ITC213: STRUCTURED PROGRAMMING

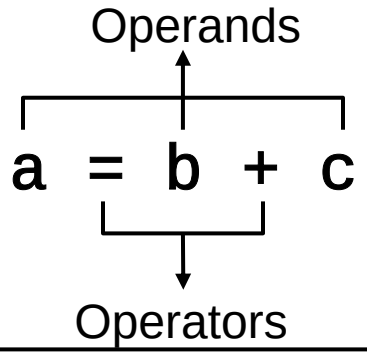
Bhaskar Shrestha
National College of Computer Studies
Tribhuvan University

Lecture 06: Operators and Expressions

Readings: Chapter 3

Operators and Operands

- C is very rich in built-in operators
- An operator is a symbol that instructs C to perform some operation, or action, on one or more operands
- An operand is something that an operator acts on
- Operators that require two operands are binary operators and operators that require one is unary



Operators Categories

- Arithmetic Operators:
  - `+`, `-`, `*`, `/`, `%`
- Assignment operators:
  - simple (`=`) and compound (`+=`, `-=`, `*=`, `/=`, etc)
- Unary Operators:
  - `-`, `++`, `--`, `sizeof`, `&`, `*`, (*type*) etc
- Relational Operators:
  - `>`, `<`, `>=` , `<=`,
- Equality Operators
  - `==`, `!=`
- Logical Operators:
  - `&&`, `||`, `!`
- Conditional Operator:
  - `? :`
- Bitwise Operators:
  - `&`, `|`, `~`

Arithmetic Operators (1/2)

- C's arithmetic operators perform mathematical operations such as addition and subtraction

Operator	Symbol	Action	Example
Addition	`+`	Adds two operands	`x + y`
Subtraction	`-`	Subtracts the second operand from the first operand	`x - y`
Multiplication	`*`	Multiplies two operands	`x * y`
Division	`/`	Divides the first operand by the second operand	`x / y`
Modulus	`%`	Gives the remainder when the first operand is divided by the second operand	`x % y`

Arithmetic Operators (2/2)

- These operators can be applied only to operands having numeric values
  - Thus, the operands can be integers, floating-point numbers and characters
- Integer division truncates any fractional part
- The expression `x % y` produces the remainder when `x` is divided by `y`, and thus is zero when `y` divides `x` exactly
  - The `%` operator cannot be applied to a `float` or `double`
- For `/` and `%`, second operand must be nonzero
- The interpretation of `%` operand is unclear, when one of the operands is negative

Arithmetic Operators Example

- Suppose that **a** and **b** are integer variables and **c** and **d** are floating-point variables. Also suppose that **c1** and **c2** is a character variable. Their initial values is shown below

  a = 11, b = 3, c = 12.5, d = 2.0, c1 = 'P', c2 = 'T'

Expression	Result values	Expression	Result values	Expression	Result values
a + b	14	c + d	14.5	c1	80
a – b	8	c + d	10.5	c1 + c2	164
a * b	33	c * d	25.0	c1 + c2 + 5	169
a / b	3	c / d	6.25	c1 + c2 + '5'	217
a % b	2	6 % b	0		

Type Conversion in Expressions

- When an operator has operands of different types, they are converted to a common type according to a small number of rules
- A type conversion always conserves the original value, if the new type is capable of representing it
- If there are no **`unsigned`** operands, the following rules apply:
  - If one of operands is **`long double`**, other is converted to **`long double`**
  - Else, If one of operands is **`double`**, other is converted to **`double`**
  - Else, If one of operands is **`float`**, other is converted to **`float`**
  - Else, If one of operands is **`long int`**, other is converted to **`long int`**
  - Otherwise, convert all **`char`** and **`short`** to **`int`**s (integral promotion)

Example

- Given
  - `int i = 7;`
    `float f = 5.5;`
    `char c = 'w';`

Expression	*Value*	*Type*
`i + f`	`12.5`	`float`
`i + c`	`126`	`int`
`i + c – ‘0’`	`78`	`int`
`(i + c) – (2 * f / 5)`	`123.8`	`float`

Operators Precedence and Associativity (1/2)

- If you use more than one operator in an expression, C uses precedence rule to find out which operator's calculation is to be formed
- When two operators have the same precedence, C uses associativity rule to determine which operation is to be carried out first
- The operators are grouped hierarchically according to their precedence
- Operations with a higher precedence are carried out before operations having a lower precedence

