Introduction to Computing

MCS1101B

Lecture 2

Recap

- Computation
  - How do computers compute - Von Neumann Architecture
- How do computer store data - memory
  - Bits and Bytes
  - There is a unique memory address for each Byte
- What can computer store - numbers
  - Binary number system
  - Operations on binary numbers
- Data, Information

Recap (contd.)

- Computer Programming
- Computer Programming Languages
  - Machine language - set of instructions
  - High-level programming languages - C, C++, Python
- Steps of Programming
  - Write a program using high-level programming language
  - Compile a program using a compiler to get the machine understandable code
  - Execute the program on machine
  - Anatomy of Programming

Anatomy of Programming

- Represent the problem formally
- Take a decision
  - Some tasks based on the decision
  - Evaluate outcomes
- Repeat until problem is solved.

Structure of a C Program

- They are a collection of functions
- Exactly one special function called “main” which must be present
- Each function has statements
  - e.g. declaration, assignment, condition check, looping
  - Statements are executed one by one

The Customary First C Code

The preprocessor

A function definition

Start of the function

A comment

A function call

A return value

End of the function

```c
#include <stdio.h>


int main (void)
{
    /* my first program in C */
    printf ("Hello, World! \n");
    return 0;
}
```

Things One Might Use in C Programming

- **Variables**
- **Constants**
- Expressions
  - Arithmetic, Logical, Assignment
- Statements
  - Declaration, Assignment,
  - Control Structures - conditional branching, looping
- Arrays
- Pointers
- Functions
- Structures

Variables and Constants

- A variables has a name, a memory address and a datatype
- Name
  - A sequence of letter and digits with first symbol being a letter or '_'
- Types of Variables
  - int, float, double, char, struct, pointer, array, void, etc
- Variables stored in memory
  - Therefore, each variable has an unique address
  - Each type has a predefined size - *typically they have standard values, but sometimes it may depend on the software/system you are using*
- Constants are basically read-only variables
  - Values, once assigned, cannot be changed

Expressions

- Variables and constants linked with operators
- Every expression evaluates to a value

- Arithmetic expressions
  - Uses arithmetic operators
  - Can evaluate to any value
- Logical expressions
  - Uses relational and logical operators
  - Evaluates to 0 (false) or 1 (true) only
- Assignment expressions
  - Uses arithmetic operators
  - Evaluates to value depending upon assignment

Arithmetic Operators

- Binary operators
  - Addition +
  - Subtraction –
  - Multiplication *
  - Division /
  - Modulus %
- Unary operators
  - Plus +
  - Minus –

- All operator except % can be used with operands of any data types
  - int, float, double, char

- % can be used only with integer operands

Operator Precedence

In decreasing order of priority

- Parenthesis ()
- Unary minus –
- Multiplication * , Division / and Modulus %
- Addition + and Subtraction –

- For same priority evaluation is done from **left to right** as they appear
- Parenthesis may be used to change the precedence of operator evaluation

Arithmetic Expressions (contd.)

Examples

- 1 + 2 * 3
  - 1 + (2 * 3) ⇒ 7
- 8 / 2 + 2 * 3
  - (8 / 2) + (2 * 3) ⇒ 10
- a - b + c + d
  - (((a - b) + c) + d)
- a * -b + d % e – f
  - a * (-b) + (d % e) - f

Let’s test your understanding …

- a + b + c * d * e ⇒ ?
- 10 / 5 ⇒ ?
- int a = 10, b = 20, c = 30, d ;
  float f;
  - d = b/a; d ⇒ ?
  - d = a/b; d ⇒ ?
  - d = c/b; d ⇒ ?
  - f = c/b; f ⇒ ?

Assignment Operator

- l-value = r-value
- l-value must be a variable where you can assign data
- r-value can be any valid for of expression
- The types of both side should usually be the same
- In the other case, r-value gets internally converted to the type of l-value
  - This can cause problems
  - e.g. int a; a = 2 * 3.3; ⇒ a = 6 and not 6.6

Assignment Expression

- Uses the assignment operator **=**
- General syntax:
  - variable_name = expression
  - l-value = r-value
- The value of the assignment expression is the value assigned to the l-value

- Examples:
  - a = 7 ⇒ 7
  - b = 2*7 - 11 ⇒ 3
  - c = x + y * 3 - z ⇒ whatever value the arithmetic expression x + y*3 - z evaluates to; depends on the value of x, y, z
  - a = a + 5 ⇒ 12

Assignment Expression (contd.)

- Several variables can be assigned to the same value using multiple assignment operators
  - a = b = c = 10
  - x = y = ‘a’
  - … and so on

- Multiple assignment operators are **right-to-left** associative
- Each of the assignment expressions are evaluates to a value and that value propagates to the next one

Assignment Operator Variations

- There are shortcuts for simple assignments
- +=, -=, *=, /=, %=

- a += b ⇒ a = a + b
- a *= 2 ⇒ a = a * 2
- … and so on.

