Introduction to Computing

MCS1101B

Lecture 3-4

By
Soumadip Biswas
Associate Professor, IEM

Recap

- Expressions
  - Arithmetic
  - Assignment
  - Logical
- Special operators
  - SizeOf
  - AddressOf (&)

- Typecasting
- Statements
  - Declaration
  - Assignment
  - Control
    - Branching
    - Looping
  - Input /Output

Conditional Statements

- Allow different sets of instructions to be executed depending on truth or falsity of a logical condition

aka. Branching

How do we specify conditions?

- Using expressions
  - non-zero value means condition is true
  - value 0 means condition is false
- Usually logical expressions, but can be any expression
  - The value of the expression will be used

The **if** Statement

```
if (expression)
    statement;
```

*The condition to be tested is any **expression** enclosed in parentheses. The **expression** is evaluated, and if its value is **non-zero**, the statement is executed.*

```
if (expression)
    compound statement;
```

```
if (expression)
{
    statement 1;
    …
    statement n;
}
```

if else Statement

```
if (expression)
        statement / compound statement;
else
        statement / compound statement;
```

Example:
Grade Computation
Find the larger of two numbers
Find the largest of three numbers

```
if (expression)
        statement / compound statement;
else if (expression)
        statement / compound statement;
else
        statement / compound statement;
```

Nested if else

- It is not necessary for all if statements to have an else part
- Every else gets matched to the closest preceding unmatched if statement
- It’s very easy to create *confusion* while writing a nested if-else
- So it is always a good idea to use parentheses to avoid any ambiguity

Ambiguous statement

```
if (expression)
if (expression)
        statement;
else
        statement;
```

The **conditional** operator **?:**

Another way of writing if else statement

- <condition>?<expression1>:<expression2>;
    - If condition is true then expression1 is executed
    - If condition is false then expression2 is executed

…used for convenience

```
int x = 10, y = 20, max;
if (x > y)
        max = x;
else
        max = y;
```

Using conditional operator…

```
max = (x > y) ? x : y;
```

The **switch** statement

- This statement can be used instead of writing lot of if else statement
- You can provide statements different cases
- switch statement with match the provided value with the case number and execute statements from that point onwards
- All statements below a matched case is executed

```
switch (<expression>)
{
    case <const-expr> : <statements>
    case <const-expr> : <statements>
    …
    case <const-expr> : <statements>
    default :  <statements>
}
```

Example: Evaluation of expressions

The **break** statement

- The break statement takes the sequence of execution out of the block
  - Works with looping as well
  -
- switch-case does not work exactly like a if else if else if…
- We use break statements to mimic the behaviour

```
switch (<integer_value>)
{
      case <integer> : <statements>
                              break;
      case <integer> : <statements>
                              break;
      …
      case <integer> : <statements>
                              break;
      default :  <statements>
}
```

Looping Statements

- Group of statements that are executed repeatedly while some condition remains true
- Each execution of the group of statements is called an **iteration** of the loop

Example:

- Read 5 integers and display their sum
- Find the smallest number among 100 integers
- Grade computation for entire class

The **while** statement

- The condition to be tested is any expression enclosed in parentheses
- The expression is evaluated, and if its value is non-zero, the statement is executed
- Then the expression is evaluated again and the same thing repeats
- The loop terminates when the expression evaluates to 0

```
while (expression)
    statement;


while (expression)
    <Compound statement>
```

The **while** statement (contd.)

Examples

- Sum of the first N natural numbers
- Sum of the squares of the first N natural numbers
- Compute GCD of two numbers
- Calculate maximum of many positive numbers
- Compute the sum of digits of a number

The **for** statement

```
for ( expr1; expr2; expr3)
    statement;
for ( expr1; expr2; expr3)
    <Compound statement>
```

- **expr1** (init) : initialize parameter(s)
- **expr2** (test): test condition, loop continues if expression is non-0
- **expr3** (update): used to alter the value of the parameter(s) after each iteration
- **statement** (body): body of loop

The **for** statement (contd.)

Example:

Computing Factorial

Equivalence of **for** and **while** ⇒

Sum of N natural numbers

```
for ( expr1; expr2; expr3)
    statement;
```

```
expr1;
while (expr2)
{
    statement
    expr3;
 }
```

The **do while** statement

- Another way of doing looping
- Used for convenience
- Example
  - Decimal to binary conversion

```
do
statement;
while (expression);
```

```
do
{
Block of statements;
} while (expression);
```

Infinite loops and the break statement

- while (1)

  { statements }

- for (; ;)

  { statements }

- do

  { statements } while (1);

- Use break statement to come out of the loop body
  - can be used with while, do while, for, switch
  - does not work with if, else
- Causes immediate exit from a while, do/while, for or switch structure
- Program execution continues with the first statement after the structure

