first assigns the value 10 to **x**, then assigns 5 to **y,** and finally assigns 15(i.e, 10+5) to **value**.

The Size of Operator

The size of is a compiler time operator and, when used with an operand, it returns the number of bytes the operand occupies.

Eg: 1) m = **sizeof**(sum);
2) n = **sizeof(long int)**
3) k = **sizeof**(235L)

3.10 LET US SUM UP

In this lesson we have leant about the following operators of C language, namely,

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment and Decrement Operators
- Conditional Operator
- Bitwise Operators
- Special Operators

With the knowledge of these operators, we can use them in our programs whenever need arises.

3.11 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1) List out the operators supported by C language
2) Specify the Arithmetic operators with examples
3) Bring out the Logical operators of C
4) Point out the role of Increment and Decrement operators
5) What do you mean by conditional operator?
6) Write a few points about Bitwise operators.

3.12 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) + - * / %
2) -1 [Remainder after division]
3) x=15

3.13 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.

Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON – 4

EXPRESSIONS

4.0 AIMS AND OBJECTIVES

In this lesson, we are going to study about the expressions, precedence of arithmetic operators, type conversion, mathematical functions and Input-output operations.

After studying this lesson, we should be able to

- identify expressions
- understand the precedence of arithmetic operators
- know how type conversion works
- get knowledge about mathematical functions of C
- mange input and output operations of C

4,1 EXPRESSIONS

The combination of operators and operands is said to be an expression.

4.2 ARITHMETIC EXPRESSIONS

An arithmetic expression is a combination of variables, constants, and operators arranged as per the syntax of the language.

Eg 1) a = x + y;

EVALUATION OF EXPRESSIONS

Expressions are evaluated using an assignment statement of the form

variable = expression;

Eg:1) x = a * b – c;
2) y = b / c * a;

This watermark does not appear in the registered version - http://www.clicktoconvert.com

4.3 PRECEDENCE OF ARITHMETIC OPERATORS

An arithmetic expression without parenthesis will be evaluated from left to right using the rule of precedence of operators. There are two distinct priority levels of arithmetic operators in C.

High priority * / %
Low priority + -

Program

```
/*Evaluation of expressions*/
main()
{
  float a, b, c, y, x, z;
  a =  9;
  b = 12;
  c =  3;
  x  = a – b / 3 + c * 2 – 1;
  y  = a – b / (3 + c) * (2 – 1);
  z  = a – (b / (3 + c) * 2) – 1;
  printf(“x = %f \n”,x);
  printf(“y = %f \n”,y);
  printf(“z = %f \n”,z);
}
```

OUTPUT

```
x  = 10.000000
y  =  7.000000
z  =  4.000000
```

SOME COMPUTATIONAL PROBLEMS

When expressions include real values, then it is important to take necessary precautions to guard against certain computational errors. Some of these errors could be approximate values for real numbers, division by zero and overflow or underflow errors.

Program

```
/*Program showing round-off errors*/
/*Sum of n terms of 1/n*/
main()
{
float sum, n, term;
int count = 1;
sum = 0;
printf(“Enter value for n \n”);
scanf(“%f”,&n);
term = 1.0 / n;
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
while(count <= n)
{
sum = sum + term;
count ++;
}
printf("sum = %f\n",sum);
}
```

OUTPUT

```
Enter value of n
 99
Sum = 1.000001
Enter value of n
 143
 Sum = 0.999999
```

We know that the sum of n terms of 1/n is 1.However due to errors in floating-point representation; the result is not always 1.

4.4 TYPE CONVERSION IN EXPRESSIONS

Automatic Type Conversion

C permits the mixing of constants and variables of different types in an expression. If the operands are of different types, the lower type is automatically converted to higher type before the operation proceeds. The result is of the higher type.

Given below are the sequence of rules that are applied while evaluating expressions.

All **short** and **char** are automatically converted to **int**; then

- If one of the operands is long double, the other will be converted to long double and the result will be in long double.
- Else, If one of the operands is double, the other will be converted to double and the result will be in double.
- Else, If one of the operands is float, the other will be converted to float and the result will be in float.
- Else, If one of the operands is unsigned long int, the other will be converted to unsigned long int and the result will be in unsigned long int.
- Else, if one of the operands is long int and other is unsigned int, then:
  - (a) if unsigned int can be converted to long int, the unsigned int operand will be converted as such and the result will be long int.
  - (b) else, both the operands will be converted to unsigned long int and the result will be unsigned long int.
- Else, If one of the operands is long int, the other will be converted to long int and the result will be in long int.
- Else, If one of the operands is unsigned int, the other will be converted to unsigned int and the result will be in unsigned int.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Casting a Value

C performs type conversion automatically. However, there are instances when we want to force a type conversion in a way that is different from the automatic conversion.

Eg: 1) ratio = female_number / male_number

Since female_number and male_number are declared as integers the ratio would represent a wrong figure. Hence it should be converted to float.

ratio = (float) female_number / male_number

The general form of a cast is:

(type-name)expression

4.5 MATHEMATICAL FUNCTIONS

Mathematical functions such as sqrt, cos, log etc., are the most frequently used ones. To use the mathematical functions in a C program, we should include the line

#include<math.h>

in the beginning of the program.

Function	Meaning
Trignometric acos(x) asin(x) atan(x) atan2(x,y) cos(x) sin(x) tan(x)	Arc cosine of x Arc sine of x Arc tangent of x Arc tangent of x/y cosine of x sine of x tangent of x
Hyperbolic cosh(x) sinh(x) tanh(x)	Hyperbolic cosine of x Hyperbolic sine of x Hyperbolic tangent of x
Other functions ceil(x) exp(x) fabs(x) floor(x) fmod(x,y) log(x) log10(x) pow(x,y) sqrt(x)	x rounded up to the nearest integer e to the power x absolute value of x x rounded down to the nearest integer remainder of x/y natural log of x, x>0 base 10 log of x.x>0 x to the power y square root of x,x>=0

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Check Your Progress

Ex 1) What will be the output of the expressions given below:
Given a = 5, b=2
x=a+b*2
y= a*b+2

4.6 MANAGING INPUT AND OUTPUT OPERATIONS

All input/output operations are carried out through functions called as **printf** and **scanf**. There exist several functions that have become standard for input and output operations in C. These functions are collectively known as *standard i/o library*. Each program that uses standard I/O function must contain the statement

#include<stdio.h>

The file name stdio.h is an abbreviation of *standard input-output header file*.

READING A CHARACTER

Reading a single character can be done by using the function **getchar**. The **getchar** takes the following form:

variable_name = **getchar();**

Eg:char name;
name=getchar();

Program

```
/*Reading a character from terminal*/
#include<stdio.h>
main()
{
   char ans;
   printf("Would you like to know my name? \n");
   printf("Type Y for yes and N for no");
   ans=getchar();
   if(ans =='Y' || ans = ='y')
      printf("\n\n My name is India \n");
   else
      printf("\n\n You are good for nothing \n");
}
```

OUTPUT

```
Would you like to know my name?
Type Y for yes and N for no:Y
My name is India
Would you like to know my name?
Type Y for yes and N for no:n
You are good for nothing
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

WRITING A CHARACTER

Like **getchar,** there is an analogous function **putchar** for writing characters one at a time to the terminal. It takes the form as shown below:

putchar (variable_name);

Eg: 1) answer='y';
putchar(answer);
will display the character y on the screen.
The statement

putchar('\n');

would cause the cursor on the screen to move to the beginning of the next line.

Program

```
/*A program to read a character from keyboard and then prints it in reverse case*/
/*This program uses three new functions:islower,toupper,and tolower.
#include<stdio.h>
#include<ctype.h>
main()
{
char alphabet;
printf("Enter an alphabet");
putchar('\n');
alphabet = ghetchar();
if(islower(alphabet))
putchar(toupper(alphabet));
else
putchar(tolower(alphabet));
}
```

OUTPUT

```
Enter An alphabet
  a
  A
Enter An alphabet
  Q
  q
 Enter An alphabet
  z
  Z
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

FORMATTED INPUT

Formatted input refers to an input data that has been arranged in a particular format. The formatted data can be read with the help of **scanf** function. The general form of **scanf** is

scanf(“control string”,arg1,arg2….argn);

The* control string *specifies the field format in which the data is to be entered and the arguments* arg1,arg2...argn *specifies the address of locations where the data are stored. Control strings and arguments are separated by commas.

Control string contains field specification which direct the interpretation of input data. It may include

- Field(or format)specifications, consisting of conversion character %, a data type character, and an optional number, specifying the field width.
- Blanks, tabs, or newlines.

Inputting integer numbers

The field specification for reading an integer number is

%wd

Eg: **scanf(“%2d %5d”,&num1, &num2);**

An input field may be skipped by specifying * in the place of field width.
For eg **,**

scanf(“%d %*d %d”,&a, &b);

Program

```
/*Reading integer numbers*/
main()
{
        int a, ,b, c, x, y, z;
        int p, q, r;
        printf(“Enter three integer numbers \n”);
        scanf(“%d  %*d  %d”,&a, &b, &c);
        printf(“%d  %d  %d \n \n”,a, b, c);

        printf(“Enter two 4-digit numbers \n”);
        scanf(“%2d  %4d ”,&x, &y);
        printf(“%d  %d  \n \n”,x, y);

        printf(“Enter two integer numbers \n”);
        scanf(“%d  %d”,&a, &x);
        printf(“%d  %d  \n \n”,a, x);
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
        printf("Enter a nine digit numbers \n");
        scanf("%3d  %4d  %3d",&p, &q, &r);
        printf("%d  %d  %d \n \n",p, q, r);

        printf("Enter two three digit numbers \n");
        scanf("%d   %d",&x, &y);
        printf("%d  %d \n \n",x, y);
}
```

OUTPUT

```
Enter three integer numbers
   1 2 3
   1 3 –3577

Enter two 4-digit numbers
6789              4321
67              89

Enter two integer numbers
        44        66
  4321  44

Enter a nine digit numbers
  123456789
        66        1234  567

Enter two three digit numbers
         123      456
       89         123
```

Inputting Real Numbers

Unlike integer numbers, the field width of real numbers is not to be specified and therefore **scanf** reads real numbers using simple specification **%f** for both the notations, namely, decimal point notation and exponential notation.

Eg: **scanf("%f %f %f", &x, &y, &z);**

If the number to be read is of **double** type, then the specification should be **%lf** instead of simple **%f**.

Inputting Character Strings

Following are the specifications for reading character strings:

%ws or **%wc**

Some versions of **scanf** support the following conversion specifications for strings:

%[characters] and %[^characters]

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The specification **%[characters]** means that only the characters specified within the brackets are permissible in the input string. If the input string contains any other character, the string will be terminated at the first encounter of such a character. The specification **%[^characters]** does exactly the reverse. That is, the characters specified after the circumflex (^) are not permitted in the input string.

Reading Mixed Data Types

It is possible to use one **scanf** statement to input a data line containing mixed mode data. In such cases, it should be ensured that the input data items match the control specifications in *order* and *type*.

Eg: **scanf("%d %c %f %s",&count, &code, &ratio, &name);**

Scanf Format Codes

Code	Meaning
%c	Read a single character
%d	Read a decimal integer
%e	Read a floating point value
%f	Read a floating point value
%g	Read a floating point value
%h	Read a short integer
%i	Read a decimal, hexadecimal, or octal integer
%o	Read an octal integer
%s	Read a string
%u	Read an unsigned decimal integer
%x	Read a hexa decimal integer
%[..]	Read a string of word(s)

Points To Remember while using scanf

- ❖ All function arguments, except the control string, must be pointers to variables.
- ❖ Format specifications contained in the control string should match the arguments in order.
- ❖ Input data items must be separated by spaces and must match the variables receiving the input in the same order.
- ❖ The reading will be terminated, when scanf encounters an 'invalid mismatch' of data or a character that is not valid for the value being read.
- ❖ When searching for a value, scanf ignores line boundaries and simply looks for the next appropriate character.
- ❖ Any unread data items in a line will be considered as a part of the data input line to the next scanf call.
- ❖ When the field width specifier *w* is used, it should be large enough to contain the input data size.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

FORMATTED OUTPUT

The **printf** statement provides certain features that can be effectively exploited to control the alignment and spacing of print-outs on the terminals.The general form of **printf** statement is

printf(“control string”,arg1, arg2…argn);

Control string consists of three types of items:

1. Characters that will be printed on the screen as they appear.
2. Format specifications that define the output format for display of each item.
3. *Escape sequence* characters such as \n, \t and \b

Output of Integer Numbers

The format specification for printing an integer number is

%wd

Output of Real Numbers

The output of real numbers may be displayed in decimal notation using the following format specification:

%w.p f

The integer *w* indicates the minimum number of positions that are to be used for the display of the value and the integer *p* indicates the number of digits to be displayed after the decimal point.

We can also display real numbers in exponential notation by using the specification

%w.p e

Printing of Single Character

A single character can be displayed in a desired position using the format

%wc

The character will be displayed *right-justified* in the field of *w* columns. We can make the display *left-justified* by placing a minus sign before the integer *w*.The default value for *w* is 1.

Printing of Strings

The format specification for outputting strings is of the form

%w.ps

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Mixed Data Output

It is permitted to mix data types in one printf statements.

Eg: **printf("%d %f %s %c",a, b, c, d);**

4.7 LET US SUM UP

In this lesson we have studied about

- expressions
- arithmetic Expressions
- precedence of Arithmetic Operators
- type Conversion In Expressions
- mathematical Functions
- **managing Input and Output Operations**

4.8 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1) What is meant by an expression?
2) Bring out the precedence of Arithmetic operators.
3) Explain how type conversion works in expressions.
4) Tabulate the mathematical functions supported by C.
5) List out the scanf Format codes.

4.9 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) x= 9 [Explanation : First b*2 = 4 , then 5+4 = 9]
y= 12 [Explanation : First a*b =10 , then 10+2 =12]

4.10 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON - 5

CONTROL STATEMENTS

5.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about the control statements, conditional statements, unconditional statements, decision making and looping.

After learning this lesson, we should be able to

- understand control statements
- make use of conditional statements
- get ideas about switch statement
- know unconditional statements
- make decisions and use looping statements.

5.1 CONTROL STATEMENTS

C language supports the following statements known as *control* or *decision making* statements.

1. **if** statement
2. **switch** statement
3. conditional operator statement
4. **goto** statement

5.2 CONDITIONAL STATEMENTS

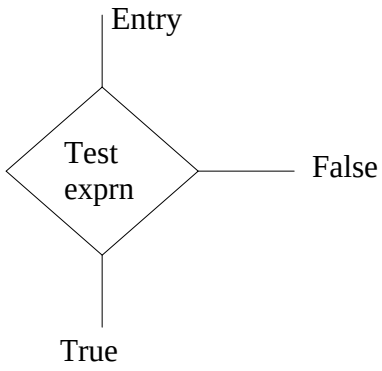
IF STATEMENT

The **if** statement is used to control the flow of execution of statements and is of the form

If(test expression)

This watermark does not appear in the registered version - http://www.clicktoconvert.com

It allows the computer to evaluate the expression first and then, depending on whether the value of the expression is 'true' or 'false', it transfers the control to a particular statement.



Eg: **if**(bank balance is zero)
Borrow money

The **if** statement may be implemented in different forms depending on the complexity of conditions to be tested.

1. Simple **if** statement
2. **if**…..**else** statement
3. Nested **if**…..**else** statement
4. **elseif ladder**

SIMPLE IF STATEMENT

The general form of a simple **if** statement is The 'statement-block' may be a single statement or a group of statement. If the *test expression* is true, the *statement-block* will be executed; otherwise the statement-block will be skipped and the execution will jump to the *statement-x*.

```
If(test exprn)
  {
    statement-block;
   }
statement-x;
```

E.g.

```
if(category = = SPORTS)
{
        marks = marks + bonus_marks;
}
printf("%f ",marks);
…….
…….
```

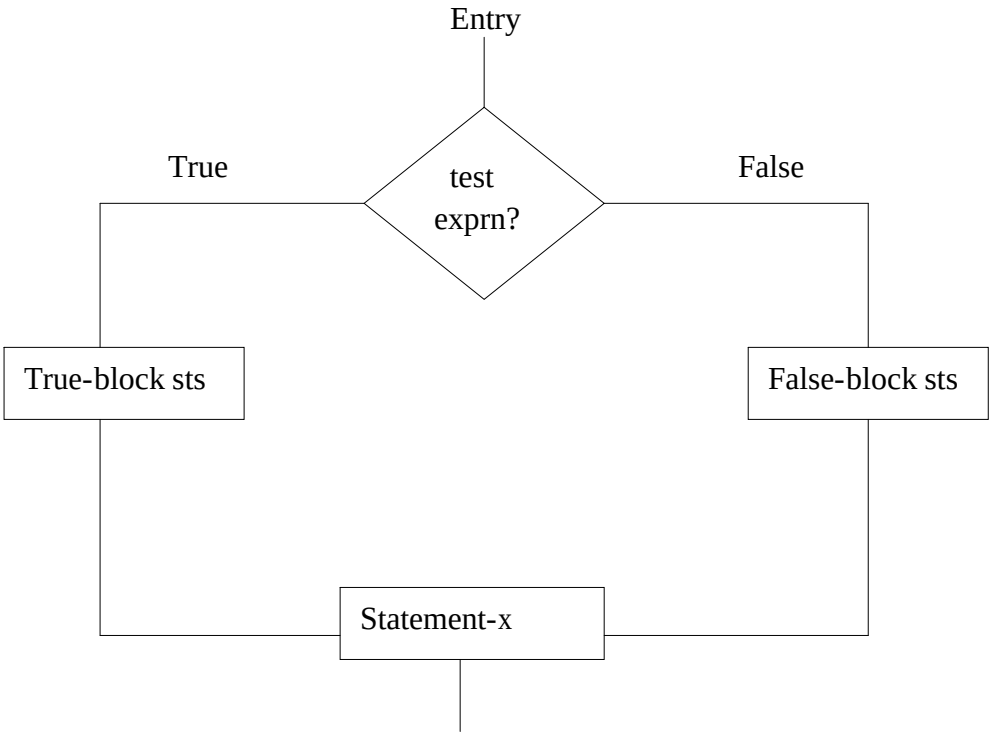
This watermark does not appear in the registered version - http://www.clicktoconvert.com

THE IF…ELSE STATEMENT

The **if….else** statement is an extension of simple **if** statement.The general form is

```
If(test expression)
  {
     True-block statement(s)
  }
else
  {
    False-block statement(s)
  }
statement-x
```

If the *test expression* is true, then the true block statements are executed; otherwise the false block statement will be executed.



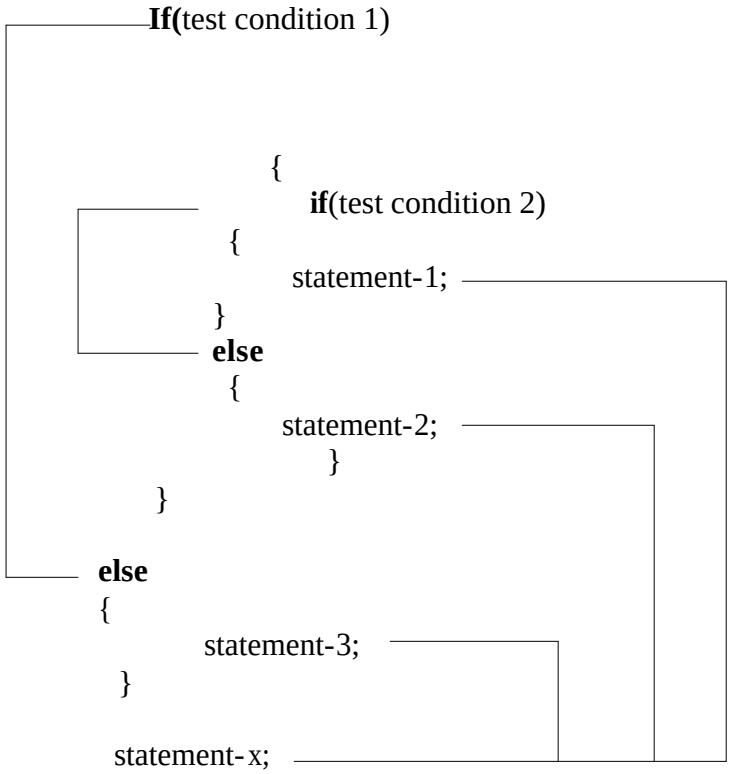
Eg:

```
……
……
if(code ==1)
   boy = boy + 1;
if(code == 2)
   girl =girl + 1;
……
……
```

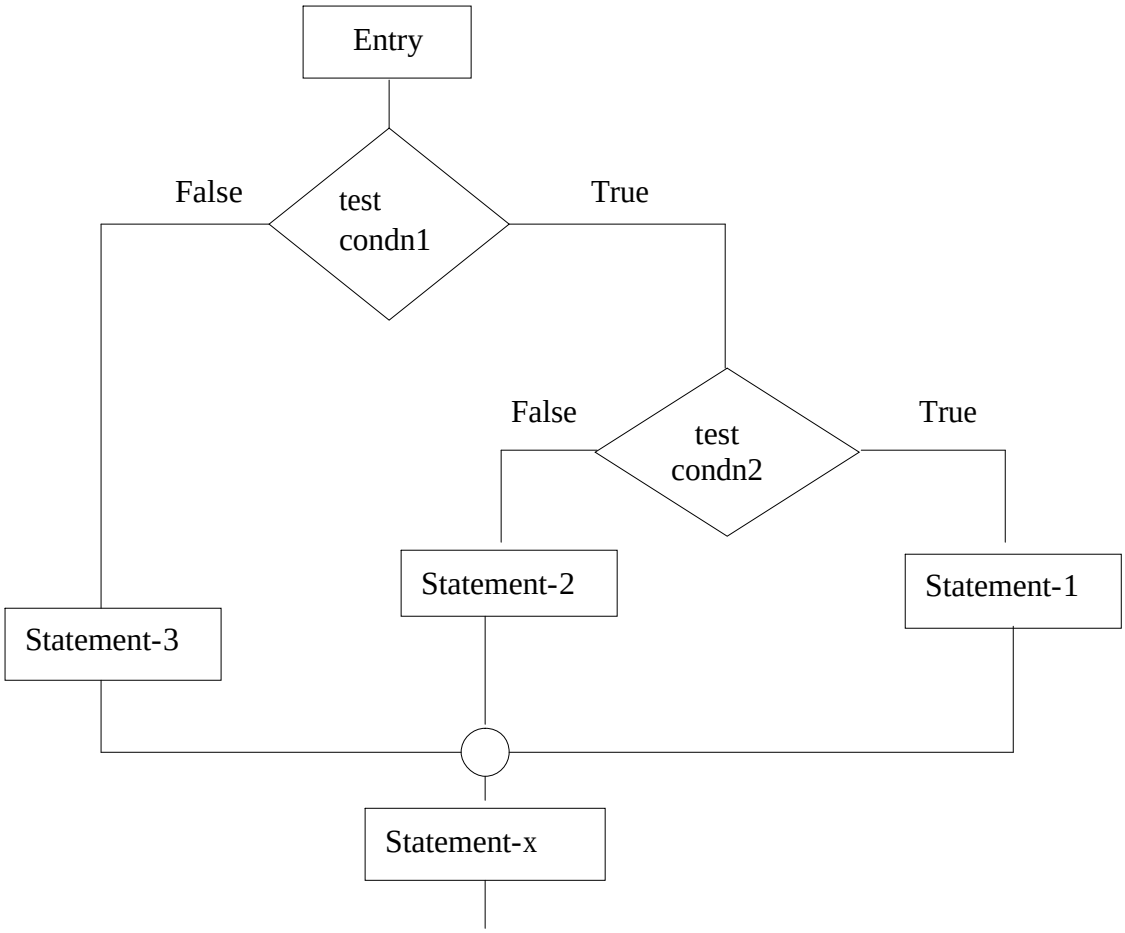
This watermark does not appear in the registered version - http://www.clicktoconvert.com

NESTING OF IF…..ELSE STATEMENTS

When a series of decisions are involved, we may have to use more than one **if….else** statements, in *nested* form as follows.