Operators Precedence and Associativity (2/2)

- The five arithmetic operators are divided into two precedence groups
  - `*`, `/` and `%` have higher precedence than `+` and `-`
- `3 + 4 * 5` is equivalent to `3 + (4 * 5)`
- Arithmetic operators are evaluated left to right
- `120 / 4 * 5` is equivalent to `(120 / 4) * 5`
- The natural order of evaluation can be altered through the use of parenthesis

Typecast

- The value of an expression can be converted to a different data type if desired
- To change the type of an expression, precede the expression with name of type enclosed in parenthesis
- For example, if `i` is a type `int`, the expression
  `(float)i`
  casts `i` to type `float`. In other words, the program makes an internal copy of the value of `i` in floating-point format
- A typecast, or simply cast does not change the data type associated with the expression itself

Assignment Operators

- Assignment operators assign value of an expression to an identifier
- The commonly used assignment operator is **=**
- **Syntax:** *identifier = expression*
- Example: `x = y;`
  - assign the value of `y` to `x`
- ***Type Conversion in Assignments:*** When expression of one type is assigned to a variable of different type, a type conversion will occur

Type Conversion in Assignments

- ***Conversion rule in assignment:*** The value of the right side (expression side) of the assignment is converted to the type of the left side (target variable)

Conversion	***Potential Problems***
Bigger floating-point type to smaller floating-point, such as double or float	Loss of precision (significant), value might be out of range for target type, in which case results is undefined
Floating-point type to integer type	Loss of fractional part, original value might be out of range for target type, in which case result is undefined
Bigger integer type to smaller integer type, such as long to short	Original value might be out of range for target type, typically just the low-order bytes are copied

Type Conversion Example

```
char ch;
int x;
float f;
ch = 'a';
x = 2635;
f = 123.23;
/* All these lines are assignment statements, that causes
type conversion */
ch = x; /* the left high-order bits of x are chopped off,
leaving ch with the lower 8 bits*/
x = f; /* x will receive the non-fractional part of f */
f = ch; /* f will convert the 8-bit integer value stored in
ch to the same value in the floating-point constant*/
f = x; /* f will convert an integer value into floating point
constant */
```

Multiple Assignment

- Multiple assignments of the form
  `identifier 1 = identifier 2 = ... = expression`
  are permissible in C
  - `x = y = z = 10`
- The assignments are carried out from right to left
  - `x = (y = (z = 10));`
- Assignment operators have lower precedence than other operators and their associativity is right to left

Compound Assignment

- C contains the following five additional assignment operators; **+=**, **-=**, **/=**, ***=** and **%=**, often known as compound assignment

```
x = x + 10
```

→

```
x += 10
```

When You Write This...	It Is Equivalent To This
`x *= y`	`x = x * y`
`y -= z + 1`	`y = y – (z + 1)`
`a /= b`	`a = a / b`
`x += y / 8`	`x = x + (y / 8)`
`y %= 3`	`y = y % 3`

Unary Operators

- Operators acting upon a single operand are unary operators
- Unary Minus(`-`)
  - Used to negate the value of the operand
- Increment and Decrement operators (`++` and `--`)
  - `++` operator causes its operand to be increased by `1`
  - `--` operator causes its operand to be decreased by `1`

Prefix and Postfix Increment and Decrement

- The increment/decrement operators can be written on either side of its operand
- ***Prefix:*** the operator is written before its operand (**++x**)
  - Increments/decrements the operand value and the resultant value is used in the expression
- ***Postfix:*** the operator is written before its operand (**x++**)
  - The current value of the operand is used in the expression and its value is incremented/decremented