- Let’s test your understanding …
- int a, b, c;
- Case 1
  - a = b = c = 5
  - a =? b = ? c =?
- Case 2
  - a=3; b= 5;
  - a += b+= 1;
  - a =? b = ?

Two More Variations

- Two unary variations which increments or decrement the value of the operand by 1
- Pre-increment operator, Post-increment operator ++
- Pre-decrement operator, Post-decrement operator --
  - a++ ⇒ a = a + 1
  - ++a ⇒ a = a + 1

- Both pre and post operators increment/decrement the value, but there is an important difference in the evaluated value of that expression
  - a = 3;
  - a++ ⇒ 3
  - ++a ⇒ 4

Logical Expressions

- Uses relational and logical operators
- This generally specifies a condition which can be either true or false

- Relational operators
  - > >=
  - < <=
  - == !=

… compares two quantities

Logical Expressions (contd.)

- Examples
  - x <= y
  - x == y
  - 1 == 2
  - x == y == 1
- (x + y < 6) ⇒ x + y < 6

- Evaluates to either 0 or 1
  - 0 ⇒ **false**
  - 1 ⇒ **true** ; *also non-zero values*
- Arithmetic expressions are evaluated first when on either side of a relational operator

Logical Operators

LOGICAL **AND**

&&

- 0 && 0 ⇒ 0
- 0 && non-zero ⇒ 0
- non-zero && 0 ⇒ 0
- non-zero && non-zero ⇒ 1

LOGICAL **OR**

||

- 0 && 0 ⇒ 0
- 0 && non-zero ⇒ 1
- non-zero && 0 ⇒ 1
- non-zero && non-zero ⇒ 1

LOGICAL **NOT**

!

- !0 ⇒ 1
- !non-zero⇒ 0
- aka. unary negation operator

Logical Expressions (contd.)

Examples

x = 1, y = 3, grades = ‘B’

- (x + y < 6) || (y >= 9) ⇒ 1
- (x == y) && (y != 5) ⇒ 0
- !(grades == ‘A’) ⇒ 1

Let’s test your understanding

- (!10) || (10 + 20 != 200) ⇒ ?
- (!10) && (10 + 20 != 200) ⇒ ?
- (4 > 3) && (100 != 200) ⇒ ?

- (x + y > 6) || (y >= 9) ⇒ ?
- (x = y) && (y == 1) ⇒ ?
- grades == ‘B’ ⇒ ?
- x = 3 && (y = 4) ⇒ ?

Bitwise Operators

- Operators that permits operation on individual bits
- Useful for low level programming such as controlling hardware

… we will discuss this operators in more details later on (if time permits).

- Bitwise AND &
- Bitwise OR |
- 1s complement ~
- Bitwise XOR ^
- Left shift <<
- Right shift >>

A Special Operator: AddressOf (&)

- Remember that each variable is stored at a location with an unique address
- Putting & before a variable name gives the address of the variable (where it is stored, not the value)
- Can be put before any variable (with no blank in between)
  - int a = 8;
  - printf (“value of a = %d, address of a=%d”, a, &a);
  - printf (“value of a = %d, address of a=%p”, a, &a);

Another Special Operator: **sizeof ()**

- This is a much used operator in C
- It is an unary operator
- It is used to compute the size of its operand in compile time
- It can used on any data type
  - sizeof (int), sizeof (char), *int a; sizeof (a)*, etc.

- It can be used with an expression as well – then it returns the size of the final value
  - int a=2; double b=10.3;
  - sizeof(a+b) ⇒ sizeof(double)

Typecasting

- Remember the **problem** with division
  - int a = 10, b = 20, c = 30, d ; float f;
  - f = c/b; f ⇒ ?
- The solution is to do the following
  - Convert at least one of the operand to floating point
  - f = c; f ⇒ 30.0
  - f /= b; or f = f/b; f ⇒ 1.50
- The shorthand of doing this is called typecasting
  - f = ((float)c)/b; f ⇒1.50
  - The type of c has not changed but the evaluated value of (float)c is now a float type

Typecasting (contd.)

- Not everything can be typecast to everything
  - Casting a float to an integer will lose information since int cannot store the fractional part
  - Similarly **int** should not be typecast to **char**
- General rule
  - Make sure the final type can store any value of the initial type

Statements in a C Program

- Parts of C program that tell the computer what to do
- Types of statements
  - Declaration statements – Declares variables etc.
  - Assignment statement – Assignment expression, followed by a **;**
  - Control statements – For branching and looping
    - Branching - if-else, switch
    - Looping - for, while, do while
  - Input/Output – Read/print, like printf/scanf

Statements (contd.)

- Compound statements
  - A sequence of statements enclosed within **{** and **}**
  - Each statement can be an assignment statement, control statement, input/output statement, or another compound statement
  - We will also call it block of statements sometimes informally

In The Next Class…

- You will learn about branching
- You will learn about looping
- You will learn about functions
- You will learn about pointers