This watermark does not appear in the registered version - http://www.clicktoconvert.com



Program

```
/*Selecting the largest of three values*/
main()
{
  float A, B, C;
  printf("Enter three values \n");
  scanf("|%f %f %f",&A, &B, &C);
  printf("\nLargest value is:");
  if(A > B)
  {       if(A > C)
                printf("%f \n",A);
          else
                printf("%f \n",C);
   }
     else
        {
          if(C > B)
            printf("%f \n",C);
          else
             printf("%f \n",B);
          }
}
```

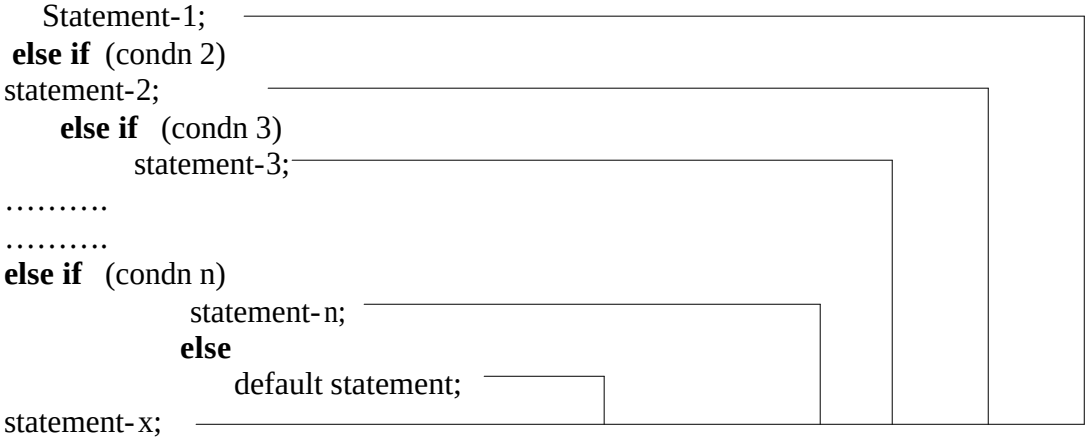
This watermark does not appear in the registered version - http://www.clicktoconvert.com

OUTPUT
Enter three values:
5 8 24

Largest value is 24

THE ELSEIF LADDER

The general form is



Program

```
/*Use of else if ladder*/
main()
{
        int units, cno;
        float charges;
        printf("Enter customer no. and units consumed \n");
 scanf("%d %d",&cno, &units );
 if(units <= 200)
       charges = 0.5 * units;
  else if(units <= 400)
        charges = 100+ 0.65 * (units – 200)
      else if (units <= 600)
              charges = 230 + 0.8 * (units – 400)
 else
               charges = 390 + (units – 600);
printf("\n \ncustomer no =%d,charges =%.2f \n",cno,charges);
}
```

OUTPUT

Enter customer no. and units consumed 101 150
Customer no=101 charges = 75.00

This watermark does not appear in the registered version - http://www.clicktoconvert.com

5.3 THE SWITCH STATEMENT

Switch statement is used for complex programs when the number of alternatives increases. The switch statement tests the value of the given variable against the list of **case** values and when a match is found, a block of statements associated with that case is executed. The general form of switch statement is

```
switch(expression)
  {
      case  value-1:
                block-1
                break;
       case   value-2:
                block-2
                break;
          …….
          …….
          default:
                default-block
                break;
}
statement-x;
```

Eg:

```
……
……
index = marks / 10;
switch(index)
{
  case 10:
  case  9:
  case  8:
        grade = “Honours”;
        break;
  case  7:
  case  6:
        grade = “first division”;
        break;
  case  5:
        grade = “second division”;
        break;
  case  4:
        grade = “third division”;
        break;
  default:
        grade = “first division”;
        break;
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
}
printf("%s \n",grade);
……
```

THE ?: OPERATOR

The C language has an unusual operator, useful for making two-way decisions. This operator is a combination of **?** and **:** and takes three operands. It is of the form

exp1?exp2:exp 3

Here *exp1* is evaluated first. If it is true then the expression *exp2* is evaluated and becomes the value of the expression. If *exp1* is false then *exp3* is evaluated and its value becomes the value of the expression.

Eg:

```
if(x < 0)
  flag = 0;
else
  flag = 1;
```

can be written as

```
flag = (x < 0)? 0 : 1;
```

5.4 UNCONDITIONAL STATEMENTS

THE GOTO STATEMENT

C supports the goto statement to branch unconditionally from one point of the program to another. The goto requires a *label* in order to identify the place where the branch is to be made. A **label** is any valid variable name and must be followed by a colon.
The general from is

```
goto label
----------
----------
label:
statement;
```

```
label:
statement;
----------
----------
goto label
```

Note: A label can be anywhere in the program, either before or after the **goto** label; statement.

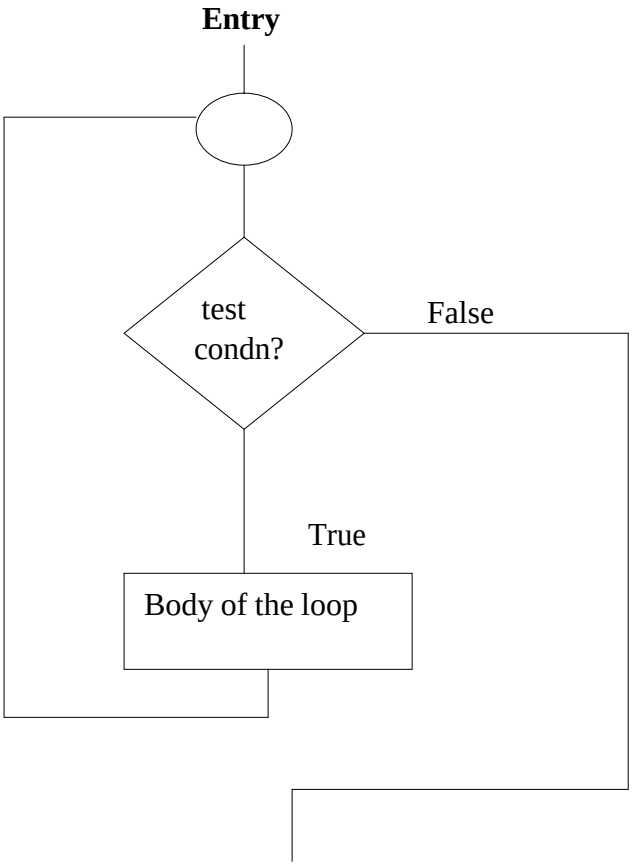
5.5 DECISION MAKING AND LOOPING

It is possible to execute a segment of a program repeatedly by introducing a counter and later testing it using the **if** statement. While this method is quite satisfactory for all practical purposes, we need to initialize and increment a counter and test its value at an appropriate place in the program for the completion of the loop.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

In looping, a sequence of statements are executed until some conditions for termination of the loop are satisfied. A *program loop* therefore consists of two segments, one known as the *body of the loop* and the other known as the *control statements.*

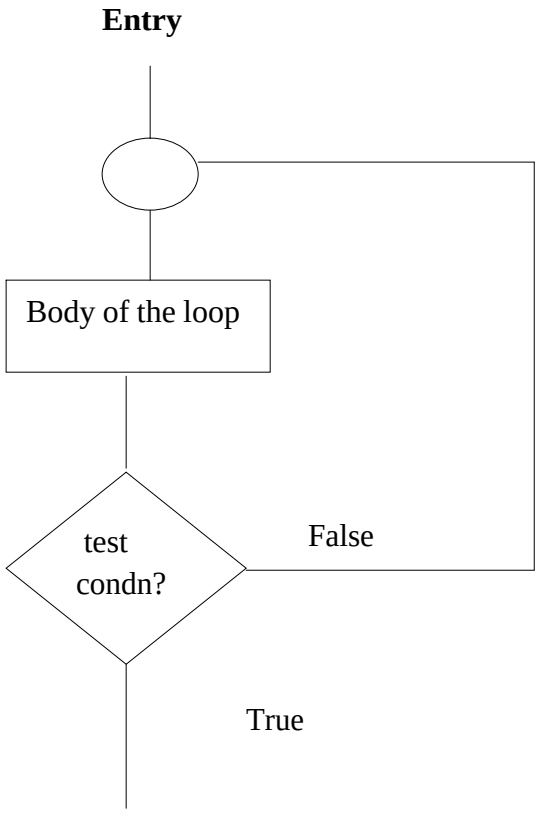
Depending on the position of the control statements in the loop, a control structure may be classified either as an *entry-controlled loop* or as the *exit-controlled loop.*



Eg:

```
main()
{
        double x, y;
        read:
        scanf("%f",&x);
        if(x < 0) goto read;
        y = sqrt(x);
        printf("%f %f \n",x, y);
        goto read;
}
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com



The C language provides for three loop constructs for performing loop operations. They are

- The **while** statement
- The **do** statement
- The **for** statement

THE WHILE STATEMENT

The basic format of the **while** statement is

```
while(test condition)
{
  body of the loop
}
```

The **while** is an *entry–controlled* loop statement. The *test-condition* is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test-condition is once again evaluated and if it is true, the body is executed once again. This process of repeated execution of the body continues until the test-condition finally becomes *false* and the control is transferred out of the loop.

Eg:

```
-----------
-----------
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
sum = 0;
n = 1;
while(n <= 10)
{
   sum = sum + n* n;
   n = n + 1;
 }
 printf("sum = %d \n",sum);
 ----------
 ----------
```

THE DO STATEMENT

In while loop the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt. Such situations can be handled with the help of the **do** statement.

```
do
{
  body of the loop
}
while(test condition);
```

Since the *test-condition* is evaluated at the bottom of the loop, the **do…..while** construct provides an *exit-controlled* loop and therefore the body of the loop is *always executed at least once.*

Eg:

```
----------
do
{
   printf("Input a number\n");
   number = getnum();
 }
 while(number > 0);
         ----------
```

Check Your Progress

Ex 1) Distinguish between while and do-while loop.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

2) When **switch** statement will be very useful?

THE FOR STATEMENT

Simple 'for' Loops

The **for** loop is another *entry-controlled* loop that provides a more consise loop control structure. The general form of the **for** loop is

```
for(initialization ; test-condition ; increment
{
        body of the loop
}
```

The execution of the **for** statement is as follows:

- *Initialization* of the *control variables* is done first.
- The value of the control variable is tested using the *test-condition.* If the condition is *true*, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.
- When the body of the loop is executed, the control is transferred back to the **for** statement after evaluating the last statement in the loop. Now, the control variable is either incremented or decremented as per the condition.

Eg 1)
```
for(x = 0; x <= 9; x = x + 1)
  {
     printf)"%d",x);
  }
  printf("\n");
```

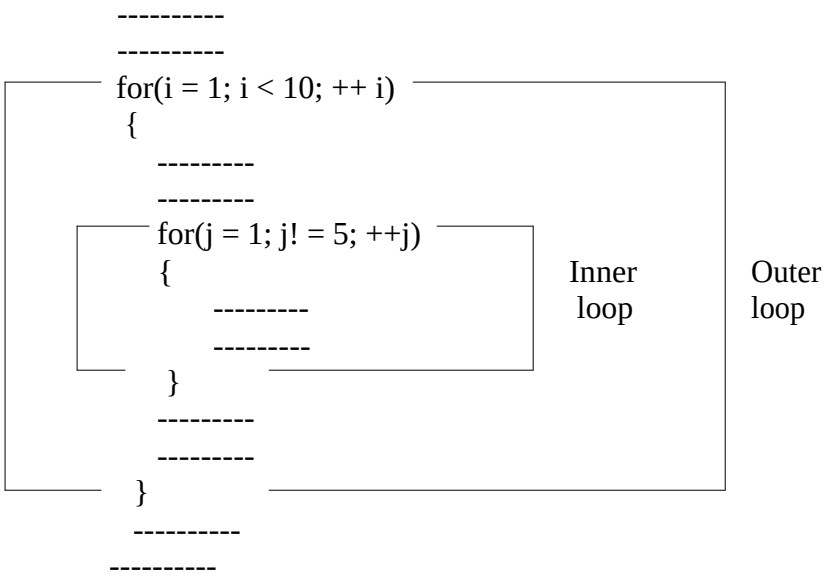
The multiple arguments in the increment section are possible and separated by *commas.*

Eg 2)
```
sum = 0;
  for(i = 1; i < 20 && sum <100; ++i)
  {
      sum =sum + i;
      printf("%d %d \n",sum);
  }
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Nesting of For Loops

C allows one **for** statement within another **for** statement.



Eg:

```
----------
----------
for(row = 1; row <= ROWMAX; ++row)
 {
    for(column = 1; column < = COLMAX; ++column)
     {
        y = row * column;
        printf("%4d", y);
     }
     printf("\n");
 }
 ----------
 ----------
```

JUMPS IN LOOPS

C permits a jump from one statement to another within a loop as well as the jump out of a loop.

Jumping out of a Loop

An early exit from a loop can be accomplished by using the **break** statement or the **goto** statement.

When the **break** statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the **break** would only exit from the loop containing it. That is, the **break** will exit only a single loop.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Skipping a part of a Loop

Like the **break** statement, C supports another similar statement called the **continue** statement. However, unlike the **break** which causes the loop to be terminated, the **continue,** as the name implies, causes the loop to be *continued* with the *next iteration* after *skipping* any statements in between. The **continue** statement tells the compiler, "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION". The format of the **continue** statement is simply

```
continue;
```

The use of **continue** statement in loops.

(a)
```
while(test-condition)
    {
     ---------
     if(--------)
        continue;
     ----------
     ----------
    }
```

(b)
```
do
{
---------
if(-------)
continue;
---------
---------
}while(test-condition);
```

(c)
```
for(initialization; test condition; increment)
    {
     ---------
    if(---------)
      continue;
    ---------
    ---------}
```

5.6 LET US SUM UP

In this lesson, we have learnt about

- **Control Statements**
- **Conditional Statements**
- **The Switch Statement**
- **Unconditional Statements**
- **Decision Making and Looping**

This watermark does not appear in the registered version - http://www.clicktoconvert.com

These concepts can now be used in our programs with ease and without any ambiguity, as they play crucial roles in many real-life problems.

5.7 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own

1) What do you understand by Control Statements?
2) Explain the Conditional Statements with simple examples.
3) Explain with syntax the **switch** statement
4) What do you mean by unconditional statements? Give examples.
5) Explain the looping statements in detail with examples.

5.8 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1)

while	**do-while loop**.
Top tested	Bottom tested
When condition fails no execution	Execution Minimum once even condition fails

2) When multiple alternatives are available the switch statement will be very useful than nested if.

5.9 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

UNIT II

LESSON – 6

ARRAYS & STRINGS

6.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about arrays with their different types and handling of character strings. We are also going to learn about the arithmetic operations that can be performed on strings and the string-handling functions in detail.

After learning this lesson, we should be able to

- understand the concepts behind arrays
- handle the character strings
- perform arithmetic operations on characters
- identify the appropriate string handling functions.

6.1 INTRODUCTION

An array is a group of related data items that share a common name. For instance, we can define array name **salary** to represent a set of salary of a group of employees. A particular value is indicated by writing a number called index number or subscript in brackets after the array name.

Eg: **salary[10]**

6.2 ONE DIMENSIONAL ARRAY

An array with a single subscript is known as one dimensional array.

Eg: 1) **int number[5];**

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The values to array elements can be assigned as follows.

Eg: 1) **number[0] = 35;**
number[1] = 40;
number[2] = 20;

Declaration of Arrays

The general form of array declaration is

type variable-name[size];

The type specifies the type of element that will be contained in the array, such as **int, float,** or **char** and the size indicates the maximum number of elements that can be stored inside the array.

Eg: **1) float height[50];**
2) int group[10];
3)char name[10];

Initialization of Arrays

The general form of initialization of arrays is:

static type array-name[size] = {list of values};

Eg:1) **static int number[3] = {0,0};**

If the number of values in the list is less than the number of elements, then only that many elements will be initialized. The remaining elements will be set to zero automatically. Initialization of arrays in C suffers two drawbacks

- ❖ There is no convenient way to initialize only selected elements.
- ❖ There is no shortcut method for initializing a large number of array elements like the one available in FORTRAN.

We can use the word ‘static’ before type declaration. This declares the variable as a static variable.

Eg : 1) **static int counter[] = {1,1,1};**
2) **……**
……
for(i =0; i < 100; i = i+1)
{
if i < 50
sum[i] = 0.0;
else
sum[i] = 1.0;
}
…….
…….

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Program

```
/*Program showing one-dimensional array*/
main()
{
int i;
float x[10],value,total;
printf("Enter 10 real numbers:\n");
for(i =0; i < 10; i++)
{
scanf("%f",&value);
x[i] = value;
}
total = 0.0;

for(i = 0; i < 10; i++)
total = total + x[i] * x[i];
printf("\n");
for(i = 0; i < 10; i++)
printf("x[%2d] = %5.2f \n",i+1,x[i]);
printf("\nTotal = %.2f\n",total);
}
```

OUTPUT

```
Enter 10 real numbers:
1.1  2.2  3.3  4.4   5.5  6.6  7.7  8.8  9.9  10.10

x[1]  =  1.10
x[2]  =  2.20
x[3]  =  3.30
x[4]  =   4.40
x[5]  =  5.50
x[6]  =  6.60
x[7]  =  7.70
x[8]  =  8.80
x[9]  =  9.90
x[10] = 10.10
Total  = 446.86
```

6.3 TWO-DIMENSIONAL ARRAYS

Two-dimensional arrays are declared as follows

```
type array-name[row_size][column_size];
```

Eg: **product[i][j] = row * column;**

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Program

```
/*Program to print multiplication table*/
#define    ROWS            5
#define     COLUMNS      5
main()
{
int    row, column, product[ROWS][COLUMNS];
int i, j;
printf("Multiplication table\n\n:") ;
printf("  ");
for(j = 1; j <= COLUMNS; j++)
printf("%4d",j);
printf("\n");
printf("                                \n");
for(i = 0; i < ROWS; i++)
{
row = i + 1;
printf("%2d|", row);
for(j =1; j <= COLUMNS; j++)
{
column = j;
product[i][j] = row * column;
printf("%4d", product[i][j]);
}
printf("\n");
}
}
```

OUTPUT

Multiplication Table

```
1    2   3   4    5
-------------------------
1  |   1   2   3   4    5
2  |   2   4   6   8   10
3  |   3   6   9  12   15
4  |   4   8  12  16   20
5  |   5  10  15  20   25
```

Check Your Progress

Ex 1) Give examples for one dimensional array.

Ex 2) Give examples for two dimensional array.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

6.4 MULTIDIMENSIONAL ARRAY

C allows arrays of three or more dimensions. The exact limit is determined by the compiler. The general form of a multidimensional array is

type **array_name**[s1][s2][s3]…s[m];

Eg: 1. **int survey[3][5][12];**
2. **float table[5][4][5][3];**

6.5 HANDLING OF CHARACTER STRINGS

INTRODUCTION

A string is a array of characters. Any group of characters(except the double quote sign) defined between double quotation marks is a constant string.
Eg: 1) “Man is obviously made to think”

If we want to include a double quote in a string, then we may use it with the back slash.
Eg: **printf(“\”well done!\””);**
will output
“well done!”
The operations that are performed on character strings are

- Reading and writing strings.
- Combining strings together.
- Copying one string to another.
- Comparing strings for equality.
- Extracting a portion of a string.

6.6 DECLARING AND INITIALIZING STRING VARIABLES

A string variable is any valid C variable name and is always declared as an array. The general form of declaration of a string variable is

char string_name[size];

Eg: **char city[10];**
char name[30];

When the compiler assigns a character string to a character array, it automatically supplies a null character (‘\0’) at the end of the string. Therefore, the size should be equal to the maximum number of characters in the string plus one. C permits a character array to be initialized in either of the following two forms

static char city[9] = “NEW YORK”;
static char city[9] = {‘N’, ‘E’, ‘W’, ‘ ‘, ‘Y’, ‘O’, ‘R’, ‘K’, ‘\0’};

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Reading Words

The familiar input function scanf can be used with %s format specification to read in a string of characters.

Eg: **char address[15];**
scanf("%s",address);

Program

```
/*Reading a series of words using scanf function*/
main()
{
char word1[40],word2[40],word3[40],word4[40];
printf("Enter text:\n");
scanf("%s %s",word1, word2);
scanf("%s", word3);
scanf("%s",word4);
printf("\n");
printf("word1 = %s \n word2 = %s \n",word1, word2);
printf("word3 = %s \n word4 = %s \n",word3, word4);
}
```

OUTPUT
Enter text:
Oxford Road, London M17ED
Word1 = Oxford
Word2 = Road
Word3 = London
Word4 = M17ED
Note: Scanf function terminates its input on the first white space it finds.

Reading a Line of Text

It is not possible to use scanf function to read a line containing more than one word. This is because the scanf terminates reading as soon as a space is encountered in the input. We can use the getchar function repeatedly to read single character from the terminal, using the function **getchar**. Thus an entire line of text can be read and stored in an array.

Program

```
/*Program to read a line of  text from terminal*/
#include<stdio.h>
main()
{
char line[81],character;
int c;
c = 0;
printf("Enter text. Press<Return>at end \n");
do
{
character = getchar();
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
line[c] = character;
c++;
}
while(character != '\n');
c = c-1;
line[c] = '\0';
printf("\n %s \n",line);
}
```

OUTPUT
Enter text. Press<Return>at end
Programming in C is interesting
Programming in C is interesting

WRITING STRINGS TO SCREEN

We have used extensively the printf function with %s format to print strings to the screen. The format %s can be used to display an array of characters that is terminated by the null character.