Other Unary Operators

- The `sizeof` operator
  - A compile time operator and, when used with an operand, it returns the number of bytes the operand occupies
  - `printf("int takes %d bytes", sizeof (int));`
- The address of (`&`) operator
  - Returns the memory address of its operand
  - ```
    int count;
    printf("%p", &count);
    ```
- Typecast: (discussed earlier)
- Unary operators have higher precedence than arithmetic operators and their associativity is right-to-left
    ```

Relational and Equality Operators

- Relational and equality operators are used to compare expressions

Operator	**Symbol**	**Question Asked**	**Example**
Greater than	`>`	Is operand 1 greater than operand 2?	`x > y`
Less than	`<`	Is operand 1 less than operand 2?	`x < y`
Greater than or equal to	`>=`	Is operand 1 greater than or equal to operand 2?	`x >= y`
Less than or equal to	`<=`	Is operand 1 less than or equal to operand 2?	`x <= y`
Equal	`==`	Is operand 1 equal to operand 2?	`x == y`
Not equal	`!=`	Is operand 1 not equal to operand 2?	`x != y`

Relational and Equality Operators

- The four relational operators fall in same precedence group and have lower precedence than arithmetic operators and their associativity is left to right
- The two equality operators fall into a separate precedence group, beneath the relational operators and they also have left-to-right associavity
- The six operators are used to form relational expression and evaluates to either ***true*** or ***false***, of type int
  - True is represented by 1 and false is represented by 0

Logical Operators

- Logical operators let you combine two or more relational expressions into a single expression that evaluates to either true or false

Operator	Symbol	Example	What It Evaluates To
AND	`&&`	`exp1 && exp2`	True (**`1`**) only if both **`exp1`** and **`exp2`** are true; false (**`0`**) otherwise
OR	`||`	`exp1 || exp2`	True (**`1`**) if either **`exp1`** or **`exp2`** is true; false (**`0`**) only if both are false
NOT	`!`	`!exp1`	False (**`0`**) if **`exp1`** is true; true (**`1`**) if **`exp1`** is false

Logical Operators

- Suppose `i` is an integer variable whose value is `7`, `f` is a floating point variable whose value is `5.5`, and `c` is character constant that represents the character `'w'`

Expression	**What it evaluates to?**
`(i >= 6) && (c == 'w')`	`1` (true)
`(i >= 6) && (c == 199)`	`0` (false)
`(f < 11) && (i > 100)`	`0` (false)
`!(f > 5)`	`0` (false)

The Conditional Operator

- The conditional operator is C's only ***ternary operator***, meaning that it takes three operands
- A conditional expression is written in the from
  `exp1 ? exp2 : exp3`
  If **`exp1`** evaluates to true (that is, nonzero), the entire expression evaluates to the value of **`exp2`**. If **`exp1`** evaluates to false (that is, zero), the entire expression evaluates as the value of **`exp3`**

```
z = x > y ? x : y
```

Assigns the value of x to z if x is greater than y
Assigns the value of y to z if x is not greater than y

Operator Precedence Table

Precedence	***Operator category***	***Operators***	***Associativity***
1 (Highest)		`() [] -> .`	L → R
2	Unary operators	`- ++ -- ! sizeof (type) & (address of)`	R → L
3	Arithmetic multiply, divide and remainder	`* / %`	L → R
4	Arithmetic add and subtract	`+ -`	L → R
5	Bitwise shift operators	`<< >>`	L → R
6	Relational operators	`< <= > >=`	L → R
7	Equality Relational operators	`== !=`	L → R
11	Logical AND	`&&`	L → R
12	Logical OR	`||`	L → R
13	Conditional operator	`?:`	R → L
14	Assignment operators	`= *= /= %= += -= &= ^= |= <<= >>=`	R → L
15 (Lowest)	Comma operator	`,`	L → R

Functions

- An independent section of program code that performs a certain task and has been assigned a name
- Every C program consists of one or more function. The one and only required function is **`main()`**
- Two types of functions
  - Library functions, which are a part of the C compiler package
  - User-defined functions, which you, the programmer, create
- By referencing a function's name, your program can execute the code in the function
- The program also can send information, called arguments, to the function, and the function can return information to the main part of the program

Library Functions

- Library functions perform most of the common tasks (such as screen, keyboard, and disk input/output) your program needs
- Library functions that are functionally similar are usually grouped together as (compiled) object programs in separate library files

Function	***Type***	***Purpose***
`abs(i)`	`int`	Return the absolute value of i.
`cos(d)`	`double`	Return the cosine of d.
`fabs(d)`	`double`	Return the absolute value of d.
`getchar()`	`int`	Enter a character from the standard input device.
`log(d)`	`double`	Return the natural logarithm of d.
`pow(d1, d2)`	`double`	Return d1 raised to the d2 power.
`printf(...)`	`int`	Send values to the standard output device
`scanf(...)`	`int`	Enter values from the standard input device