The **continue** statement

- Skips the remaining statements in the body of a while, for or do/while structure
  - Proceeds with the next iteration of the loop
- while and do/while loop
  - Loop-continuation test is evaluated immediately after the continue statement is executed
- for loop
  - expr3 is evaluated, then expr2 is evaluated

Nested Loops: Printing a 2-D Figure

- How would you print the following diagram?

```
* * * * * *
* * * * * *
* * * * * *
* * * * * *
```

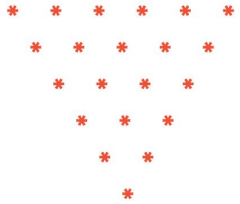
- Nested Loops
  - break and continue with nested loops

Half Pyramid

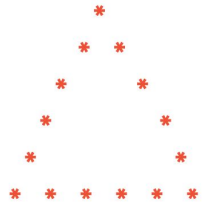
Inverted
Half Pyramid

Hollow Inverted
Half Pyramid

Full Pyramid



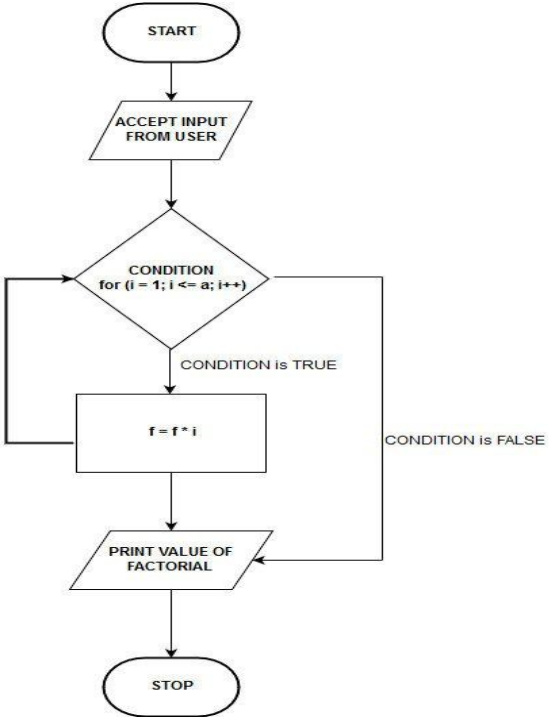
Inverted Full Pyramid



Hollow Full Pyramid

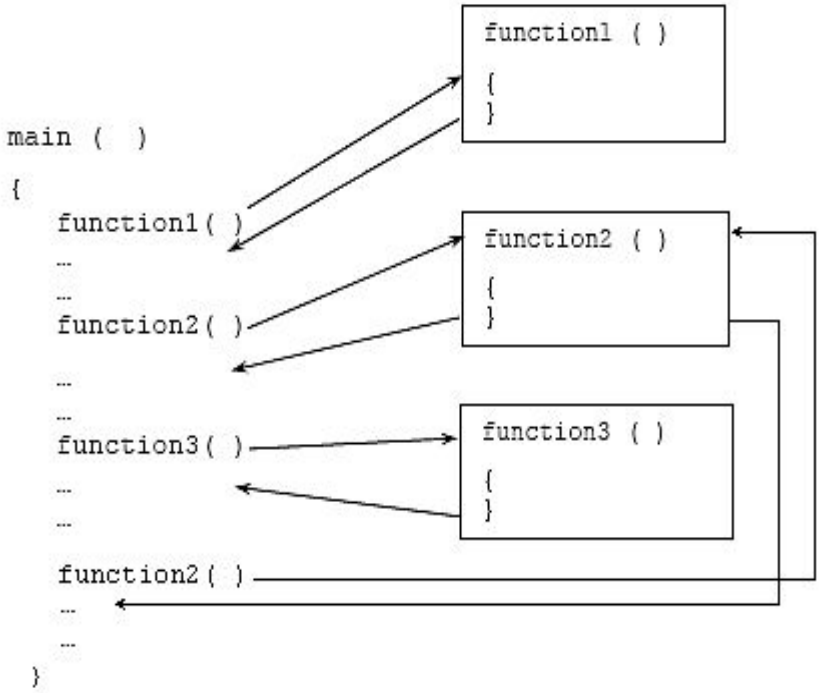
Sequence of Execution

- The flow of a program i.e. the steps and branches can be represented in graphically
- Represented using Flow chart
  - Example: a for loop ⇒
  - *Let's understand this on the board*



Functions

- A program segment that carries out some specific, well-defined task
- Example
  - A function to add two numbers
  - A function to find the largest of n numbers
- A function will carry out its intended task whenever it is **called** or **invoked**
  - Can be **called** multiple times



Functions (contd.)

- Examples
  - Print a banner
  - Factorial computation
  - Gcd computation

- A function definition has two parts:
  - The first line, called header
  - The body of the function
  - May or may not have a return value

```
return-value-type   function-name   ( parameter-list )
{
        declarations and statements
}
```

In The Next Class…

- You will learn about array and pointers
- You will learn more about functions