For eg, the statement

printf("%s", name);

can be used to display the entire contents of the array **name**.

6.7 ARITHMETIC OPERATIONS ON CHARACTERS

C allows us to manipulate characters the same way we do with numbers. Whenever a character constant or character variable is used in an expression, it is automatically converted into integer value by the system.

For eg, if the machine uses the ASCII representation, then,

```
x = 'a';
printf("%d \n",x);
```

will display the number 97 on the screen.

The C library supports a function that converts a string of digits into their integer values. The function takes the form

x = atoi(string)

PUTTING STRINGS TOGETHER

Just as we cannot assign one string to another directly, we cannot join two strings together by the simple arithmetic addition. That is, the statements such as

```
string3 = string1 + string2;
string2 = string1 + "hello";
```

are **not** valid. The characters from string1 and string2 should be copied into string3 one after the other. The process of combining two strings together is called concatenation.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

COMPARISON OF TWO STRINGS

C does not permit the comparison of two strings directly. That is, the statements such as

if(name1 == name2)
if(name == "ABC");

are **not** permitted. It is therefore necessary to compare the two strings to be tested, character by character. The comparison is done until there is a mismatch or one of the strings terminate into a null character, whichever occurs first.

6.8 STRING - HANDLING FUNCTIONS

C library supports a large number of string-handling functions that can be used to carry out many of the string manipulation activities. Following are the most commonly used string-handling functions.

Function	Action
strcat( )	Concatenates two strings
strcmp( )	Compares two strings
strcpy( )	Copies one string over another
strlen( )	Finds the length of the string

strcat( ) Function

The strcat function joins two strings together. It takes the following form

strcat(string1,string2**);**

Eg: **strcat(part1, "GOOD");**

strcat(strcat(string1,string2),string3);

Here three strings are concatenated and the result is stored in string1.

strcmp( ) Function

It is used to compare two strings identified by the arguments and has a value 0 if they are equal.It takes the form:

strcmp(string1,string2);

Eg: 1) **strcmp(name1,name2);**
2) strcmp(name1,"john";
3) strcmp("ram", "rom");

This watermark does not appear in the registered version - http://www.clicktoconvert.com

strcpy( ) Function

This function works almost like a string assignment operator. It takes the form

```
strcpy(string1,string2);
```

This assigns the content of string2 to string1.

Eg: 1) **strcpy(city, "DELHI");**
2) strcpy(city1,city2);

strlen( ) Function

This function counts and returns the number of characters in a string.

```
n = strlen(string);
```

Program

```
/*Illustration of string-handling functions*/
#include<string.h>
main()
{
char s1[20],s2[20],s3[20];
int x, l1, l2, l3;
printf("Enter two string constants \n");
printf("?");
scanf("%s %s", s1, s2);
x = strcmp(s1, s2);
if(x != 0)
printf("Strings are not equal \n");
strcat(s1, s2);
else
printf("Strings are equal \n");
strcpy(s3,s1);
l1 = strlen(s1);
l2 = strlen(s2);
l3 = strlen(s3);
printf("\ns1 = %s \t length = %d characters \n",s1, l1);
printf("\ns2= %s \t length = %d characters \n",s2, l2);
printf("\ns3 = %s \t length = %d characters \n",s3, l3);
}
```

OUTPUT

```
Enter two string constants
?   New York
Strings are not equal
s1 = New York          length = 7 characters
s2 = York              length = 4 characters
s3 = New York          length = 7 characters
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
Enter two string constants
? London London
Strings are equal
s1 = London          length = 6 characters
s2 = London          length = 6 characters
s3 = London          length = 6 characters
```

Check Your Progress

Ex 3) String handling functions are available in _____ header file.

6.9 LET US SUM UP

In this lesson, we have learnt about

- what we mean by arrays
- one, two and multidimensional arrays
- character strings
- declaring and initializing string variables
- arithmetic operations on characters
- string handling functions

The concepts discussed in this lesson can be very useful in many programs, as arrays constitute an important area in the programming domain.

6.10 LESSON-END ACTIVITIES

Try to find the answers to the following exercises on your own

1) What do you mean by array?
2) How will you declare one dimensional array? Give an example.
3) How will you declare two dimensional arrays? Give an example.
4) What do you understand by multidimensional array?
5) Describe the String handling functions of C.

6.11 MODEL ANSWERS TO CHECK YOUR PROGRESS

[Answers vary for Ex 1 and Ex 2]

```
Ex 1)  int strength[50];
       int x[100];
Ex 2)  int x[3][3];
       float a[5][5]
Ex 3) string.h
```

6.12 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON - 7

USER-DEFINED FUNCTIONS

7.0 AIMS AND OBJECTIVES

In this lesson, we are going to study about the need for user-defined functions, the form of C functions, the category of functions, handling of non-integer functions, recursion and functions with arrays.

After studying this lesson, we should be able to

- understand the need for user-defined functions
- the form of C functions
- category of functions
- handle non-integer functions
- understand recursion
- use functions with arrays

7.1 INTRODUCTION

C functions can be classified into two categories, namely, library functions and user-defined functions. **Main** is an example of user-defined functions, printf and scanf belong to the category of library functions. The main difference between these two categories is that library functions are not required to be written by us whereas a user-defined function has to be developed by the user at the time of writing the program.

7.2 NEED FOR USER-DEFINED FUNCTIONS

- It facilitates top-down modular programming.
- The length of the source program can be reduced by using functions at appropriate places.
- It is easy to locate and isolate a faulty function for further investigations.
- A function can be used by many other programs

This watermark does not appear in the registered version - http://www.clicktoconvert.com

7.3 THE FORM OF C FUNCTIONS

All functions have the form

```
Function-name(argument list)
argument declaration;
{
 local variable declarations;
 executable statement-1;
 executable statement-2;
 ……
 ……
 return(expression);
}
```

A function that does nothing may not include any executable statements. For eg:

do_nothing( ) { }

RETURN VALUES AND THEIR TYPES

The return statement can take the form:

```
return
or
return(expression);
```

Eg:**if(x <= 0)**
return(0);
else
return(1);

CALLING A FUNCTION

A function can be called by simply using the function name in the statement.

```
Eg:main()
{
int  p;
p = mul(10,5);
printf("%d \n", p);
}
```

When the compiler executes a function call, the control is transferred to the function mul(x,y).The function is then executed line by line as described and the value is returned, when a return statement is encountered. This value is assigned to p.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

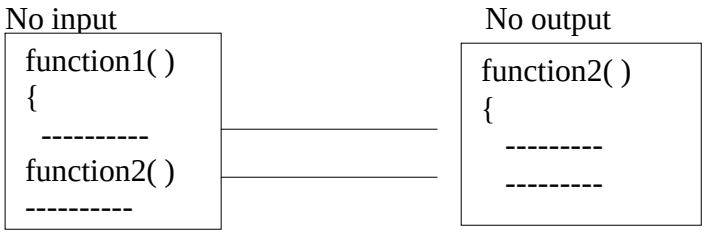
7.4 CATEGORY OF FUNCTIONS

A function may belong to one of the following categories.

1) Functions with no arguments and no return values.
2) Functions with arguments and no return values.
3) Functions with arguments and return values.

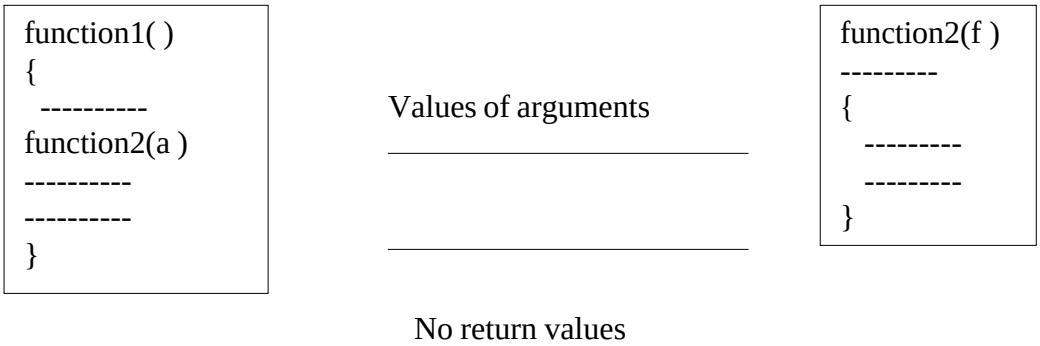
NO ARGUMENTS AND NO RETURN VALUES

A function does not receive any data from the calling function. Similarly, it does not return any value.



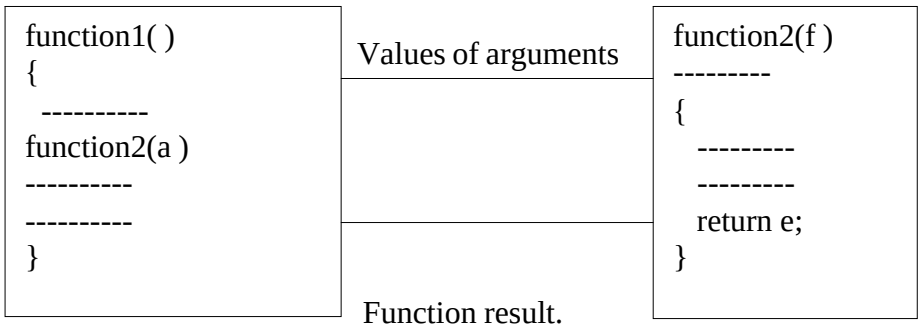
ARGUMENTS BUT NO RETURN VALUES

The nature of data communication between the calling function and the called function with arguments but no return values is shown in the diagram.



ARGUMENTS WITH RETURN VALUES

Here there is a two-way data communication between the calling and the called function.



This watermark does not appear in the registered version - http://www.clicktoconvert.com

7.5 HANDLING OF NON-INTEGER FUNCTIONS

We must do two things to enable a calling function to receive a non integer value from a called function:

1. The explicit type-specifier, corresponding to the data type required must be mentioned in the function header. The general form of the function definition is

```
type specifier function name(argument list)
argument declaration;
{
   function statements;
  }
```

2. The called function must be declared at the start of the body in the calling function.

NESTING OF FUNCTIONS

C permits nesting of functions freely. main can call function1,which calls function2,which calls function3,…..and so on.

7.6 RECURSION

When a function in turn calls another function a process of 'chaining' occurs. **Recursion** is a special case of this process, **where a function calls itself**.

Eg:1)
```
main()
{
  printf("Example for recursion");
  main();
 }
```

Check your Progress

Ex 1) Is main() a user defined function?
2) Functions must return always some value (True/False)
3) Can a function call itself ? If so, what do you infer?

7.7 FUNCTIONS WITH ARRAYS

To pass an array to a called function, it is sufficient to list the name of the array, without any subscripts, and the size of the array as arguments.

Eg:1) **largest(a,n);**

This watermark does not appear in the registered version - http://www.clicktoconvert.com

7.8 LET US SUM UP

In this lesson , we have studied about

- the need for user-defined functions
- the form of C functions
- the category of functions
- handling of non-integer functions
- recursion
- how to use functions with arrays

7.9 LESSON-END ACTIVITIES

Try to find the answers to the following exercises on your own

1) What is the need for user-defined functions?
2) Bring out the form of C functions
3) Describe the category of functions
4) Explain how will you handle non-integer functions
5) What is meant by Recursion? Give an example.
6) Explain how will you use functions with arrays.

7.10 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) Yes , main() is a user defined function
2) Functions must return always some value (False)
3) A function can call itself. It is allowed. It is known as Recursive call.

7.11 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON – 8

STORAGE CLASSES

8.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about automatic variables(local/internal),external variables, static variables, register variables and ANSI C functions.

After learning this lesson, we should be able to

- understand automatic variables(local/internal)
- identify external variables
- know about static variables
- gain knowledge about register variables
- understand what we mean by ANSI C functions

8.1 INTRODUCTION

A variable in C can have any one of the four storage classes.

1. Automatic variables.
2. External variables.
3. Static variables.
4. Register variables.

8.2 AUTOMATIC VARIABLES (LOCAL/INTERNAL)

Automatic variables are declared inside a function in which they are to be utilized. They are created when a function is called and destroyed automatically when the function is exited.

```
Eg:main()
{
int number;
---------
---------
}
```

We may also use the keyword **auto** to declare automatic variables explicitly.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

8.3 EXTERNAL VARIABLES

Variables that are both alive and active throughout the entire program are known as external variables. They are also known as global variables.

Eg: **int number;**
float length = 7.5;

```
main()
{
-------
-------
}
function1( )
{
-------
-------
}
function2( )
{
-------
-------
}
```

The keyword **extern** can be used for explicit declarations of external variables.

8.4 STATIC VARIABLES

As the name suggests, the value of a static variable persists until the end of the program. A variable can be declared static using the keyword **static.**

Eg:1) **static int x;**
2) static int y;

8.5 REGISTER VARIABLES

We can tell the compiler that a variable should be kept in one of the machine's registers, instead of keeping in the memory. Since a register access is much faster than a memory access, keeping the frequently accessed variables in the register will lead to faster execution of programs. This is done as follows:

register int count;

Check Your Progress

Ex 1) Is the keyword auto is compulsory in declaration?

---

2) What is the other name for external variables?

---

3) Can we declare all variables as register variables?

This watermark does not appear in the registered version - http://www.clicktoconvert.com

8.6 ANSI C FUNCTIONS

The general form of ANSI C function is

```
data-type function-name(type1 a1,type2 a2,…..typeN aN)
{
  --------
  --------     (body of the function)
  --------
}
```

Eg: 1) **double funct(int a, int b, double c)**

Function Declaration

The general form of function declaration is

```
data-type function-name(type1 a1,type2 a2,…..typeN aN)
```

```
Eg:main()
{
float a, b, x;
float mul(float length,float breadth);/*declaration*/
---------
---------
x = mul(a,b);
}
```

8.7 LET US SUM UP

In this lesson, we have learnt about

- the automatic variables(local/internal)
- identification of external variables
- the use of static variables
- importance of register variables
- ANSI C functions

8.8 LESSON-END ACTIVITIES

Try to find the answers to the following exercises on your own

1) List out the four storage classes of C
2) Explain automatic variables.
3) What do you understand by external variables?
4) Why we need static variables?
5) Highlight the importance of register variables.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

8.9 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) No, not necessary.
2) Global variables
3) No. Limited Registers available. Frequently used variables like loop indexes can be declared as register variables.

8.10 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON – 9

POINTERS

9.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about pointers, accessing the address of a variable, accessing a variable through its pointer, pointer expressions, pointers and arrays, pointers and character strings.

After learning this lesson, we should be able to

- understand pointers
- access the address of a variable
- access a variable through its pointer
- use pointer expressions
- use pointers and arrays
- gain knowledge about pointers and character strings

9.1 INTRODUCTION

Pointers are another important feature of C language. Although they may appear a little confusing for a beginner, they are powerful tool and handy to use once they are mastered. There are a number of reasons for using pointers.

1. A pointer enables us to access a variable that is defined outside the function.
2. Pointers are more efficient in handling the data tables.
3. Pointers reduce the length and complexity of a program.
4. They increase the execution speed.
5. The use of a pointer array to character strings result in saving of data storage space in memory.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

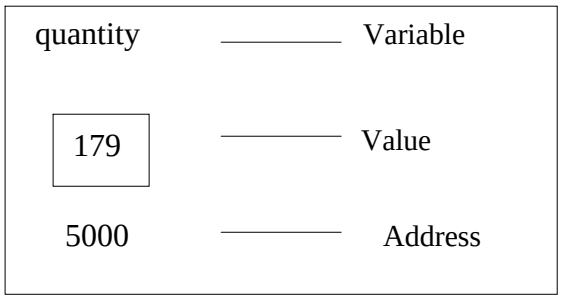
9.2 UNDERSTANDING POINTERS

Whenever we declare a variable, the system allocates, somewhere in the memory, an appropriate location to hold the value of the variable. Since, every byte has a unique address number, this location will have its own address number.

Consider the following statement:

int quantity = 179;

This statement instructs the system to find a location for the integer variable quantity and puts the value 179 in that location. Assume that the system has chosen the address location 5000 for quantity. We may represent this as shown below.



Representation of a variable

During execution of the program, the system always associates the name quantity with the address 5000. To access the value 179 we use either the name quantity or the address 5000. Since memory addresses are simply numbers, they can be assigned to some variables which can be stored in memory, like any other variable. **Such variables that hold memory addresses are called pointers. A pointer is, therefore, nothing but a variable that contains an address which is a location of another variable in memory.**

Since a pointer is a variable, its value is also stored in the memory in another location. Suppose, we assign the address of quantity to a variable p. The link between the variables p and quantity can be visualized as shown below. The address of p is 5048.

Variable	***Value***	***Address***
quantity	179	5000
p	5000	5048

Pointer as a variable

Since the value of the variable p is the address of the variable quantity, we may access the value of quantity by using the value of p and therefore, we say that the variable p 'points' to the variable quantity. Thus, p gets the name 'pointer'.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

9.3 CESSING THE ADDRESS OF A VARIABLE

The actual location of a variable in the memory is system dependent and therefore, the address of a variable is not known to us immediately. However we determine the address of a variable by using the operand & available in C. The operator immediately preceding the variable returns the address of the variable associated with it. For example, the statement

p = &quantity;

would assign the address 5000(the location of quantity) to the variable p. The &operator can be remembered as 'address of'.

The & operator can be only used with a simple variable or an array element. The following are **illegal** use of address operator:

1. &125 (pointing at constants).
2. int x[10];
   &x (pointing at array names).
3. &(x+y) (pointing at expressions).

If x is an array ,then expressions such as

&x[0] and &x[i + 3]

are **valid** and represent the addresses of 0^{th} and $(i + 3)^{th}$ elements of x.

The program shown below declares and initializes four variables and then prints out these values with their respective storage locations.

```
Program
/*************************************************************************/
/*              ACCESSING ADDRESSES OF VARIABLES                         */
/*************************************************************************/
main()
{
        char a;
        int x;
        float p, q;
         a  = 'A';
         x  = 125;
         p  = 10.25 , q = 18.76;
         printf("%c  is stored as addr %u . \n", a, &a);
         printf("%d  is stored as addr %u . \n",x , &x);
         printf("%f  is stored as addr %u . \n", p, &p);
         printf("%f  is stored as addr %u . \n", q, &q);
 }

 A is stored at addr 44336
125 is stored at addr  4434
10.250000 is stored at addr 4442
18.760000 is stored at addr 4438.
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

DECLARING AND INITIALIZING POINTERS

Pointer variables contain addresses that belong to a separate data type, which must be declared as pointers before we use them. The declaration of the pointer variable takes the following form:

data type *pt _name;

This tells the compiler three things about the variable pt _name:

1. The asterisk(*) tells that the variable pt _name.
2. pt _name needs a memory location.
3. pt_name points to a variable of type data type.

Example:

```
1. int *p;
2. float *x;
```

Once a pointer variable has been declared, it can be made to point to a variable using an assignment operator such as

p= &quantity;

Before a pointer is initialized it should not be used.

Ensure that the pointer variables always point to the corresponding type of data.

Example:

```
float a, b;
int x, *p;
p = &a;
b = *p;
```

will result in erroneous output because we are trying to assign the address of a float variable to an integer pointer. When we declare a pointer to be of int type, the system assumes that any address that a pointer will hold will point to an integer variable.

Assigning an absolute address to a pointer variable is prohibited. The following is wrong.

```
int *ptr;
….
ptr = 5368;
….
….
```

A pointer variable can be initialized in its declaration itself. For example,

```
int  x, *p = &x;
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

is perfectly valid. It declares x as an integer variable and p as a pointer variable and then initializes p to the address of x. The statement

int *p = &x, x;

is not valid.

9.4 ACCESSING A VARIABLE THROUGH ITS POINTER

To access the value of the variable using the pointer, another unary operator *(asterisk), usually known as the indirection operator is used. Consider the following statements:

```
int quantity, *p, n;
quantity = 179;
p= &quantity;
n= *p;
```

The statement n=*p contains the indirection operator *. When the operator * is placed before a pointer variable in an expression (on the right-hand side of the equal sign), the pointer returns the value of the variable of which the pointer value is the address. In this case, *p returns the value of the variable quantity, because p is the address of the quantity. The * can be remembered as 'value at address'. Thus the value of n would be 179. The two statements

```
p= &quantity;
n= *p;
```

are equivalent to

```
n= *&quantity;
```

which in turn is equivalent to

```
n= quantity;
```

The following program illustrates the distinction between pointer value and the value it points to and **the use of indirection operator(*) to access the value pointed to by a pointer.**

Check Your Progress

Ex 1) Specify a few reasons to use Pointers.

Ex 2) Pointer variable stores _____

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
Program  ACCESSING VARIABLES USING POINTERS

main( )
{
        int  x, y ;
        int  * ptr;
        x =10;
        ptr = &x;
        y = *ptr;
        printf ("Value of x is  %d \n\n",x);
      printf ("%d is stored  at addr %u \n" , x, &x);
       printf ("%d is stored  at addr %u \n" , *&x, &x);
       printf ("%d is stored  at addr %u \n" , *ptr, ptr);
       printf ("%d is stored  at addr %u \n" , y, &*ptr);
       printf ("%d is stored  at addr %u \n" , ptr, &ptr);
       printf ("%d is stored  at addr %u \n" , y, &y);
       *ptr= 25;
       printf("\n Now x = %d \n",x);
 }
```

pointed to by the pointer ptr to y.

Note the use of assignment statement

*ptr=25;

This statement puts the value of 25 at a memory location whose address is the value of ptr. We know that the value of ptr is the address of x and therefore the old value of x is replaced by 25. This, in effect, is equivalent to assigning 25 to x. This shows how we can change the value of a variable indirectly using a pointer and the indirection operator.

9.5 POINTER EXPRESSIONS

Like other variables, pointer variables can be used in expressions. For example, if p1 and p2 are properly declared and initialized pointers , then the following statements are valid.

1) y = *p1* *p2; same as (* p1) * (* p2)

2) sum = sum + *p1;

3) z = 5* - *p2/ *p1; same as (5 * (-(* p2)))/(* p1)

4) *p2 = *p2 + 10;

Note that there is a blank space between / and * in the statement 3 above.

C allows us to add integers to or subtract integers from pointers , as well as to subtract one pointer from another. p1 + 4, p2 – 2 and p1 – p2 are all allowed. If p1 and p2 are both

This watermark does not appear in the registered version - http://www.clicktoconvert.com

pointers to the same array, then p2 – p1 gives the number of elements between p1 and p2. We may also use short-hand operators with the pointers.

p1++;

--p2;

Sum += *p2;

Pointers can also be compared using the relational operators. Pointers cannot be used in division or multiplication. Similarly two pointers cannot be added.

A program to illustrate the use of pointers in arithmetic operations.

```
Program  POINTER EXPRESSIONS
main ( )
{
        int a, b, *p1,* p2, x, y, z;
        a   =  12;
        b   =  4;
        p1 =  &a;
        p2 =  &b;
        x   =  *p1 *   *p2 – 6;
        y   =  4* - *p2  /  *p1  +  10;



        printf(“Address of a  =  %u\n”, p1);

        printf(“Address of b  =  %u\n”, p2);
        printf(“\n”);
        printf(“a  = %d,  b =   %d\n”, a, b);
        printf(“x  = %d,  y =   %d\n”, x, y);
        *p2 =  *p2 + 3;
        *p1 =   *p2 - 5;
         z      =   *p1  *  *p2 – 6;
        printf(“\n a = %d, b = %d,” , a , b);
        printf(“\n z = %d\n” , z);
}
```

POINTER INCREMENTS AND SCALE FACTOR

We have seen that the pointers can be incremented like

p1 = p2 + 2;

p1 = p1 + 1;

and so on .

Remember, however, an expression like

p1++;

This watermark does not appear in the registered version - http://www.clicktoconvert.com

will cause the pointer p1 to point to the next value of its type.

That is, when we increment a pointer, its value is increased by the length of the data type that it points to. This length is called the scale factor.

The number of bytes used to store various data types depends on the system and can be found by making use of size of operator. For example, if x is a variable, then size of(x) returns the number of bytes needed for the variable.

9.6 POINTERS AND ARRAYS

When an array is declared, the compiler allocates a base address and sufficient amount of storage to contain all the elements of array in contiguous memory location. The base address is the location of the first element (index 0) of the array. The compiler also defines the array name as a constant pointer to the first element. Suppose we declare an array x as follows:

static int x[5] = {1,2,3,4,5};

Suppose the base address of x is 1000and assuming that each integer requires two bytes, the five elements will be stored as follows:

Elements	x[0]	x[1]	x[2]	x[3]	x[4]
Value	1	2	3	4	5
Address	1000	1002	1004	1006	1008

The name x is defined as a constant pointer pointing to the first element x[0] and therefore value of x is 1000, the location where x[0] is stored . That is ,

x = &x[0] =1000

Accessing array elements using the pointer

Pointers can be used to manipulate two-dimensional array as well. An element in a two-dimensional array can be represented by the pointer expression as follows:

((a+i)+j) or *(*(p+i)+j)

The base address of the array a is &a[0][0] and starting at this address, the compiler allocates contiguous space for all the elements, row-wise. That is, the first element of the second row is placed immediately after the last element of the first row, and so on.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

A program using Pointers to compute the sum of all elements stored in an array is presented below:

POINTERS IN ONE-DIMENSONAL ARRAY

```
main ( )
{
        int  *p, sum , i
        static  int x[5] = {5,9,6,3,7};
        i = 0;
        p = x;
        sum  = 0;
        printf("Element    Value   Address \n\n");
        while(i < 5)
        {
           printf("  x[%d}  %d  %u\n", i, *p, p);
           sum = sum + *p;
           i++, p++;
        }
        printf("\n  Sum   =  %d  \n",  sum);
        printf("\n  &x[0] = %u  \n", &x[0]);
        printf("\n  p        = %u \n", p);
        }
```

Output

Element	Value	Address
X[0]	5	166
X[1]	9	168
X[2]	6	170
X[3]	3	172
X[4]	7	174

Sum = 55
&x[0] = 166
p = 176

9.7 POINTERS AND CHARACTER STRINGS

We know that a string is an array of characters, terminated with a null character. Like in one-dimensional arrays, we can use a pointer to access the individual characters in a string. This is illustrated in the program given below.

```
/* Pointers and character Strings */
main()
  {
     char * name;
     int length;
    char * cptr = name;
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
        name = "DELHI";
        while ( *cptr != '\0')
         {
           printf( "%c is stored at address %u \n", *cptr,cptr);
           cptr++;
         }

         length = cptr-name;
         printf("\n length = %d \n", length);
}
```

String handling by pointers

One important use of pointers in handling of a table of strings. Consider the following array of strings:

char name[3][25];

This says that name is a table containing three names, each with a maximum length of 25 characters (including null character).

Total storage requirements for the name table are 75 bytes.

Instead of making each row a fixed number of characters , we can make it a pointer to a string of varying length.

For example,

```
static char *name[3] = { "New zealand",
                         "Australia",
                         "India"
                       };
```

declares name to be an array of three pointers to characters, each pointer pointing to a particular name as shown below:

name[0] → New Zealand

name[1] → Australia

name[2] → India

This watermark does not appear in the registered version - http://www.clicktoconvert.com

9.8 LET US SUM UP

In this lesson, we have learnt about

- the concepts behind pointers
- accessing the address of a variable
- accessing a variable through its pointer
- how to use pointer expressions
- applying pointers and arrays
- using pointers and character strings

9.9 LESSON-END ACTIVITIES

Try to find the answers to the following exercises on your own

1) What do you mean by Pointers?
2) How will you declare and initialize pointers?
3) Explain the method of accessing a variable through its Pointer.
4) Write short notes on Pointers and Arrays.

9.10 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1)

- A pointer enables us to access a variable that is defined outside the function.
- Pointers are more efficient in handling the data tables.
- Pointers reduce the length and complexity of a program.
- They increase the execution speed.
- The use of a pointer array to character strings result in saving of data storage space in memory.

Ex 2) Pointer variable stores address of the variable

9.11 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON - 10

POINTERS AND FUNCTIONS

10.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about pointers as function arguments, pointers and structures and the preprocessor directives of C programming language.

After learning this lesson, we should be able to

- use pointers as function arguments
- understand pointers and structures
- know about the preprocessor of C

10.1 POINTERS AS FUNCTION ARGUMENTS

```
Program POINTERS AS FUNCTION PARAMETERS
main ( )
{
        int   x , y;
        x = 100;
        y = 200;
        printf("Before exchange : x  =  %d  y  =  %d \n\n ", x , y);
        exchange(&x, &y);
        printf("After exchange :  x  =  %d  y  =  %d \n\n " , x , y);
}
exchange(a, b)
int *a, *b;
{
        int t;
        t = * a;        /*Assign the value at address a to t*/
        * a = * b ;     /*Put the value at b into a*/
        * b = t;        /*Put t into b*/
}
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

In the above example, we can pass the address of the variable a as an argument to a function in the normal fashion. The parameters receiving the addresses should be pointers. The process of calling a function using pointers to pass the addresses of variable is known as call by reference. The function which is called by 'Reference' can change the value of the variable used in the call.

Passing of pointers as function parameters

1. The function parameters are declared as pointers.
2. The dereference pointers are used in the function body.
3. When the function is called, the addresses are passed as actual arguments. Pointers parameters are commonly employed in string functions.

Pointers to functions

A function, like a variable has an address location in the memory. It is therefore, possible to declare a pointer to a function, which can then be used as an argument in another function. A pointer to a function is declared as follows:

type (*fptr)( );

This tells the compiler that fptr is a pointer to a function which returns type value.

A program to illustrate a function pointer as a function argument.

```
Program
POINTERS TO FUNCTIONS
#include <math.h>
#define PI  3.141592
main ( )
{
        double y( ), cos( ), table( );
        printf("Tableof y(x) = 2*x*x-x+1\n\n");
        table(y, 0.0 , 2.0, 0.5);

        printf("\n Table of cos(x) \n\n");
        table(cos, 0.0, PI , 0.5);
}
        double table(f, min, max, step)
        double (*f) ( ), min, max , step;
        {
            double a, value;
             for( a = min; a < = max; a + = step)
             {
                  value = (*f)(a);
                  printf("%5.2f  %10.4f\n", a, value);
              }
        }
        double y(x)
        double x;
        {
                  return (2*x*x-x+1);
        }
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

10.2 POITNTERS AND STRUCTERS

The name of an array stands for the address of its zeroth element.

Consider the following declaration:

```
struct inventory
 {
   char name[30];
   int number;
   float price;
 }  product[2], *ptr;
```

This statement declares product as an array of two elements, each of type of struct inventory and ptr as a pointer to data objects of the type struct inventory.

The assignment

ptr = product;

would assign the address of the zeroth element of product to ptr. Its members can be accessed using the following notation .

ptr → name

ptr → number

ptr → price

Initially the pointer ptr will point to product[0], when the pointer ptr is incremented by one it will point to next record, that is product[1].

We can also use the notation

(*ptr).number

to access the member number.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

A program to illustrate the use of structure pointers.

```
Program POINTERS TO STRUCTURE VARIABLES
struct invent
{
        char *name[20];
        int number;
        float price;
};
main( )
{
        struct invent product[3], *ptr;
        printf("INPUT\n\n");
        for(ptr = product; ptr < product + 3; ptr + +)

                scanf("%s %d %f", ptr → name, &ptr → number , & ptr →
        price);
        printf("\Noutput\n\n");
        ptr = product;
        while(ptr < product +3)

        {
        printf("%-20s %5d  %10.2f\n" ,
                        ptr → name,
                       ptr → number ,
                        ptr →  price);   ptr++;
        }
}
```

While using structure pointers we should take care of the precedence of operators.

For example, given the definition

```
struct
 {
   int count;
   float *p;
 } *ptr;
```

Then the statement

++ptr → count;

increments count, not ptr.

However ,

(++ptr) → count;

increments ptr first and then links count.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

10.3 THE PREPROCESSOR

The Preprocessor, as the name implies, is a program that processes the source code before it passes through the complier. It operates under the control of preprocessor command lines or directives. Preprocessor directives are placed in the source program before the main line. Before the source code passes through the compiler, it is examined by the preprocessor for any preprocessor directives. If there are any, appropriate actions (as per the directives) are taken and then the source program is handed over to the compiler.

Preprocessor directives begin with the symbol # in column one and do not require a semicolon at the end.

Commonly used Preprocessor directives

Directive	**Function**
#define	Defines a macro substitution
#undef	Undefines a macro
#include	specifies the files to be include
# ifdef	Tests for a macro definition
#endif	specifies the end of #if
#ifndef	Tests whether a macro is not defined
#if	Tests a compile time condition
#else	specifies alternatives when #if fails

Preprocessor directives can be divided into three categories

1) Macro substitution directives
2) File Inclusion directives
3) Compiler control directives

Macro Substitution directive

The general form is

```
#define identifier string
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Examples 1) #define COUNT 100 (simple macro substitution)
2) #define CUBE(x) x*x*x (macro with arguments)
3) # define M 5
#define N M+1 (nesting of macros)

File Inclusion directive

This is achieved by

#include "filename" or #include <filename>

Examples 1) #include <stdio.h>
2) #include "TEST.C"

Check Your Progress

Ex 1) Give examples for macro substitution directives

Ex 2) Differentiate #include <…> and #include "…"

Compiler control directives

These are the directives meant for controlling the compiler actions. C preprocessor offers a feature known as conditional compilation, which can be used to switch off or on a particular line or group of lines in a program. Mostly #ifdef and #ifndef are used in these directives.

10.4 LET US SUM UP

In this lesson, we have learnt about

- using pointers as function arguments
- pointers and structures
- the preprocessor of C

10.5 LESSON-END ACTIVITIES

1) Explain the method of using pointers as function arguments
2) How will you use pointers in structures?
3) What is meant by preprocessor? Why is it required?
4) Sketch out the categories of preprocessor directives.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

10.6 MODEL ANSWERS TO CHECK YOUR PROGRESS

[Answers vary for Ex 1]

Ex 1) #define A 10
#define B 20
#define C A+B
#define SQR(x) x*x

Ex 2) #include <…>
Search for the specified entry in standard directories only.

#include "…"

Search the specified entry first in current directory and then in standard directories.

10.7 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

UNIT III

LESSON - 11

STRUCTURES

11.0 AIMS AND OBJECTIVES

This lesson introduces us the definition of structure, giving values to members, structure initialization, comparison of structure variables and arrays of structures with detailed description of concepts associated.

After learning this lesson, we should be able to

- define structure
- differentiate between array and structure
- give values to members
- initialize structure
- compare structure variables
- use arrays of structures

11.1 INTRODUCTION

C supports a constructed data type known as structures, a mechanism for packing data of different types. A structure is a convenient tool for handling a group of logically related data items. For example, it can be used to represent a set of attributes, such as student _ name, roll _ number and marks. The concept of a structure is analogous to that of a ‘record’ in many other languages. More examples of such structures are:

time : seconds, minutes, hours
data : day, month, year
book : author, title, price, year
city : name, country, population

This watermark does not appear in the registered version - http://www.clicktoconvert.com

11.2 STRUCTURE DEFINITION

Unlike arrays, structure must be defined first for their format that may be used later to declare structure variables. Let us use an example to illustrate the process of structure definition and the creation of structure variables. Consider a book database consisting of book name, author, number of pages, and price. We can define a structure to hold this information as follows:

```
struct book _bank
 {
    char title[20];
    char author[15];
    int pages;
    float price;
 };
```

The keyword struct declares a structure to hold the details of four data fields, namely title, author, pages, and price. These fields are called structure elements or members. Each member may belong to different type of data. book _ bank is the name of the structure and is called the structure tag. The tag name may be used subsequently to declare variables that have the tag's structure.

The general format of a structure definition is as follows:

```
struct  tag _ name
  {
       data _ type      member1;
       data _ type      member2;
         ---------        -----
         ---------        -----
   };
```

In defining a structure, we may note the following syntax:

1. The template is terminated with a semicolon.
2. While the entire definition is considered as a statement, each member is declared independently for its name and type in a separate statement inside the template.
3. The tag name such as book _ bank can be used to declare structure variables of its type, later in the program.

11.3 ARRAY VS STRUCTURE

1. An array is a collection of related data elements of same type. Structure can have elements of different types.
2. An array is derived data type whereas structure is a programmer-defined one.
3. Any array behaves like a built-in data type. All we have to do is to declare an array variable and use it. But in the case of a structure, first we have to design and declare a data structure before the variables of that type are declared and used.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

11.4 GIVING VALUES TO MEMBERS

We can access and assign values to the members of a structure in a number of ways. The members themselves are not variables. They should be linked to the structure variables in order to make them meaningful members. For example, the word title has no meaning, whereas the phrase 'title of book' has a meaning. The link between a member and a variable is established using the member operator '.',which is also known as 'dot operator' or 'period operator'. For example,

book1.price

is the variable representing the price of the book1 and can be treated like any other ordinary variable. Here is how we would assign values to the member of book1:

```
strcpy(book1.title, "COBOL");
strcpy(book1.author, "M.K.ROY");
book1.pages = 350;
book1. price =140;
```

We can also use scanf to give the values through the keyboard.

```
scanf("%s\n", book1.title);
scanf("%d\n", &book1.pages);
```

are valid input statements.

Example :

Define a structure type, struct personal, that would contain person name, date of joining and salary. Using this structure, write a program to read this information for one person from the keyboard and print the same on the screen.

Structure definition along with the program is shown below. The scanf and printf functions illustrate how the member operator '.' is used to link the structure members to the structure variables. The variable name with a period and the member name is used like an ordinary variable.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Program

```
/*************************************************************************/
/* DEFINING AND ASSIGNING VALUES TO STRUCTURE MEMBERS */
/*************************************************************************/
struct personal
  {
     char name[20];
     int day;
     char month[10];
     int year;
     float salary;
  };
 main()
  {
     struct personal person;
     printf("Input values\n");
     scanf("%s %d %s %d %f",
                  person .name,
                &person. day,
                  person.month,
                &person.year,
                &person.salary);
     printf("%s %d %s %d %.2f\n",
                   person .name,
                   person. day,
                  person.month,
                  person.year,
                  person.salary);
}
```

11.5 STRUCTURE INITIALIZATION

Like any other data type, a structure variable can be initialized at compile time.

```
main()
{
struct
{
int weight;
float height;
}
student ={60, 180.75};
……
……
}
```

This assigns the value 60 to student. weight and 180.75 to student. height. There is a one-to-one correspondence between the members and their initializing values.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

A lot of variation is possible in initializing a structure. The following statements initialize two structure variables. Here, it is essential to use a tag name.

```
main()
{
struct st _ record
{
int weight;
float height;
};
struct st_record student1 ={60, 180.75};
struct st_record student2 ={53, 170.60};
……
……
}
```

C language does not permit the initialization of individual structure member within the template. The initialization must be done only in the declaration of the actual variables.

11.6 COMPARISON OF STRUCTURE VARIABLES

Two variables of the same structure type can be compared the same way as ordinary variables. If person1 and person2 belong to the same structure, then the following operations are valid:

Operation	Meaning
person1 = person2	Assign person2 to person1.
person1 = =person2	Compare all members of person1 and person2 and return 1 if they are equal, 0 otherwise.
person1 != person2	Return 1 if all the members are not equal, 0 otherwise.

Note that not all compilers support these operations. For example, Microsoft C version does not permit any logical operations on structure variables. In such cases, individual member can be compared using logical operators.

11.7 ARRAYS OF STRUCTURES

We use structure to describe the format of a number of related variables. For example, in analyzing the marks obtained by a class of students, we may use a template to describe student name and marks obtained in various subjects and then declare all the students as structure variables. In such cases, we may declare an array of structure, each elements of the array representing a structure variable. For example,

```
struct class student[100];
```

It defines an array called student, that consists of 100 elements. Each elements is defined to be of the type struct class. Consider the following declaration:

```
struct marks
{
   int subject1;
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
    int subject2;
    int subject3;
  };
 main()
  {
    static struct marks  student[3] =
              { {45, 68, 81}, {75, 53, 69}, {57,36,71}};
```

This declares the student as an array of three elements students[0], student[1], and student[2] and initializes their members as follows:

```
student[0].subject1=45;
student[0].subject2=68;
      ……
      ……
student[2].subject3=71;
```

An array of structures is stored inside the memory in the same way as a multi- dimensional array.

Check Your Progress

Ex 1) Can we compare structure variables ? If so, how?

Ex 2) Construct a structure for bank details.

Student[0].subject1	45
.subject2	68
.subject3	81
Student[1].subject1	75
.subject2	53
.subject3	69
Student[2].subject1	57
.subject2	36
.subject3	71

The array student inside memory.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

11.8 LET US SUM UP

In this lesson we have learnt about

- the method defining structure
- how to differentiate between array and structure
- how to give values to members
- initializing structure
- comparing structure variables
- using arrays of structures

11.9 LESSON-END ACTIVITIES

Try to find the answers to the following exercises on your own

1) Bring out the general format of structure definition.
2) How will you initialize structures?
3) How will you compare structure variables? Explain.
4) Explain arrays of structures.

11.10 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) Yes, we can compare structure variables. We can use =, = = , ! = for our purpose.

[Answers vary]

Ex 2) A structure for bank details is given below :

```
struct  bank
  {
    int  accno;
    char name[25];
    float amount;
   }
```

11.11 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON - 12

MORE ABOUT STRUCTURES AND UNION

12.0 AIMS AND OBJECTIVES

In this lesson, we are going to study about structures within structures, structures and functions, unions, size of structures and Bit-fields.

After learning this lesson, we should be able to

- understand structures within structures
- make use of structures and functions
- understand unions
- identify the size of structures
- gain knowledge on bit fields

12.1 INTRODUCTION

C permits the use of arrays as structure members. We have already used arrays of characters inside a structure. Similarly, we can use single or multi- dimensional arrays of type int or float. For example, the following structure declaration is valid:

```
struct marks
 {
   int number;
   float  subject[3];
   student[2];
 }
```

Here, the member subject contains three elements, subject [0], subject [1] and subject [2]. These elements can be accessed using appropriate subscripts. For example, the name

```
student[1].subject[2];
```

would refer to the marks obtained in the third subject by the second student.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

12.2 STRUCTURES WITHIN STRUCTURES

Structures within structures means nesting of structures. Nesting of structures is permitted in C. Let us consider the following structure defined to store information about the salary of employees.

```
struct salary
  {
      char name[20];
      char department[10];
      int       basic _ pay;
      int       dearness_ allowance;
      int       house _ rent _ allowance;
      int       city_ allowance;
   }
       employee;
```

This structure defines name, department , basic pay and three kinds of allowances. We can group all items related to allowance together and declare them under a substructure as shown below:

```
struct salary
{
char name[20];
char department[10];
struct
{
int  dearness;
int house_rent;
int city;
}
allowance;
}
employee;
```

An inner structure can have more than one variable. The following form of declaration is legal:

```
struct salary
{
……
struct
{
int dearness;
…….
}
allowance,
arrears;
}
employee[100];
```

It is also possible to nest more than one type of structures.

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
struct personal_record
{
struct name_part name;
struct addr_part address;
struct date date _ of _ birth
……..
……..
};
struct personal_record person 1;
```

The first member of this structure is name which is of the type struct name_part. Similarly, other members have their structure types.

12.3 STRUCTURES AND FUNCTIONS

C supports the passing of structure values as arguments to functions. There are three methods by which the values of a structure can be transferred from one function to another.

The first method is to pass each member of the structure as an actual argument of the function call.

The second method involves passing of a copy of the entire structure to the called function.

The third approach employs a concept called pointers to pass the structure as an argument.

The general format of sending a copy of a structure to the called function is:

function name(structure variable name)

The called function takes the following form:

```
data_type function name(st_name)
struct_ type st_name;
 {
      …….
      ……..
       return (expression);
 }
```

The following points are important to note:

1. The called function must be declared for its type, appropriate to the data type it is expected to return. For example, if it is returning a copy of the entire structure, then it must be declared as struct with an appropriate tag name.
2. The structure variable used as the actual argument and the corresponding formal argument in the called function must be of the same struct type.
3. The return statement is necessary only when the function is returning some data. The expression may be any simple variable or structure variable or an expression using simple variables.
4. When a function returns a structure, it must be assigned to a structure of identical type in the calling function.
5. The called function must be declared in the calling function for its type, if it is placed after the calling function.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Check Your Progress

Ex 1) Can we nest the structures?

---

2) Can we use arrays within structure?

12.4 UNIONS

Like structures, a union can be declared using the keyword union as follows:

```
union item
{
   int m;
   float x;
   char c;
} code;
```

This declares a variable code of type union item.

The compiler allocates a piece of storage that is large enough to hold the largest variable type in the union.

To access a union member, we can use the same syntax that we use for structure members. That is,

```
code.m
code.x
code.c
```

are all valid member variables. During accessing, we should make sure that we are accessing the member whose value is currently stored. For example, the statement such as

```
code.m = 379;
code.x=7859.36;
printf("%d", code.m);
```

would produce erroneous output.

In effect, a union creates a storage location that can be used by any one of its members at a time. When a different member is assigned a new value, the new value supercedes the previous member's value.

12.5 SIZE OF STRUCTURES

We normally use structures, unions and arrays to create variables of large sizes. The actual size of these variables in terms of bytes may change from machine to machine. We may use the unary operator sizeof to tell us the size of a structure. The expression

```
sizeof(struct x)
```

will evaluate the number of bytes required to hold all the members of the structure x. If y is a simple structure variable of type struct x, then the expression

```
sizeof(y)
```

would also give the same answer. However, if y is an array variable of type struct x, then

```
sizeof(y)
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

would give the total number of bytes the array requires.

This kind of information would be useful to determine the number of records in a database. For example, the expression

sizeof(y) / sizeof(x)

would give the number of elements in the array y.

12.6 BIT FIELDS

C permits us to use small bit fields to hold data items and thereby to pack several data items in a word of memory. Bit fields allow direct manipulation of string of a string of preselected bits, as if it is represented an integral quantity.

A bit field is a set of adjacent bits whose size can vary from 1 to 16 bits in length. A word can be divided into a number of bit fields. The name and size of bit fields are defined using a structure.

The general form of bit filed definition is

```
struct  tag-name
   {

      data-type  name1 : bit-length;
      data-type  name2 : bit-length;
      data-type  name3 : bit-length;
      ---------
      ---------
      ---------
      data-type  nameN : bit-length;
   }
```

The data type is either int or unsigned int or signed int and the bit-length is the number of bits used for the specific name. The bit-length is decided by the range of value to be stored.

The largest value that can be stored is 2^{n-1}, where n is bit-length. The internal representation of bit-field is machine dependent. It depends on the size of int and the ordering of bits.

Example :

Suppose we want to store and use the personal information of employees in compressed form. This can be done as follows:

```
struct personal
 {
      unsigned sex:          1
      unsigned age :         7
      unsigned  m_status:    1
      unsigned children:     3
      unsigned          :    4
}  emp;
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

This defines a variable name emp with 4 bit fields. The range of values each filed could have is as follows:

Bit Filed	Bit length	Range of values
sex	1	0 or 1
age	7	0 to 127 ($2^7 - 1$)
m_status	1	0 or 1
children	3	0 to 7 ($2^3 - 1$)

The following statements are valid :

```
emp.sex =1 ;
emp.age = 50;
```

It is important to note that we can not use scanf to read the values in to the bit field.

12.7 LET US SUM UP

In this lesson, we have learnt about

- structures within structures
- structures and functions
- the concept of unions
- identification of the size of structures
- bit fields

12.8 LESSON-END ACTIVITIES

Try to find the answers to the following exercises on your own

1) Explain arrays within structures with a simple example.
2) Explain structures within structure with a simple example.
3) Differentiate Structure and Union.
4) Write a brief note on Bit-fields.

12.9 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) Yes, we can nest the structures

2) Yes, we can use arrays within structure

12.10 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON – 13

FILES

13.0 AIMS AND OBJECTIVES

In this lesson, we are going to describe the concept of files, how to define , open and close a file. Also, we are to discuss about the Input/output operations that can be performed on files.

After learning this lesson, we should be able to

- define and open a file
- close file
- perform input/output operations on files

13.1 INTRODUCTION

Many real-life problems involve large volumes of data and in such situations, the console oriented I/O operations pose two major problems.

1. It becomes cumbersome and time consuming to handle large volumes of data through terminals.
2. The entire data is lost when either the program is terminated or the computer is turned-off.

It is therefore necessary to have a more flexible approach where data can be stored on the disk and read whenever necessary, without destroying the data. This method employs the concept of files to store data.

There are two distinct ways to perform file operations in C. The first one is known as the low-level I/O and uses UNIX system calls. The second method is referred to as the high-level I/O operation and uses functions in C's standard I/O library.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

High-level I/O functions

Function name	Operation
fopen( )	# Creates a new file for use . # Opens an existing file for use.
fclose( )	# Closes a file which has been opened for use.
getc( )	# Reads a character from a file.
putc( )	# Writes a character to a file .
fprintf( )	# Writes a set of data values to a file.
fscanf( )	# Reads a set of data values from a file.
getw( )	# Reads an integer from a file.
putw( )	# Writes an integer to a file.
fseek( )	# Sets the position to the desired point in the file.
ftell( )	# Gives the current position in the file(in terms of bytes from the start).
rewind( )	# Sets the position to the beginning of the file.

13.2 DEFINING AND OPENING A FILE

Data structure of a file is defined as FILE in the library of standard I/O function definitions. Therefore all files should be declared as type FILE before they are used. FILE is a defined data type.

The following is the general format for declaring and opening a file:

```
FILE    *fp;
fp = fopen(“filename”, “mode”);
```

Mode can be one of the following:

r open the file for reading only.
w open the file for writing only.
a open the file for appending(or adding)data to it.

The additional modes of operation are:

r+ the existing file is opened to the beginning for both reading and writing.
w+ same as w except both for reading and writing.
a+ same as a except both for reading and writing.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

13.3 CLOSING A FILE

A file must be closed as soon as all operations on it have been completed. This ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken. It also prevents any accidental misuse of the file. The syntax for closing a file is

```
fclose( file_pointer);
```

This would close the file associated with the FILE pointer file_pointer .
Example:

```
…..
…..
FILE *p1, *p2;
p1 = fopen(“INPUT”, “w”);
p2 = fopen(“OUTPUT” , “r”);
…..
…..
fclose(p1);
fclose(p2);
…..
```

This program opens two files and closes them after all operations on them are completed. Once a file is closed, its file pointer can be reused for another file. All files are closed automatically whenever a program terminates.

Check Your Progress

Ex 1) Specify the mode used for reading.

---

2) Specify the mode used for writing.

13.4 INPUT/OUTPUT OPERATIONS ON FILES

The getc and putc functions

The file i/o functions getc and putc are similar to getchar and putchar functions and handle one character at a time. The statement

```
putc(c, fp1);
```

writes the character contained in the character variable c to the file associated with FILE pointer fp1. Similarly, getc is used to read a character from a file that has been opened in the read mode. For example, the statement

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
c = getc(fp2);
```

would read a character from the file whose file pointer is fp2.

A program to illustrate to read data from the keyboard, write it to a file called INPUT, again read the same data from the INPUT file, and then display it on the screen is given below:

Program

```
/*************************************************************************/
/*                  WRITING TO AND READING FROM A FILE                   */
/*************************************************************************/
#include <stdio.h>
main ( )
{
        FILE *fp1;
        char c;
        printf("Data input\n\n");
        f1 = fopen("INPUT" , "w" )          /* Open the file INPUT              */
        while((c = getchar( ))!=EOF)        /* Get a character from keyboard    */
                putc(c,f1)                  /* Write a character to INPUT       */
        fclose(f1);                         /* Close the file INPUT             */
        printf("\n Data OUTPUT\n\n");
        f1 = fopen("INPUT" , "r")           /* Reopen the file INPUT            */
        while ((c =getc( f1))!=EOF)         /* Read a character from INPUT      */
                printf("%c", c);            /* Display a character on the screen */
        fclose(f1);                         /* close the file INPUT             */
}
```

Output

```
        Data input                              Data output
This a program to test the file handling
features in the system  ^z                  This is a program to test the file handling
                                              features in the system
```

Character oriented read/write operations on a file.

The getw and putw functions

The getw and putw are integer oriented functions. They are similar to getc and putc functions and are used to read and write integer values only.

The general forms of getw and putw are:

```
putw(integer,fp);
```

```
getw(fp);
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

A program to read a series of integer numbers from the file DATA and then write all odd numbers into the file ODD and even numbers into file EVEN is given below:

Program HANDLING OF INTEGER DATA FILES

```
#include <stdio.h>
main ( )
{
        FILE *f1, *f2, *f3;
        int number, i;
        printf("Contents of DATA file\n\n");
        f1 = fopen("DATA", "w");            /*Create DATA file                    */
        for(i =1; i< =30; i++)
        {
                scanf("%d", &number);
                if(number == -1)break;
                putw(number,f1);
        }
        fclose(f`1);

        f1 = fopen("DATA" , "r");
        f2 = fopen("ODD" , "w");
        f3 = fopen("EVEN", "w");

        while((number = getw(f1))!= EOF)   /*Read from DATA file        */
        {
                if(number %2  = = 0)
                  putw(number,f3);          /*Write to EVEN file          */
                else
                  putw(number,f2);          /*Write to ODD file           */
        }
        fclose(f1);
        fclose(f2);
        fclose(f3);
        f2 = fopen("ODD" , "r");
        f3 = fopen("EVEN" , "r");
        printf("\n\nContents of ODD file \n\n");
        while((number = getw(f2)) != EOF)
        printf("%4d", number);
          printf("\n\n Contents of EVEN file\n\n");
        while((number = getw(f3)) ! = EOF)
          printf("%4d" , number);
        fclose(f2);
        fclose(f3);
}
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The fprintf and fscanf functions

The functions fprintf and fscanf perform I/O operations that are identical to the printf and scanf functions except of course that they work on files. The general form of fprintf is

```
fprintf(fp, “control string”, list);
```

The general format of fscanf is

```
fscanf(fp, “control string”, list);
```

Write a program to open a file named INVENTORY and store in it the following data:

Item name	**Number**	**Price**	**Quantity**
AAA-1	111	17.50	115
BBB-2	125	36.00	75
C-3	247	31.75	104

Extend the program to read this data from the file INVENTORY and display the inventory table with the value of each item

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
/*Program HANDLING OF FILES WITH MIXED DATA TYPES  */
/*                    (scanf and fprintf )                                  */

#include <stdio.h>
main( )
{
        FILE    *fp;
        int number,quantity,i;
        float price,value;
        char item[10],filename[10];
        printf("Input file name\n");
        scanf("%s",filename);
        fp = fopen(filename, "w");
        printf("Input inventory data\n\n");
        printf("Item name      Number         Price    Quantity\n");
        for(i=1;i <=3;i++)
        {
            fscanf(stdin,  "%s %d %f %d",
                        item, &number, &price, &quantity);
            fprintf(fp,  "%s %d %2f %d",
                        item, number, price,  quantity);
        }
        fclose(fp);
        fprintf(stdout, "\n\n");
        fp = fopen(filename, "r");
        printf("Item name      Number         Price    Quantity\n");
        for(i=1; i <=3; i++)
        {
          fscanf(fp, "%s %d %f %d",
                        item, &number,  &price, &quantity);
          value = price * quantity;
          fprintf(stdout, "%-8s  %7d  %8.2f  %8d  %11.2f\n",
                        item, number, price, quantity, value); }
        fclose(fp);
}
```

13.5 LET US SUM UP

In this lesson, we have learnt about

- defining and opening files
- closing files
- performing input/output operations on files

Having learnt these important concepts on files, we can use them without any ambiguity.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

13.6 LESSON-END ACTIVITIES

Try to find the answers to the following exercises on your own

1) Specify the general format for declaring and opening a file.
2) List out the different file modes and explain them.
3) Explain the role of getc and putc functions.
4) Explain the role of getw and putw functions.
5) How will you close a file?

13.7 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) The mode used for reading is ‘r’

2) The mode used for writing is ‘w’

13.8 REFERENCES

Ashok N Kamthane: “PROGRAMMING AND DATA STRUCTURES” – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON - 14

ERROR HANDLING DURING FILE I/O OPERATIONS

14.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about error handling and random access to files.

After learning this lesson, we should be able to

- use feof()
- use ferror()
- understand Random access to files
- understand ftell() and fseek() functions.

14.1 INTRODUCTION TO ERROR HANDLING

The feof function can be used to test for an end of file condition. It takes a FILE pointer as its only argument and returns non zero integer value if all of the data from the specified file has been read, and returns zero otherwise. If fp is a pointer to file that has just been opened for reading , then the statement

```
if(feof(fp))

printf("End of data.\n");
```

would display the message "End of data." on reaching the end of file condition. The ferror function reports the status of the file indicated. It also takes a FILE pointer as its argument and returns a nonzero integer if an error has been detected up to that point, during processing. It returns zero otherwise. The statement

```
if(ferror(fp) !=0)
printf("An error has occurred.\n");
```

would print the error message, if the reading is not successful. If the file cannot be opened for some reason then the function returns a null pointer. This facility can be used to test whether a file has been opened or not.

Example:

```
if(fp = = NULL)
printf("File could not be opened.\n");
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Check Your Progress

Ex 1) The purpose of feof() is to _____

2) The purpose of ferror () is to _____

14.2 RANDOM ACCESS TO FILES

There are occasions, where we need accessing only a particular part of a file and not in reading the other parts. This can be achieved by using the functions fseek, ftell, and rewind available in the I/O library.

ftell takes a file pointer and returns a number of type long, that corresponds to the current position and useful in saving the current position of a file, which can be used later in the program. It takes the following form:

```
n = ftell(fp);
```

rewind takes a file pointer takes a file pointer and resets the position to the start of the file.

Example :

```
rewind(fp);
n = ftell(fp);
```

would assign 0 to n because the file position has been set to the start of the file by rewind.

fseek function is used to move the file position to a desired location within the file.

The syntax is

```
fseek(file ptr, offset, position);
```

The position can take any one of the following three values:

Value	Meaning
0	Beginning of file.
1	Current position.
2	End of file.

The offset may be positive, meaning move forwards, or negative , move backwards.

The following examples illustrates the operation of the fseek function:

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Statement	Meaning
fseek(fp,0L,0);	Go to the beginning .
fseek(fp,0L,1);	Stay at the current position.
fseek(fp, 0L,2);	Go to the end of the file, past the last character of the file.
fseek(fp,m,0);	Move to (m+1)th bytes in the file.
fseek(m,1);	Go forward by m bytes.
fseek(fp,-m,1);	Go backward by m bytes from the current position.
fseek(fp,-m,2);	Go backward by m bytes from the end.

When the operation is successful, fseek returns 0 or if the operation fails it returns –1.

14.3 LET US SUM UP

In this lesson , we have learnt about

- using feof()
- using ferror()
- Random access to files
- ftell() and fseek() functions.

14.4 LESSON-END ACTIVITIES

Try to find the answers to the following exercises on your own

1) Explain the usage of feof() .
2) Explain the usage of ferror() .
3) What do you infer by Random access to files?
4) Explain the operation of fseek() function with examples.

14.5 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) The purpose of feof() is to test the End-of-file condition

2) The purpose of ferror () is to report the status of file indicated

14.6 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON - 15

COMMAND LINE ARGUMENTS

15.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about the command line arguments , namely, argc and argv . Also, we are going to discuss a program that makes use of command line arguments.

After reading this lesson, we should be able to

- understand command line arguments
- understand the usage of argc and argv parameters.

15.1 COMMAND LINE ARGUMENTS

It is a parameter supplied to a program when a program is invoked. The main can take two arguments called argc and argv. The variable argc is an argument counter that counts the number of arguments on the command line. The argv is an argument vector and represents an array of character pointers that point to the command line arguments. The size of this array will be equal to the value of the argc. In order to access the command line arguments, we must declare the main function and its parameters as follows.

```
main(argc,argv)
int argc;
char *argv[];
{
……
……
}
```

A program that will receive a file name and a line of text as command line arguments and write the text to the file is presented below:

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Check Your Progress

Ex 1) What is the role of argc?

---

2) What is the role of argv?

15.2 PROGRAM FOR COMMAND LINE ARGUMENTS

COMMAND LINE ARGUMENTS

```
#include<stdio.h>
main(argc,argv)                         /* main with arguments                    */
int argc;                               /*  argument count                        */
char argv[];                            / * list of arguments                    */
{
        FILE *fp;
        int i;
        char word[15];
        fp = fopen(argv[1], “w”);       /* Open file with name argv[1]            */
        printf(“\nNo of arguments in command line = %d\n\n”, argc);
        for(i=2;i<argc;i++)
              fprintf(fp,”%s”,argv[i]); /*  Write to file arv[1]          */
        fclose(fp);

/* Writing content of the file to screen                                          */
        printf(“Contents of %s file\n\n”,argv[1]);
        fp = fopen(argv[1], “r”);
        for(i=2:i<argc;i++)
        {
           fscanf(fp,”%s”,word);
           printf(“%s”,word);
        }
        fclose(fp);
        printf(“\n\n”);
)
```

15.3 LET US SUM UP

In this lesson, we have learnt about

- command line arguments
- the usage of argc and argv parameters.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

15.4 LESSON-END ACTIVITIES

Try to find the answers to the following exercises on your own

1) What do you mean by command line arguments?
2) How many arguments main() can take?

15.5 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) The role of argc is to count the number of arguments on the command line.

Ex 2) The role of argv is to represent an array of character pointers that point to the command line arguments.

15.6 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

UNIT IV

LESSON – 16

LINEAR DATA STRUCTURES

16.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about the definition of data structures and its representation. .We can also know how to search and retrieve an element.

After reading this lesson, we should be able to

- know the different types of data structure.
- the predecessor and successor of a list
- search easily and also retrieve an element from a database.

16.1 INTRODUCTION

Data structure is one of the method of representation of logical relationships between individual data elements related to the solution of a given problem. Data structure is the most convenient way to handle data of different data types including abstract data type for a known problem. For example, the characteristics of a house can be represented by house name, house number, location, number of floors, number of rooms on each floor etc.

Data structure is the base of the programming tools and the choice of data structure provides the following things:

1. The data structure should be satisfactory to represent the relationship between data elements.
2. The data structure should be easy so that the programmer can easily process the data, as per requirement.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Data structures have been classified in several ways as outlined below:

Linear: In the linear method, values are arranged in a linear fashion. An array, linked list, stacks and queues are examples of linear data structure in which values are stored in a sequence. There is a relation between the adjacent elements of a linear list.

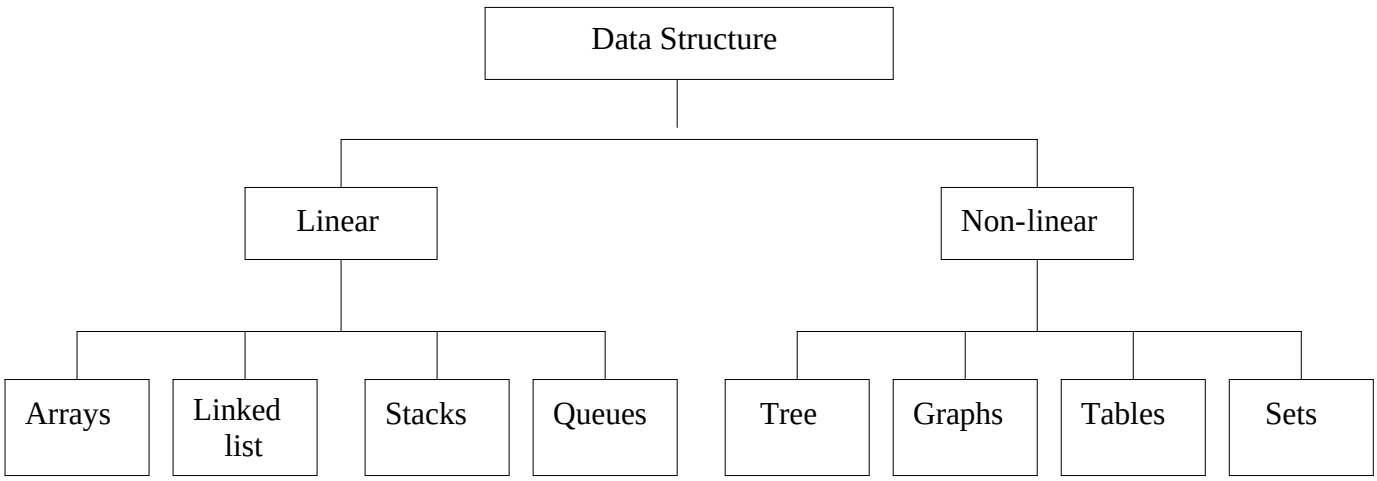
Non-linear: This is just opposite to linear. The data values in this structure are not arranged in order. Tree, graph, table and sets are examples of non-linear data structure.

Homogenous: In this type of data structure, values of same data types are stored. An example of it can be an array that stores similar data type elements having different locations for each element of them.

Non-homogenous: In this type, data values of different types are grouped like in structure and classes.

Dynamic: In dynamic data structure like references and pointers, size and memory locations can be changed during program execution.

Static: A static keyword in C/C++ is used to initialize the variable to 0(NULL). Static variable value remains in the memory throughout the program. Value of the static variable persists. In C++, a member function can be declared as static and such a function can be invoked directly.



Types of data structure

This watermark does not appear in the registered version - http://www.clicktoconvert.com

16.2 IMPLEMENTATION OF A LIST

There are two methods to implement the list: They are classified as Static and Dynamic.

Static implementation: Static implementation can be achieved using arrays. Though it is a very simple method, it has a few limitations. Once the size of an array is declared, its size cannot be altered during program execution. In array declaration, memory is allocated equal to array size. Thus, the program itself decides the amount of memory needed and it informs accordingly to the compiler. Hence memory requirement is determined in advance before compilation and the compiler provides the required memory.

Static allocation of memory has a few disadvantages. It is an inefficient memory allocation technique. It is suitable only when we exactly know the number of elements to be stored.

Dynamic implementation: Pointers can also be used for implementation of a stack. The linked list is an example of this implementation. The limitations noticed in static implementation can be removed using dynamic implementation. The dynamic implementation is achieved using pointers. Using pointer implementation at run time there is no restriction on number of elements. The stack may be expanded. It is efficient in memory allocation because the program informs the compiler its memory requirement at run time. Memory is allocated only after an element is pushed.

16.3 TRAVERSAL OF A LIST

A simple list can be created using an array in which elements are stored in successive memory locations. The following program is an example.

Given below is a program to create a simple list of elements. The list is displayed in original and reverse order.

```
#include <stdio.h>
#include <conio.h>
main()
{
int sim[5],j;
clrscr();
printf("\n Enter five elements :");
for(j=0; j<5; j++)
scanf("%d", &sim[j]);
printf("\n List :");
for(j=0; j<5; j++)
printf("%d", sim[j]);
printf("\n Reverse List :");
for(j=4; j>=0; j--)
printf("%d",sim[j]);
}
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

OUTPUT

```
Enter five elements: 1 5 9 7 3
List    : 1 5 9 7 3
Reverse list: 3 7 9 5 1
```

Explanation: In the above program, using the declaration of an array a list is implemented. Using for loop and scanf() statements five integers are entered. The list can be displayed using the printf() statement. Once a list is created, various operations such as sorting and searching can be applied.

16.4 SEARCHING AND RETRIEVING AN ELEMENT

Once a list is created, we can access and perform operations with the elements. One can also specify some conditions such as to search numbers which are greater than 5, or equal to 10 or any valid condition. If the list contains more elements, then it may be difficult to find a particular element and its position.

Consider the following program which creates a list of integer elements and also search for the entered number in the list.

```
#include <stdio.h>
#include <conio.h>
main()
{
int sim[7], j, n, f=0;
clrscr();
printf("\n Enter seven Integers :");
for (j=0; j<7; j++)
scanf("%d", &sim[j]);
printf("\n Enter Integer to search :");
scanf("%d", &n);
for(j=0; j<7; j++)
{
if(sim[j]= =n)
{
f=1;
printf("\n Found ! position of integer %d is %d", n, j+1);
break;
}
}
if(f= =0)
printf("\n Element not found !");
}
```

OUTPUT

```
Enter seven integers: 24 23 45 67 56 89
Enter integer to search: 67
Found ! position of integer 67 is 5.
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Note: In the above program an array sim[7] is declared and it stores seven integers, which are entered by the programmer. Followed by this, an element is entered to search in the list. This is done using the for loop and the embedded if statement. The if() statement checks the entered element with the list elements. If the match is found the flag is set to '1', else it remains '0'. When a match is found, the number's position is displayed.

Check Your Progress

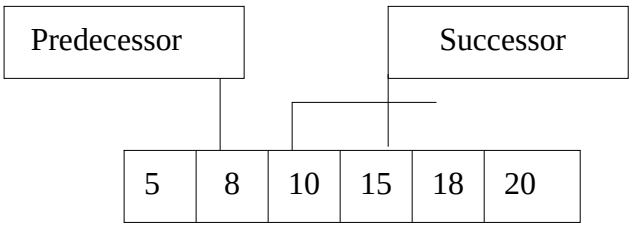
Ex 1) Specify examples of non-linear data structure.

Ex 2) Give an example for dynamic data structure.

16.4.1 PREDECESSOR AND SUCCESSOR

The elements of a list are located with their positions. For example, we can place the elements at different locations and call them as element n, (n-1) as predecessor and (n+1) as successor. Once the position of an element is fixed, we can easily find its predecessor and successor. In other words, the elements have relative positions in the list. The left element is the predecessor and the right element is the successor.

The first element of a list does not have a predecessor and the last one does not have a successor.



Predecessor and successor

The above diagram shows the predecessor and successor elements of number 10.

The following program displays the predecessor and successor elements of the entered element from the list.

Program to find the predecessor and successor of the entered number in a list.

```
#include<stdio.h>
#include<conio.h.
main()
{
int num[8], j, n, k=0, f=0;
clrscr();
printf("\n Enter eight elements  : ");
for(j=0; j<8; j++)
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
scanf("%d", &num[j]);
printf("\n Enter an element :");
scanf("%d", &n);
for(j=0; j<8; j++)
{
if (n= = num[j])
{
f=1;
(j>0) ? printf("\n The predecessor of  %d is  %d", num[j],num[j-1]);
printf(" No predecessor");
(j= =7) ? printf("\n No successor");
printf("\n The successor of  %d is %d", num[j], num[j+1]);
break;
}
k++;
}
if(f = = 0)
printf("\n The element % d is not found in list", n);}
```

OUTPUT

Enter eight elements : 1 2 5 8 7 4 46
Enter an element : 5

The predecessor of 5 is 2
The successor of 5 is 8

Enter eight elements : 1 2 3 4 5 6 7 8

Enter an element:1
No predecessor
The successor of 1 is 2

Enter eight elements: 12 34 54 76 69 78 85 97

Enter an element : 3

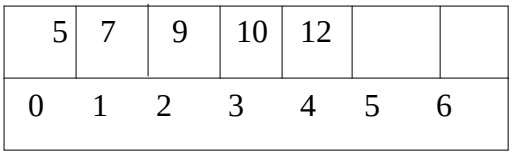
The element 3 is not found in the list.

INSERTION

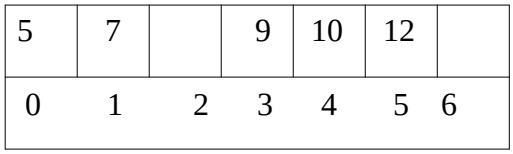
When a new element is added in the list it called as **appending.** However, when an element is added in between two elements in the list, then the process is called as **insertion.** The insertion can be done at the beginning, inside or anywhere in the list.

For successful insertion of an element, the array implementing the list should have at least one empty location. If the array is full, insertion cannot be possible. The target location where element is to be inserted is made empty by shifting elements downwards by one position and the newly inserted element is placed at that location.

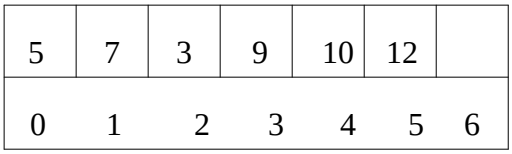
This watermark does not appear in the registered version - http://www.clicktoconvert.com



(a)



(b)



(c)

Insertion

There are two empty spaces available. Suppose we want to insert 3 in between 7 and 9. All the elements after 7 must be shifted towards end of the array. The entered number 3 can be assigned to that memory location .The above example describes the insertion of an element.

The following program illustrates the insertion operation

```
#include<stdio.h>
#include<conio.h.>
#include<process.h>
main()
{
int num[8]={0}, j, k=0, p, n;
clrscr();
printf("\n Enter elements (0 to exit) : " );

for(j=0; j<8; j++)
{
scanf("%d" , &num[j]);
if( num[j]= =0)
break;
}

if(j<8)
{
printf("\n Enter position number and element :");
sacnf("%d %d", &p,&n);
--p;
}
else
while(num[k]!=0)
k++;
k--;
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com


```
for(j=k;j>=p;j--)
num[j+1]=num[j];
num[p]=n;

for(j=0;j<8;j++)
printf("%d', num[j]);
}
```

OUTPUT

Enter elements(0 to Exit): 1 2 3 4 5 6 0
Enter position number and element: 5 10
1 2 3 4 10 5 6 0

DELETION

Like insertion, deletion of an element can be done from the list. In deletion, the elements are moved upwards by one position.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
main()
int num[8]={0}, j, k=0, p, n;
clrscr();
printf("\n Enter elements(0 to Exit) :");

for(j=0;j<8;j++)
{
scanf(%d", &num[j]);
if(num[j]= =0)
break;
}
printf("\n Enter an element to remove:");
scanf("%d", &n);

while(num[k]!=n)
k++;

for(j=klj<7;j++)
num[j]=num[j+1];
num[j]=0;

for(j=0;j<8;j++)
printf("%d", num[j]);
```

OUTPUT

Enter elements(0 to exit): 5 8 9 4 2 3 4 7
Enter element to remove: 9
5 8 4 2 3 4 7 0

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Explanation:

In the above program, elements are entered by the user. The user can enter maximum eight elements. The element which is to be removed is also entered. The while loop calculates the position of the element in the list. The second for loop shifts all the elements next to the specified element one position up towards the beginning of the array. The element which is to be deleted is replaced with successive elements. The last element is replaced with zero. Thus, the empty spaces are generated at the end of the array.

SORTING

Sorting is a process in which records are arranged in ascending or descending order. The records of the list of these telephone holders are to be sorted by the name of the holder. By using this directory, we can find the telephone number of any subscriber easily. Sort method has great importance in data structures. Different sort methods are used in sorting elements/records.

Here is the program to sort entered numbers by exchange method

```
#include<stdio.h>
#include<conio.h>
main( )
{
int num{8}={0};
int k=0,h,a,n,tmp;
clrscr( );
printf("\nEnter numbers :");
for(a=0;a<8;++a)
scanf("%d",&num[a]);

while(k<7)
{
for (h=k+1;h<8;++h)

if (num[k]>num[h])
{ tmp=num[k];
num[k]=num[h];
num[h]=tmp;
}
printf ("\n Sorted array:");
for (k=0;k<8;++k)
printf("%d",num[k]);
}
```

OUTPUT:

Enter numbers:4 5 8 7 9 3 2 1
Sorted array: 1 2 3 4 5 7 8 9

This watermark does not appear in the registered version - http://www.clicktoconvert.com

16.5 MERGING LISTS

Merging is a process in which two lists are merged to form a new list. The new list is the sorted list. Before merging individual lists are sorted and then merging is done.

Write a program to create two array lists with integers. Sort and store elements of both of them in the third list

```
# include<stdio.h>
# include<conio.h>
# include<math.h>

main( )
{
int m,n,p,sum=0;
int listA[5],listB[5],listC[10]={0};
clrscr( );
printf("\nEnter the elements for first list:");
for (m=0;m<5;m++)
{
scanf("%d",&listA[m]);
if (listA[m]==0) m--;
sum=sum+abs(listA[m]);
}
printf("\n Enter element for the second list:");
for (n=0;n<5;n++)
{
scanf("%d",&listB[n]);
if(listA[n]==0) n--;
sum=sum+abs(listB[n]);
}
p=n=m=0;
m=m-sum;
while (m<sum)
{
for (n=0;n<5;n++)
{
if (m==listA[n] ||m==listB[n])
listC[p++]=m;
if (m==listA[n] && m= =listB[n])
listc[p++]=m;
}
m++;
}
puts("Merged sorted list:");
for (n=0;n<10;++n)
printf ("%d", listC[n]);
}
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

OUTPUT:

Enter elements for first list:1 5 4 -3 2

Enter elements for second list:9 5 1 2 10
Merged sorted list:
-3 1 1 2 2 4 5 5 9 10

16.6 LET US SUM UP

In this lesson, we have

- described the definition of data structures and different types of data structures
- explained the list implementation and list traversal algorithm.
- suggested the method of merging the list.

16.7 LESSON END ACTIVITIES

1. Define data structure.
2. Mention the different types of data structure.
3. What is meant by Predecessor and Successor?
4. Explain how will you merge a list.

16.8 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) Examples of non-linear data structure are
Tree, Graph, Table, Sets

Ex 2) An example for dynamic data structure is Pointer

16.9 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source,1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON – 17

STACK

17.0 AIMS AND OBJECTIVES

In this lesson, we are going to discuss about stack and its representations. Also, we are to learn about the operations like insertion and deletion.

After reading this lesson, we should be able to

- know how to create the stack
- know the operations performed in stacks
- understand the method of deleting elements from the stack

17.1 INTRODUCTION

There are two common data objects found in computer algorithm called stacks and queues Data objects in an ordered list is A(a1,a2,……an) where n>=0 and ai are the atoms taken from the set. If the list is empty or Null then n=0.

17.2 STACK

Stack is an array of size N, where N is an unsigned integer. It is an ordered list in which all insertions and deletions are made at one end called **TOP.**

17.3 REPRESENTATION AND TERMS

Storage: A function contains local variables and constants .These are stored in a stack. Only global variables in a stack frame.

Stack frames: This data structure holds all formal arguments ,return address and local variables on the stack at the time when function is invoked.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

TOP: The top of the stack indicates its door. The stack top is used to verify the stack's current position, that is, underflow and overflow. Some programmers refer to -1 assigned to top as initial value. This is because when an element is added the top is incremented and it would become zero. It is easy to count elements because the array element counting also begins from zero. Hence it is convenient to assign -1 to top as initial value.

Stack underflow: When there is no element in the stack or the stack holds elements less than its capacity, then this stack is known as stack underflow. In this situation, the TOP is present at the bottom of the stack. When an element is pushed, it will be the first element of the stack and top will be moved to one step upper.

Stack overflow: When the stack contains equal number of elements as per its capacity and no more elements can be added such status of stack is known as stack overflow. In such a position, the top rests at the highest position.

Check Your Progress

Ex 1) Top of the stack indicates its
---

2) Static implementation of stack is achieved using ----------

17.4 STACK OPERATIONS

CREATE (S)	← Creates an empty stack
ADD (i,S)	←Add i to rear of stack
DELETE (S)	← Removes the top element from stack
FRONT (S)	←Returns the top element of stack.
ISEMT(S)	← returns true if stack is empty else false.

17.4.1 INSERTION OPERATION

CREATE ( ) : = declare Stack(1:n) ,top<- 0
ISEMTS (S) =If TOP=0 then true else false.
TOP(S) = if top= 0 then error else stack (top)
ADD & DELETE operations are implemented as,
Procedure ADD (item, stack,n,top)
If top >=n then call stack-Full
Top<- top + 1
Stack (top) <- item
End ADD**.**

17.4.2 DELETION OPERATION

Procedure DELETE (item, stack,top)
If top <=0 then call stack-empty
Item <- stack(top)
Top<- top-1
End delete.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

17.5 IMPLEMENTATION OF A STACK

The stack implementation can be done in the following ways:

Static implementation: Static implementation can be achieved using arrays. Though, it is a very simple method, but it has few limitations. Once the size of an array is declared, its size cannot be altered during program execution. While in array declaration, memory is allocated equal to array size. The vacant space of stack (array) also occupies memory space. Thus, it is inefficient in terms of memory utilization. In both the cases if we store less argument than declared, memory is wasted and if we want to store more elements than declared, array could not expand. It is suitable only when we exactly know the number of elements to be stored.

Elements are entered stored in the stack .The element number that is to be deleted from the stack will be entered from keyboard. Recall that in a stack, before deleting any element, it is compulsory to delete all elements before that element. In this program the elements before a target element are replaced with value NULL(0). The elements after the target element are retained. Recall that when an element is inserted in the stack the operation is called 'push', When an element is deleted from the stack then the operation is pop.

The push() operation inserts an element into the stack. The next element pushed() is displayed after the first element and so on. The pop() operation removes the element from the stack. The last element inserted is deleted first.

17.6 LET US SUM UP

In this lesson, we have

- described about stack and the operations performed in stack
- also discussed the purpose of push( ) and pop ( ) operations
- explained how to represent a stack
- mentioned about insertion and deletion operations

17.7 LESSON END ACTIVITIES

1. What is stack and how to represent it in a list?
2. Mention the two operations performed in stack.
3. What ways are used to implement the stack?

17.8 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) Top of the stack indicates its key element
2) Static implementation of stack is achieved using arrays

17.9 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source,1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON – 18

LINKED LIST

18.0 AIMS AND OBJECTIVES

In this lesson, we are going to discuss about linked list, single linked list with header, single linked list without header and performing the operations insertion and deletion.

After learning this lesson, we should be able to

- understand the concepts behind linked list
- identify single linked list with header
- distinguish Single linked list without header
- perform insertion and deletion operations

18.1 INTRODUCTION

Data representation have the property that successive nodes of data object were stored at a fixed distance. Thus,

(i) if element aij table stored at the location Lij
(ii) if the ith node in a queue at location Li.
(iii) if the top most node of the stack will be at the location Lt.

When a sequential mapping is used for ordered list, operations like insertion and deletion becomes expensive.

For eg., consider some English words. Three letter word ending with “AT” like BAT,CAT,EAT,FAT,HAT,…….,WAT. If we want to add GAT in the list we have to move BAT,CAT…..,WAT.If we want to remove the word LAT then delete many from the list.The problem is list having varying sizes, so sequential mapping is in adequate.

By storing each list with maximum size storage space may be wasted. By maintaining the list in a single array large amount of data movement is needed. The data object polynomial are ordered by exponent while matrices ordered rows and columns. The alternative representation to minimize the time needed for insertion and deletion is achieved by linked representation.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

18.2 LINKED LIST WITH HEADER

The following steps are used to create a linked list with header.

1. **Three pointers –header, first and rear** –are declared. The header pointer is initially initialized with NULL. For example, header = NULL, where header is pointer to structure. If it remains NULL it implies that the list has no element. Such a list is known as a NULL list or an empty list.

2. In the second step, memory is allocated for the first node of the linked list. For example, let the address of the first node be 1888. An integer, say 8, is stored in the variable num and the value of header is assigned to pointer next.

Header First first->next

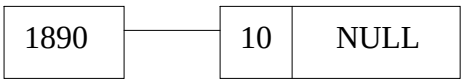
Both header and rear are initialized the address of the first node.

The statement would be
Header=first;
Rear=first;

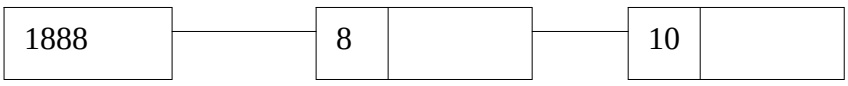
3. The address of pointer first is assigned to pointers header and rear. The rear is used to identify the end of the list and to detect the NULL pointer.

4. Again, create a memory location, suppose 1890, for the successive node.
node node->next

node node->next



5. Join the element of 1890 by assigning the value of node rear->next. Move the rear pointer to the last node.



Consider the following statements

1. node->next=NULL;

The above statement assigns NULL to the pointer next to current node. If the user does not want to create more nodes, the linked list can be closed here.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

2. rear->next=node;

The rear pointer keeps track of the last node,the address of current node(node) is assigned to link field of the previous node.

3. rear=node;

Before creating the next node, the address of the last created node is assigned to pointer rear, which is used for statement(2). In function main(), using while loop the elements of the linked list are displayed.

Header: The pointer header is very useful in the formation of a linked list. The address of first node(1888) is stored in the pointer header. The value of the header remains unchanged until it turns out to be NULL. The starting location of the list ca only be determined by the pointer header.

```
While ( header!=NULL)
{
printf("%d", header->num);
header=header->next;
}
```

18.3 LINKED LIST WITHOUT HEADER

In the last topic we discussed how a linked list can be created using header. The creation of a linked list without header is same as that of a linked list with header. The difference in manipulation is that, in the linked list with header, the pointer header contains the address of the first node. In the without header list, pointer first itself is the starting of the linked list.

OPERATIONS OF A SINGLY LINKED LIST

INSERTION

Insertion of an element in the linked list leads to several operations. The following steps are involved in inserting an element.

Creation of node: Before insertion, the node is created. Using malloc () function memory space is booked for the node.

Assignment of data: Once the node is created, data values are assigned to members.

Adjusting pointers: The insertion operation changes the sequence. Hence, according to the sequence, the address of the next element is assigned to the inserted node. The address of the current node (inserted) is assigned to the previous node.
The node can be inserted in the following positions in the list.

Insertion of the node at the starting: The created node is inserted before the first element. After insertion, the newly inserted element will be the first element of the linked list. In this insertion only the contents of the successive node's pointer are changed.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Insertion at the end of the list: A new element is appended at the end of the list. This is easy as compared to other two (a) and (c) operations. In this insertion only the contents of the previous node's pointer are changed.

Insertion at the given position in the list: In this operation, the user has to enter the position number. The given pointer is counted and the element is inserted. In this insertion contents of both the previous and next pointers are altered.

Check Your Progress

Ex 1) What are the three pointers required to create a linked list with header?

2) Using _____function memory space is booked for the node.

Insertion of the Node at the starting

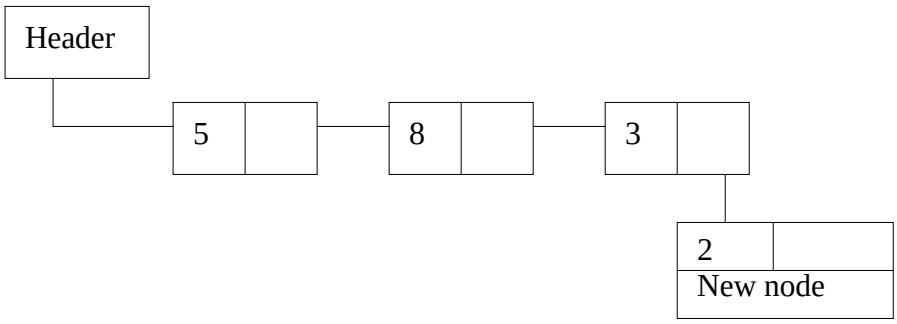
Inserting an element at the beginning involves updating links between link fields of two pointers. After insertion of new node, the previously existing nodes are shifted ahead.

The new node, which is to be inserted, is formed and the arrow indicates the position where it will be inserted.

After insertion, the new node will be the first node and its link field points to the second element, which was previously the first element.

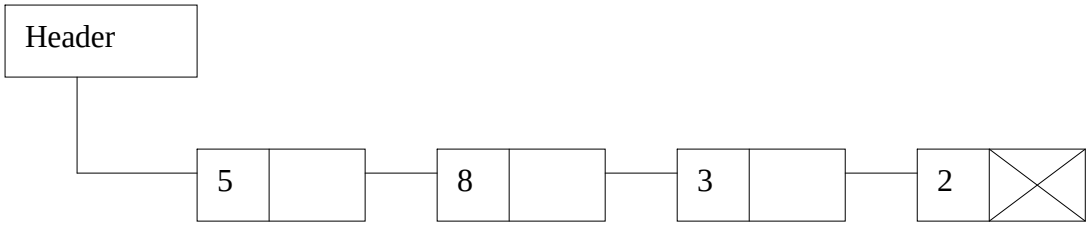
Insertion of the Node at the end

A new element is inserted or appended at the end of the existing linked list. The address of the newly created node is linked to the previous node that is NULL. The new node link field is assigned a NULL value.



(a) Before insertion

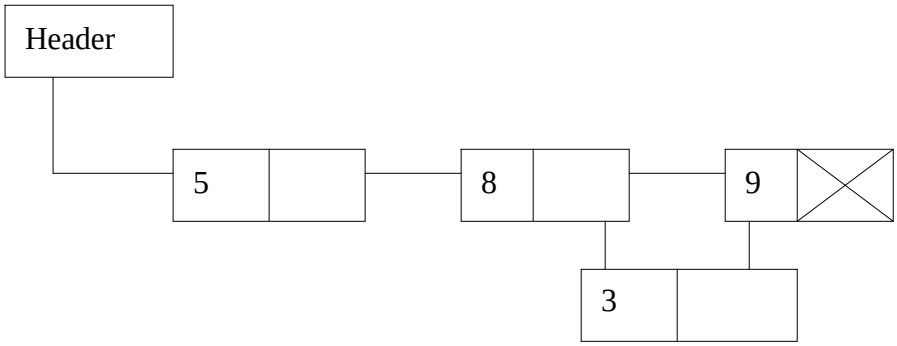
This watermark does not appear in the registered version - http://www.clicktoconvert.com



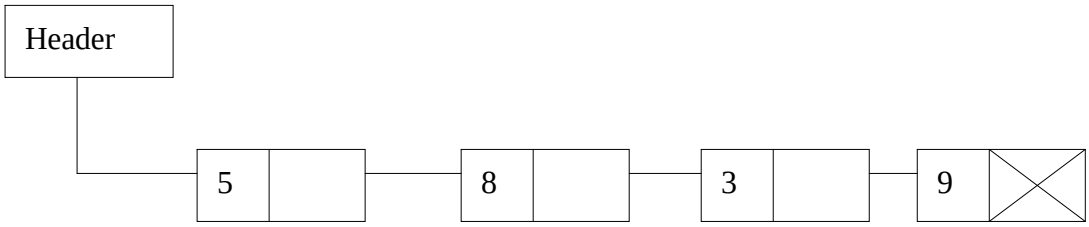
(b) After insertion

Insertion of a Node at a given position

Insertion of a node can be done at a specific position in the linked list. The following figure explain the insertion of a node at the specific position in the linked list. Suppose we want to insert a node at the third position, then



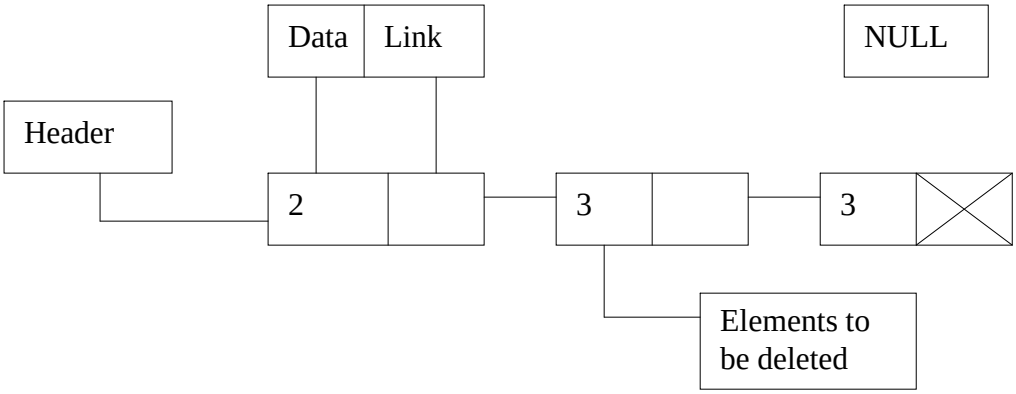
(a) Formed Linked list



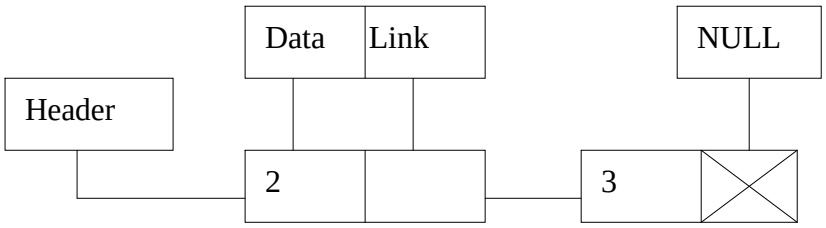
(b) Linked list after insertion

This watermark does not appear in the registered version - http://www.clicktoconvert.com

DELETION



(a) Deletion from the linked list



(b) Linked list after deletion of element 3

In figure (a) linked list elements are shown. The element 3 is to be deleted. After deletion the linked list would be shown as in figure(b).

Deleting a node from the list has the following three cases.

1. Deleting the first node
2. Deleting the last node
3. Deleting the specific node

While deleting the node, the node which is to be deleted is searched first. If the first node is to be deleted, then the second node is assigned to the header. If the last node is to be deleted, the last but one node is accessed and its link field is assigned a NULL value. If the node which is to be deleted is in between the linked list, then the predecessor and successor nodes of the specified node are accessed and linked.

18.4 LET US SUM UP

In this lesson, we have learnt

- the concept of linked lists with and without header.
- the operations like insertion and deletion performed with list.
- the position of the node before and after insertion in a list.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

18.5 LESSON END ACTIVITIES

1. What is a linked list?
2. Bring out the differences between linked list with and without header
3. Specify the three cases that occur in deletion of a node.

18.6 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) The three pointers required to create a linked list with header are Header, First, Rear

2) Using malloc**()** function memory space is booked for the node.

18.7 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source,1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON - 19

DOUBLY LINKED LIST

19.0 AIMS AND OBJECTIVES

In this lesson, we are going to study about doubly linked list, performing insertion and deletion operation and the difference between single and doubly linked list

After reading this lesson, we must be able to

- differentiate single and doubly linked list
- know how the circular list works in doubly linked list

19.1 INTRODUCTION

The doubly linked list has two link fields one linking in forward direction and another one in backward direction.

19.2 DOUBLY LINKED LIST

If we point to a specific node say P, we can move only in the direction of links. The only way to fine which precedes P is to start back at the beginning. The same is followed to delete the node. The problem is to which direction the link is to be moved. In such cases doubly linked list is used. It has two link fields one linking in forward direction and another one in backward direction.

It has atleast three fields as follows:

It has one special node called head node. It may or may not be C**ircular.** An empty list is not really empty since it will always have its head node.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Insert algorithm:

Procedure DINSERT (P,X)
(insert node p to right of node x)
LLINK (P) ← X
RLINK (P) ← RLINK (X) (set LLINK & RLINK of nodes)
LLINK (RLINK (X) ← P
End Dinsert

Delete algorithm:

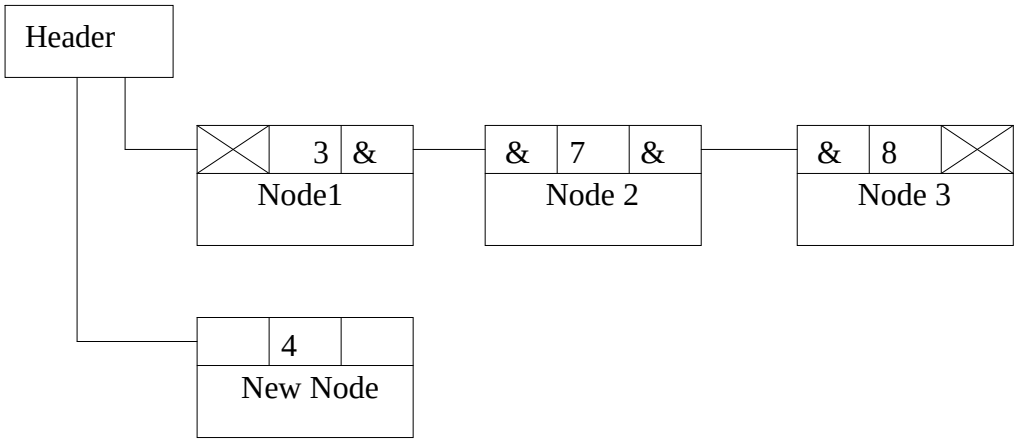
Procedure DDELETE (X,L)
If X=L then no more nodes (L ← list)
RLINK (LLINK (X)) ← RLINK (X)
LLINK (RLINK(X)) ← LLINK(X)
Call RET (x)
End DDelete

19.2.1 INSERTION

Insertion of an element in the doubly linked list can be done in three ways.

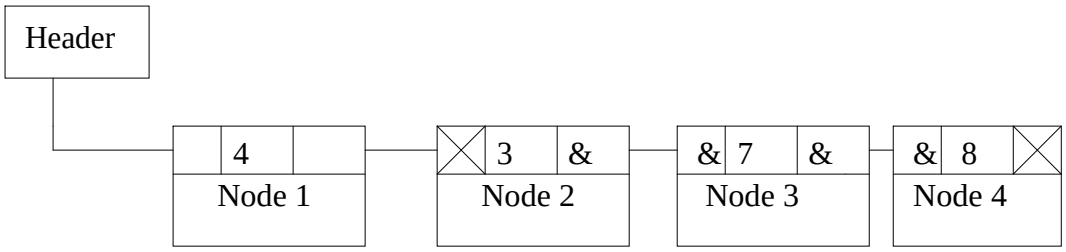
1. Insertion at the beginning
2. Insertion at the specified location
3. Insertion at the end

Insertion at the beginning: The process involves creation of a new node, inputing data and adjusting the pointers.



(a) Before insertion

This watermark does not appear in the registered version - http://www.clicktoconvert.com



(b) After insertion

Insertion at the end or a specified position: An element can be inserted at a specific position in the linked list.The following program can be used to insert an element at a specific position.

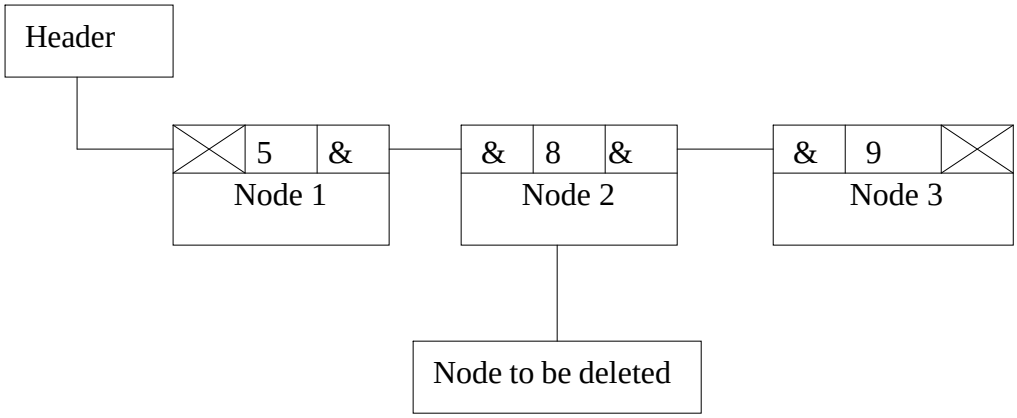
```
int addafter(int n, int p)
{
  struct  doubly *temp, *t;
  int j;
  t= first;

  for(j=0;j<p-1;j++)
{
  t=t-next;
  if(t= =NULL)
 {
    printf("\n There are less than %d element:",p);
     return 0;
}
temp=(struct doubly*) malloc(sizeof(struct doubly));
temp->next=t->next;
temp->number=n;
t->next=temp;
return 0;
}
```

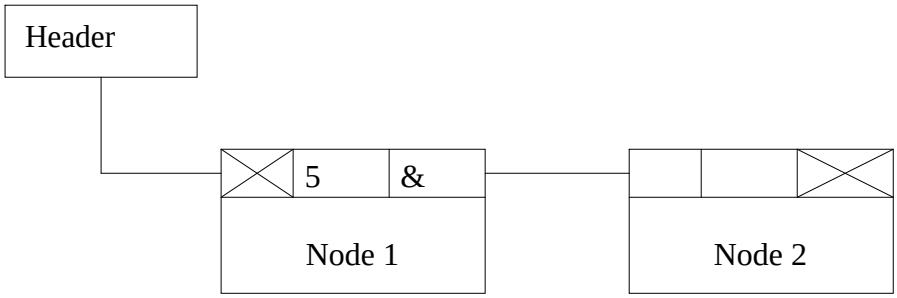
19.2.2 DELETION

An element can be removed from the linked list by an operation called deletion. Deletion can be done at the beginning or at a specified position.

This watermark does not appear in the registered version - http://www.clicktoconvert.com



(a) Before deletion



(b) After deletion

In the deletion process the memory of the node to be deleted is released using free( ) function. The previous and next node are linked.

Check Your Progress

Ex 1) The element can be inserted anywhere with the help of

---

2) Mention the two types of links used in doubly linked list.

19.3 DIFFERENCE BETWEEN SINGLE AND DOUBLY LINKED LIST

Single linked list	Doubly linked list
Only one link is available to show the next node	Two links are available to show the node as LLINK and RLINK
There is no head-node	It has a special node called as Head-node
Circular links are not available	Circular links are possible with the help of the two links
Insertion and deletion are expensive	Insertion and deletion are easily possible
More space is wasted due to unusage of memory allocation	Empty space is not present because in empty space the head node is present

This watermark does not appear in the registered version - http://www.clicktoconvert.com

19.4 LET US SUM UP

In this lesson, we have learnt about

- doubly linked list
- insertion operation
- deletion operation
- the Difference between single and doubly linked list

19.5 LESSON END ACTIVITIES

1. What is doubly linked list? Explain in detail
2. Differentiate between single and doubly linked lists.
3. Explain the insertion and deletion operation in a doubly linked list

19.6 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) The element can be inserted anywhere with the help of
Circular link

2) Mention the two types of links used in doubly linked list.
   - L link
   - R link

19.7 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source,1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON – 20

QUEUES

20.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about Queues and the operations that can be performed on queues.

After learning this lesson, we should be able to

- understand queues
- know rear and front of a queue
- the insertion operation on queues
- The deletion operation on queues

20.1 INTRODUCTION

Queue is used for the application of job scheduling. It is used in batch processing .when the jobs are queued –up and executed one after the other they were received.

20.2 QUEUES

Queues ignore the existence of priority. There are two different ends called FRONT and REAR end. Deletions are made from the FRONT end. If job is submitted for execution it joins at the REAR end. job at the front of queue is next to be executed.

Queue is an common data objects found in computer algorithm. In queue the insertions are made at one end called rear and the deletion at one end called front. Queue follows the FIFO basis (**First in First Out).**

Deletions are made from the Front end and joins at the REAR end. Job at the FRONT of queue is next to be executed. Some operations performed on queues are:

CREATE Q (Q)	← Creates an empty queue
ADD (i,Q)	←Add i to rear of queue
DELETE (Q)	← Removes the front element
FRONT (Q)	←Returns the front element of queue.
ISEMTQ	← returns true if queue is empty else false.

20.2.1 OPERATIONS OF A QUEUE

Add procedure on a queue:

Procedure ADDQ (item , Q, n, rear)

This watermark does not appear in the registered version - http://www.clicktoconvert.com

If rear = n then Queue- full
Rear <- rear +1
Q(rear) <- item
end ADDQ.

Delete Procedure of Queue:

Procedure DELETE Q(item, Q, Front)
If front = rear then queue –empty
Front <- front +1
Item <- Q(front)
End DeleteQ

Check Your Progress

Ex 1) Where are queues used?

Ex 2) Name the two ends of a Queue.

Queue- full does not imply that there are n elements in queue
Queue –full is to move the entire queue to the left so that the first element is again at Q(1)and front=0.

This is time consuming, especially when there are many elements in queue at the time of queue – full signal.

20.3 LET US SUM UP

In this lesson, we have learnt about

- the usage of front and rear end.
- performing the operations in Queue and how to represent it.

20.4 LESSON END ACTIVITIES

1. What is meant by Queue?
2. Mention the need of Front and Rear ends in queues

20.5 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) Queues are used in applications of Job scheduling and in Batch processing.
2) The two ends of a Queue are FRONT and REAR.

20.6 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source,1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

UNIT V

LESSON - 21

SEARCHING AND SORTING

21.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn what we mean by searching and the concepts behind linear searching.

After reading this lesson, we should be able to

- understand searching
- use linear search

21.1 INTRODUCTION

The searching of an item starts from the first item and goes up to the last item until the expected item is found. An element that is to be searched is checked in the entire data structure in a sequential way from starting to end. Sorting is a term used to arrange the data items in an order.

21.2 SEARCHING

The searching methods are classified into two types as Linear search and Binary search methods.

21.3 LINEAR SEARCH

The linear search is a usual method of searching data. The linear search is a sequential search. It is the simple and the easiest searching method. An element that is to be searched is checked in the entire data structure in a sequential way from starting to end. Hence, it is called linear search. Though, it is straight forward, it has serious limitations. It consumes more time and reduces the retrieval rate of the system. The linear or sequential name implies that the items are stored in a systematic manner. The linear search can be applied on sorted or unsorted linear data structure.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The number of iterations for searching depends on the location of an item. If it were located at first position then the number of iteration required would be 1. Here, least iterations would be needed hence, this comes under the best case.

In case the item to be searched is observed somewhere at the middle then the number of iterations would be approximately N/2, where N is the total number of iterations. This comes under average number of iterations. In the worst case to search an item in a list, N iterations would be required provided the expected item is at the end of the list.

Check Your Progress

Ex 1) Linear search is a _____ search.

2) The number of iterations for linear search depends on _____

21.4 LET US SUM UP

In this lesson, we have learnt about

- searching
- linear search

Knowing these concepts will certainly facilitate us to include them wherever we find necessary applications.

21.5 LESSON END ACTIVITIES

1. How will you search an element in a list?
2. Explain the linear search method.

21.6 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) Linear search is a sequential search.

2) The number of iterations for linear search depends on the key value.

21.7 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON - 22

BINARY SEARCH

22.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about the binary search method.
After reading this lesson, we should be able to know when to use linear search and binary search in our application programs.

22.1 INTRODUCTION

Next to linear and sequential search the better known methods for searching is binary search. This search begins by examining the record in the middle of file rather at any one end.

It takes 0(log n)time for the search. It begins by examining the record in the middle of file rather at any one end. File being searched is ordered by non-decrementing values of the key.

Based on the result of comparison with the middle key km one of the following conclusion is obtained.

a. If k < km record being searched in lower half of the file.
b. If k=km is the record which is begin searched for.
c. If k>km records begin searched in upper half of the file.

After each comparison either it terminates successfully or the size of the file searching is reduced to one half of the original size. 0(log2n). The file be examined at most [(n/2)^k] where n is the number of records , k is key. In worst case this method requires 0(logn) comparisons. Always in binary search method the key in the middle of subfile is currently examined. Suppose there are 31 records. The first tested is k16 because[(1+31)/2]=16. If k is less than k16 then k8 is tested next because [(1+15)/2]=8 or if k>k16 then k24 is tested & it is repeated until the desired record is reached.

Check Your Progress

Ex 1) Binary search begins examining the record _____

2) Can we apply binary search on unsorted data structure?

This watermark does not appear in the registered version - http://www.clicktoconvert.com

22.2. BINARY SEARCH ALGORITHM

```
Procedure BINSRCH
//search an ordered sequential file.F with records R1…..Rn and the keys
K1<=k2<=……..kn for a record Ri such that ki=k;
i=0 if there is no such record else ki=k
 Throughout the algorithm, l is the smallest index such that ki may be k and u the largest
index such that ku may be k//
          l <-1;u<-n
    while l<=u do
m<-[(l+u)/2]//compute index of middle record//
case:
   :k>km:l,-m+1//look in upper half//
   :k=km:l<-m; return
   :k<km:l<-m-1//look in lower half//
 end
end
I<-0//no record with key k//
end BINSRCH
```

22.3 LET US SUM UP

In this lesson, we have learnt about

- the binary search method
- the algorithm used for Binary search

22.4 LESSON END ACTIVITIES

1. What is meant by binary search?
2. Write a procedure to search an element using binary method

22.5 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) Binary search begins examining the record in the middle of file.

2) No, we can not apply binary search on unsorted data structure.

22.6 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON -23

SORTING

23.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about sorting and insertion and selection sort methods.

After reading this lesson, we should be able to

- understand Insertion sort
- understand Selection sort

23.1 INTRODUCTION

Sorting is an important phenomenon in the programming domain. If large volumes of records are available, then sorting of them becomes very crucial. If the records are sorted either in ascending or descending order based on keys, then searching becomes easy.

23.2 SORTING

Sorting is a process in which records are arranged in ascending or descending order. In real life we come a cross several examples of such sorted information. For example, in a telephone directory the names of the subscribers and their phone numbers are written in ascending alphabets. The records of the list of these telephone holders are to be sorted by their names. By using this directory, we can find the telephone number and address of the subscriber very easily. The sort method has great impact on data structures in our daily life. For example, consider the five numbers 5,9,7,4,1.

The above numbers can be sorted in ascending or descending order.

The representations of these numbers in
Ascending order (0 to n): 1 4 5 7 9
Descending order (n to 0): 9 7 5 4 1
Similarly, alphabets can be sorted as given below.
Consider the alphabets B, A, D, C, E. These are sorted in
Ascending order (A to Z): A B C D E
Descending order (Z to A): E D C B A.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

INSERTION SORT

In insertion sort an element is inserted at the appropriate place. Here the records must be sorted from R1,R2,…….Rn in non – decreasing value of the key k. Assume n is always greater than 1.

Algorithm:

```
Procedure INSORT(R,N)
K0← -α
For I ←2 TO N DO
T ← RJ
Call INSERT (T,J-1)
END {for}
END {INSORT}

Procedure INSERT (R,I)
    (Insert record R with key K into the ordered sequence such that resulting
    ordered sequence is also ordered on key K. Assume that R0 is a dummy record such
    that K>=K0)
        j ← I
        while K<kj do
        Rj+1 ← Rj
        J← j-1
        END {while}
        Rj+1 ← R
        End {insert}
```

case 1:

```
K0 ← -α
for j ← 2 to 6 do
T ← R2
Call Insert (R2,1)
Procedure insert (R,I)
j←I  i←1  (i.e.) j=1

while K<kj do
3<5              (True)
R2←R1
j←1-1 = 0 (ie.-j=0)
end {while}
R0+1←R
R1←R
R1 ← 3  (ie.R1=3)
```

This watermark does not appear in the registered version - http://www.clicktoconvert.com

SELECTION SORT

The selection sort is nearly same as exchange sort. Assume we have a list containing n elements. By applying selection sort, the first element is compared with remaining (n-1) elements. The smallest element is placed at the first location. Again, the second element compared with the remaining (n-2) elements. If the item found is lesser than the compared elements in the remaining n-2 list than the swap operation is done. In this type, the entire array is checked for the smallest element and then swapped.

In each pass, one element is sorted and kept at the left. Initially the elements are temporarily sorted and after next pass, they are permanently sorted and kept at the left. Permanently sorted elements are covered with squares and temporarily sorted with encircles. Element inside the circle 'O' is chosen for comparing with the other elements marked in a circle and sorted temporarily. Sorted elements inside the square 'y' are shown.

Time Complexity

The performance of sorting algorithm depends upon the number of iterations and time to compare them. The first element is compared with the remaining n-1 elements in pass 1. Then n-2 elements are taken in pass 2. This process is repeated until the last element is encountered. The mathematical expression for these iterations will be equal to (n-1) + (n-2) +…..+ (n-(n-1)). Thus the expression becomes n*(n-1) / 2.

Thus the number of comparisons is proportional to (n^2). The time complexity of selection sort is O (n^2).

Check Your Progress

Ex 1) What is sorting?

2) Point out the critical parameters that affect the performance of sorting algorithms.

23.3 COMPARISON WITH OTHER METHODS

1. This method requires more number of comparisons than quick sort and tree sort.
2. It is easier to implement than other methods such as quick sort and tree sort. The performance of the selection sort is quicker than bubble sort.

Consider the elements 2, 6, 4, 8 and 5 for sorting under selection sort method.

1. In pass 1, select element 2 and check it with the remaining elements 6, 4, 8 and 5. There is no smaller value than 2, hence the element 2 is placed at the first position.
2. Now, select element 6 for comparing with remaining (n-2) elements i.e., 4, 8, and 5. The smaller element is 4 than 6. Hence we swap 4 and 6 and 4 is placed after 2.
3. Again we select the next element 6 and compare it with the other two elements 8, 5 and swapping is done between 6 and 5.
4. In the next pass element 8 is compared with the remaining one element 6. In this case, 8 is swapped with 6.
5. In the last pass the 8 is placed at last because it is highest number in the list.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Pass	No. 1	2	3	4	5
1	2	6	4	8	5
2	2	4	6	8	5
3	2		5	8	6
4	2	4		6	8
5	2	4	5		
Sorted No.	2	4	5	6	8

23.4 LET US SUM UP

In this lesson, we have learnt

- Insertion sort
- Selection sort

23.5 LESSON END ACTIVITIES

1. What is sorting?
2. What is Insertion sorting method?
3. Explain in detail about the selection sort method?

23.6 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) Sorting is a process in which the records are arranged in ascending or descending order.

2) The critical parameters that affect the performance of sorting algorithms are the number of iterations and the time to compare elements.

23.7 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.

Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein: Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON – 24

BUBBLE & QUICK SORTING

24.0 AIMS AND OBJECTIVES

In this lesson, we are going to study about Bubble sort and Quick sort.

After reading this lesson, we should be able to

- understand the Bubble sort method
- understand the Quick sort method

24.1 INTRODUCTION

Bubble sort is a commonly used sorting algorithm and it is easy to understand. In this type, two successive elements are compared and swapping is done if the first element is greater than the second one. The elements are sorted in ascending order. Though it is easy to understand, it is time consuming. The quick sort method works by dividing into two partitions.

24.2 BUBBLE SORT

The bubble sort is an example of exchange sort. In this method comparison is performed repetitively between the two successive elements and if essential swapping of elements is done. Thus, step-by-step the entire array elements are checked. It is different from the selection sort. Instead of searching the minimum element and then applying swapping, two records are swapped instantly upon noticing that they are not in order.

Let us consider the elements 9, 5, 1, 4 and 3 for sorting under bubble sort.

1. In pass 1, first element 9 is compared with its next element 5.The next element is smaller than the 9. Hence, it is swapped. Now the list is 5, 9, 1, 4, 3 again the 9 is compared with its next element 1 the next element is smaller than the 9 hence swapping is done. The same procedure is done with the 4 and 3 and at last we get the list as 5, 1, 4, 3, 9.

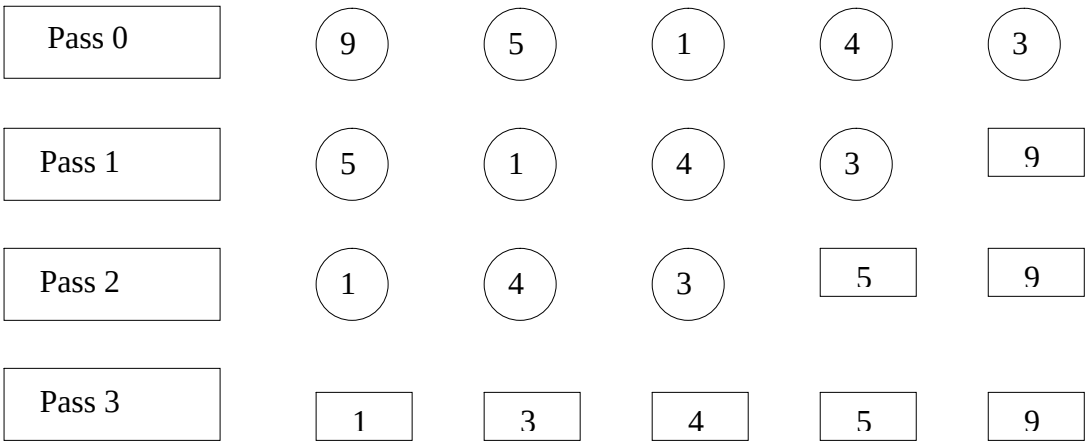
This watermark does not appear in the registered version - http://www.clicktoconvert.com

2. In pass 2, the elements of pass 1 are compared. The first element 5 is compared with its next element 1.5 and 1 are swapped because 1 is smaller than 5. Now the list becomes 1, 5, 4, 3, 9. Next, 5 is compared with element 4. Again, the 5 and 4 are swapped. This process is repeated until all successive elements are compared and if the succeeding number is smaller than the present number then the numbers are swapped.

   The final list at the end of this pass is 1, 4, 3, 5, 9.

3. In pass 3, the first element 1 is compared with the next element 4. The element 4 is having the higher value than the first element 1, hence they remain at their original positions. Next 4 is compared with the subsequent element 3 and swapped due to smaller value of 3 than 4.

4. At last, the sorted list obtained is as 1, 3, 4, 5, 9.



24.2.1 TIME COMPLEXITY

The performance of bubble sort in worst case is n(n-1)/2. This is because in the first pass n-1 comparisons or exchanges are made; in the second pass n-2 comparisons are made. This is carried out until the last exchange is made. The mathematical representation for these exchanges will be equal to (n-1) + (n-2) +…. + (n(n-1)). Thus the expression becomes n*(n-1)/2.

Thus, the number of comparisons is proportional to (n^2). The time complexity of bubble sort is 0 (n^2).

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Check Your Progress

Ex 1) Write a few points on Bubble sort.

---

2) Bubble sort is time consuming. Is it True?

24.3 QUICK SORT

It is also known as partition exchange sort. It was invented by C A R Hoare. It is based on partition. Hence, the method falls under divide and conquer technique. The main list of element is divided into two sub-lists. For example, suppose lists of X elements are to be sorted. The quick sort marks an element in a list called as pivot or key. Consider the first element J as a pivot. Shift all the elements whose value is less than J towards the left and elements whose value is grater than J to right of J. Now, the key element divides the main list in to two parts. It is not necessary that selected key element must be at middle. Any element from the list can act as key element. However, for best performance is given to middle elements. Time consumption of the quick sort depends on the location of the key in the list.

Consider the following example in which five elements 8, 9, 7, 6, 4 are to be sorted using quick sort.

Consider pass 1 under which the following iterations are made. Similar operations are done in pass 2, pass 3, etc.

In iteration 1 the first element 8 is marked as pivot one. It is compared with the last element whose value is 4. Here 4 is smaller than 8hence the number are swapped. Iteration 2 shows the swapping operation.

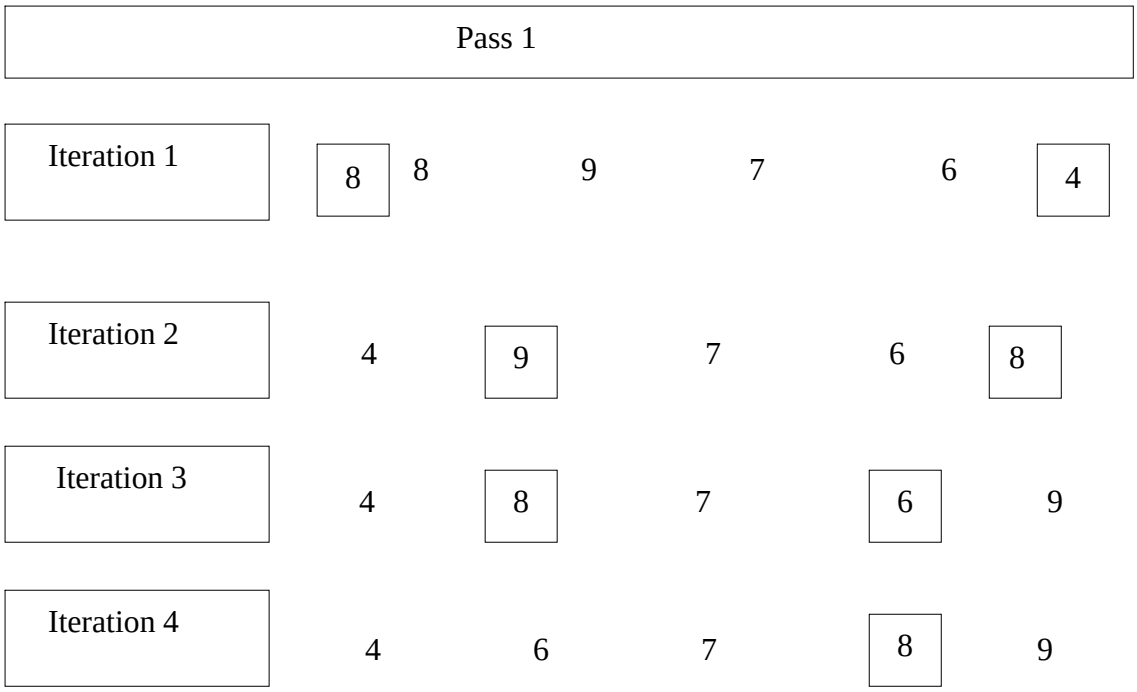
In the iteration 3 and 4, the position of 8 is fixed. In iteration 2, 8 and 9 are compared and swapping is done after Iteration 2.

In iteration 3, 8 and 6 are compared and necessary swapping is done. After this, it is impossible to swap. Hence, the position of 8 is fixed. Because of fixing the position of 8 the main list is divided into two parts. Towards left of 8 elements having smaller than 8 are placed and towards the right greater than 8 are placed.

Towards the right of 8 only one element is present hence there is no need of further swapping. This may be considered under Pass2.

However towards the left of 8 there are three elements and these elements are to be swapped as per the procedure described above. This may be considered under Pass3.

This watermark does not appear in the registered version - http://www.clicktoconvert.com



24.3.1 TIME COMPLEXITY

The efficiency of quick sort depends upon the selection of pivot. The pivot can bifurcate the list into compartments. Sometimes, the compartments may have the same sizes or dissimilar sizes. Assume a case in which pivot is at middle. Thus, the total elements towards lift of it and right are equal.

We can express the size of list with the power of 2. The mathematical representation for the size is $n=2^s$.

The value of s can be calculated as $\log_2 n$.

After the first pass is completed there will be two compartments having equal number of elements that is, n/2 elements are on the left side as well as right side. In the subsequent pass, four portions are made and each portion contains n/4 elements. In this way, we will be proceeding further until the list is sorted. The number of comparisons in different passes will be as follows.

Pass 1 will have n comparisons. Pass 2 will have 2*(n/2) comparisons. In the subsequent passes will have 4*(n/4), 8*(n.8) comparisons and so on. The total comparisons involved in this case would be O(n)+O(n)+O(n)+…..+s. The value of expression will be O (n log n).

Thus time complexity of quick sort is O(n log n).

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Check Your Progress

Ex 3) Quick sort falls under _____ technique.

4) The efficiency of quick sort depends upon the _____

Comparison with other methods

1. This is the fastest sorting method among all.
2. It is somewhat complex, so a little difficult to implement than other sorting algorithms.

16.7 LET US SUM UP

In this lesson, we have learnt about

- the bubble sort method
- the quick sort method

24.5 LESSON END ACTIVITIES

1. Explain bubble sort method with an example
2. Describe in detail the quick sort method.
3. Explain the purpose of pivot element in quick sorting

24.6 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) The bubble sort is an example of exchange sort. In this method comparison is performed repetitively between the two successive elements and if essential swapping of elements is done

2) Yes, it is true that Bubble sort is time consuming.

3) Quick sort falls under divide and conquer technique.

4) The efficiency of quick sort depends upon the selection of pivot.

24.7 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

LESSON – 25

TREE SORT & HEAP SORT

25.0 AIMS AND OBJECTIVES

In this lesson, we are going to learn about Tree sort and Heap sort.

After reading this lesson, we should be able to

- understand the tree sort
- understand the heap sorting methodologies.

25.1 INTRODUCTION

If the nodes in the binary tree are in specific prearranged order, then heap sorting method can be used. **A Heap** is defined to be a complete binary tree with a property that the value of each node is at least as large as the value of its children nodes.

25.2 TREE SORT

In binary tree, we know that the elements are inserted according to the value greater or less in between node and the root in traversing. If the value is less than traversing node then, insertion is done at left side. If the value is greater than traversing node, it is inserted at the right side. The elements of such a tree can be sorted. This sorting involves creation of binary tree and then in order traversing is done.

25.3 HEAP SORT

In heap sort, we use the binary tree, in which the nodes are arranged in specific prearranged order. The rule prescribes that each node should have bigger value than its child node. The following steps are to be followed in heap sort.

1. Arrange the nodes in the binary tree form.
2. Node should be arranged as per specific rules.
3. If the inserted node is bigger than its parent node then replace the node.
4. If the inserted node is lesser than its parent node then do not change the position.
5. Do not allow entering nodes on right side until the child nodes of the left are fulfilled.

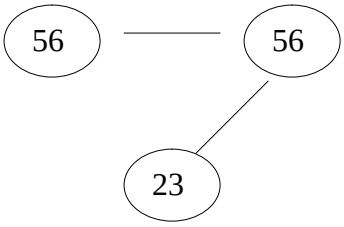
This watermark does not appear in the registered version - http://www.clicktoconvert.com

6. Do not allow entering nodes on left side until the child nodes of the right are fulfilled.
7. The procedure from step 3 to step 6 is repeated for each entry of the node.

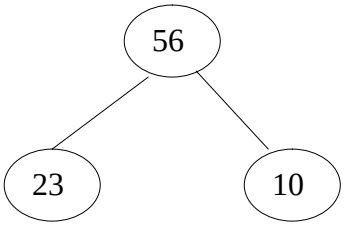
Consider the numbers 56, 23, 10, 99, 6, 19, 45, 45, 23 for sorting using heap. Sorting process is shown in the below figure.

1. At first, we pick up the first element 56 from the list. Make it as the root node.
2. Next, take the second element 23 from the list. Insert this to the left of the root node 56. Refer to Fig. 16.7(2).
3. Then take the third element 10 from the list for insertion. Insert it to the right of the root node.
4. Take the fourth element 99. Insert it to the left side of node 23. The inserted element is greater than the parent element hence swap 99 with 23. But the parent node 56 is smaller than the child node 99 hence 99 and 56 are swapped. After swapping 99 becomes the root node.
5. Consider the next element 6 to insert in the tree. Insert it at the left side. There exists a left node hence insert it to the right of 56.
6. Element 19 is inserted to the right side of 99 because the left side gets full. Insert the element 19 to the right side of node 10. However, the parent element is lesser than the child hence swap 19 with 10.
7. Now element 45 is to be inserted at the right side of 19. However, the parent element is having value lesser than the child element hence swap 45 with19.
8. Now the right side is fully filled hence add the next element 45 to the left. The element 45 is inserted at the last node of left i.e., 23. However, the parent element is having value lesser than the child element hence swap 45 with 23.
9. Insert the last element 23 to the left side node i.e. 45. The left of 45 is already filled hence insert element 23 at the right of 45. 10. At last, we get a sorted heap tree.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

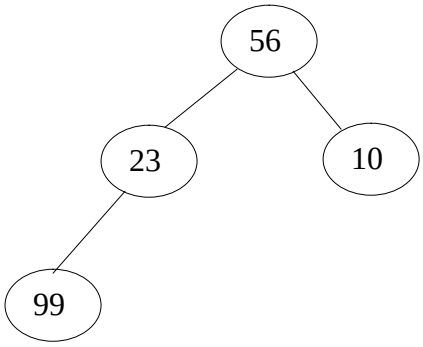


(1)

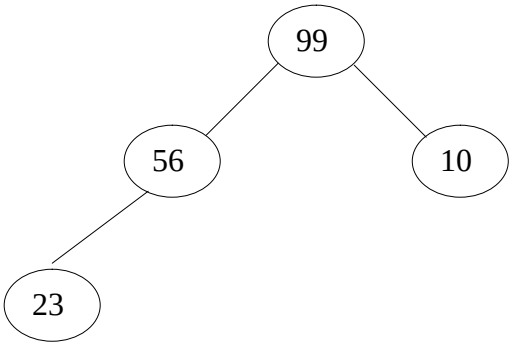


(2)

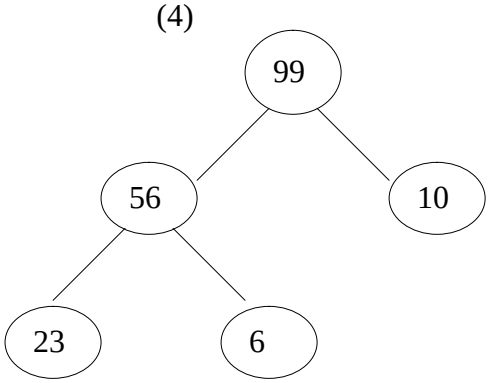
(3)



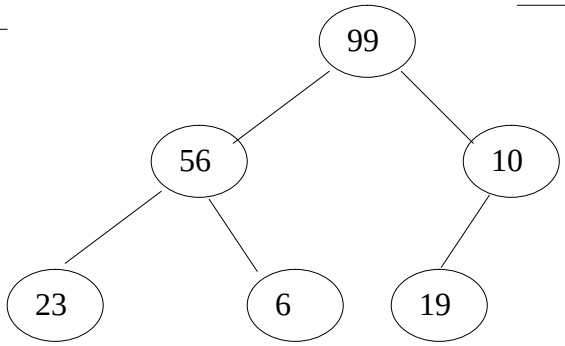
(4)



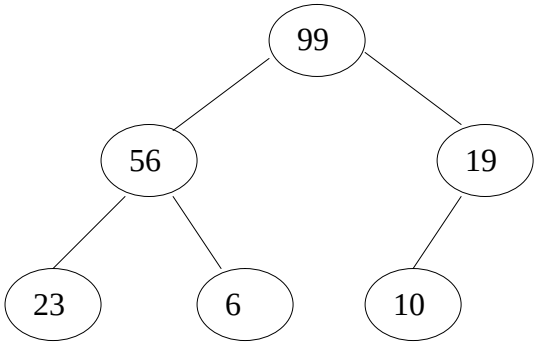
(5)



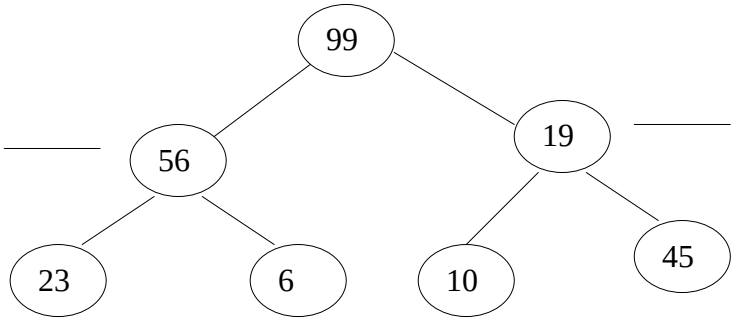
(6)



(7)

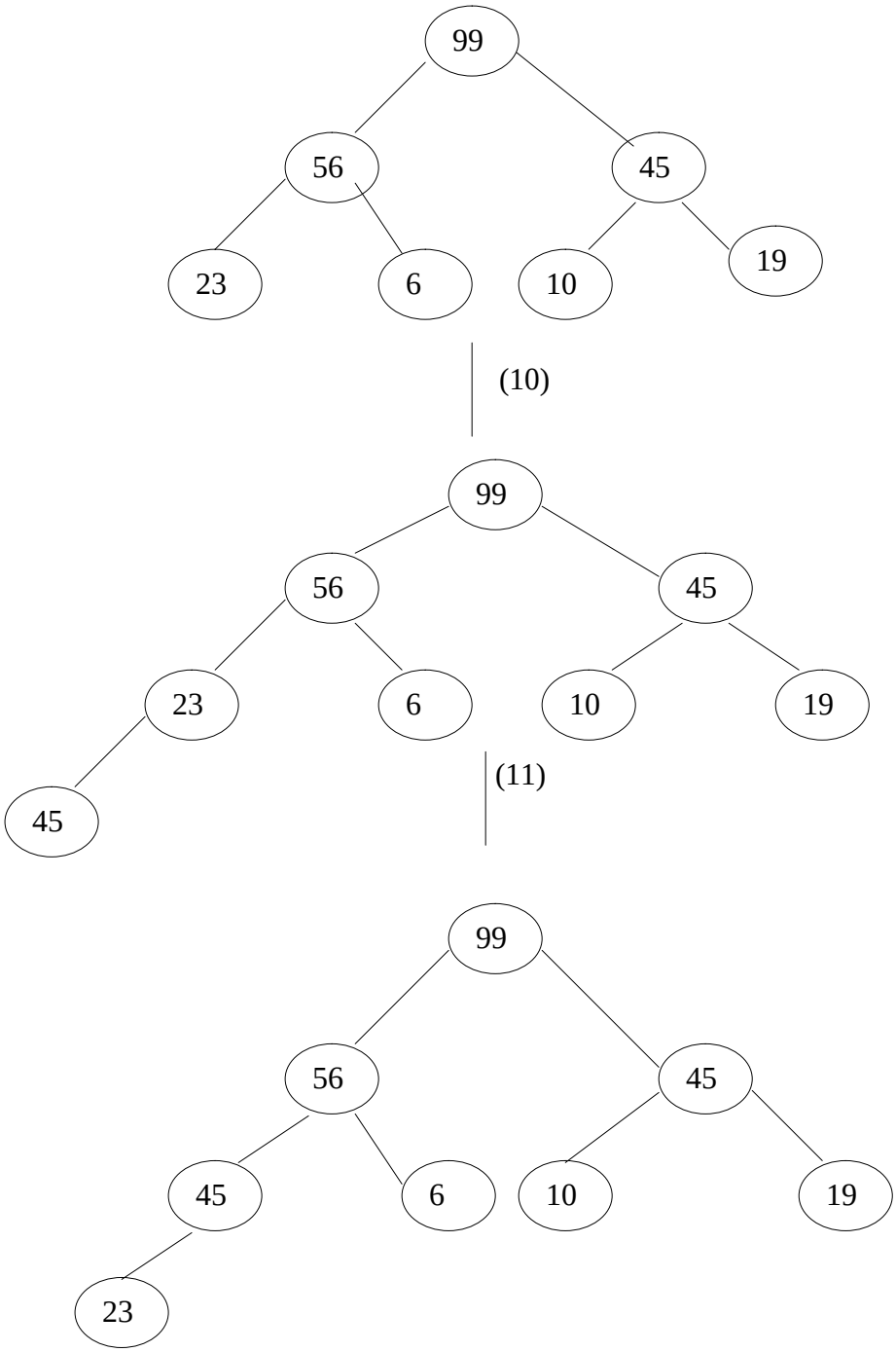


(8)



(9)

This watermark does not appear in the registered version - http://www.clicktoconvert.com



This watermark does not appear in the registered version - http://www.clicktoconvert.com

Check Your Progress

Ex 1) State the rule prescribed by Heap Sort.
2) Define Heap.

25.4 LET US SUM UP

In this unit, we have learnt about

- Tree sort
- Heap sort

25.5 LESSON END ACTIVITIES

1. Write notes on tree sort method
2. Explain the Heap sorting method with an example.

25.6 MODEL ANSWERS TO CHECK YOUR PROGRESS

Ex 1) The rule prescribed by Heap Sort is that each node should have bigger value than its child node.

2) A Heap is defined to be a complete binary tree with a property that the value of each node is at least as large as the value of its children nodes.

25.7 REFERENCES

Ashok N Kamthane: "PROGRAMMING AND DATA STRUCTURES" – Pearson Education, First Indian Print 2004, ISBDN 81-297-0327-0.

E Balagurusamy: Programming in ANSI C, Tata McGraw-Hill, 1998.
Ellis Horowitz and Sartaj Sahni: Fundamentals of Data Structure, Galgotia Book Source, 1999.

Aaron M Tanenbaum, Yedidyeh langsam, Moshe J Augenstein:
Data Structure using C PHI PUB.