

Introduction to Computing

MCS1101B

Lecture 4

Recap

- Control Statements

- Branching
- Looping

- Branching

- if
- if else
- if else if else if ...
- ? :
- Nested if else
- switch

- Looping

- while
- for
- do while
- break, continue

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

Nested Loops: Printing a 2-D Figure

`printf ("*");` → 

`for (i=0; i<5;i++)`
`printf ("*");` → 

`for (j=0; j<5;j++)`
`for (i=0; i<5;i++)`
`printf ("*");` → 

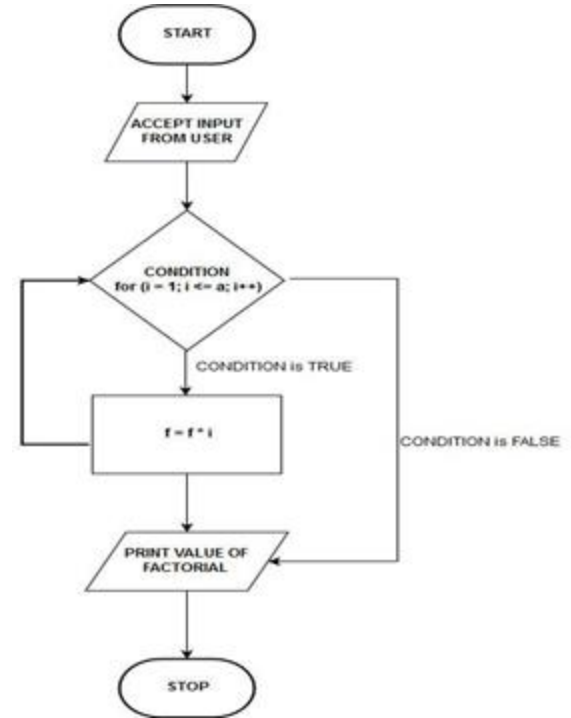
`for (j=0; j<5;j++)`
`for (i=0; i<j;i++)`
`printf ("*");` → ?

`for (j=0; j<5;j++)`
`for (i=0; i<=j;i++)`
`printf ("*");` → ?

`for (j=0; j<5;j++)`
`for (i=j; i<5;i++)`
`printf ("*");` → ?

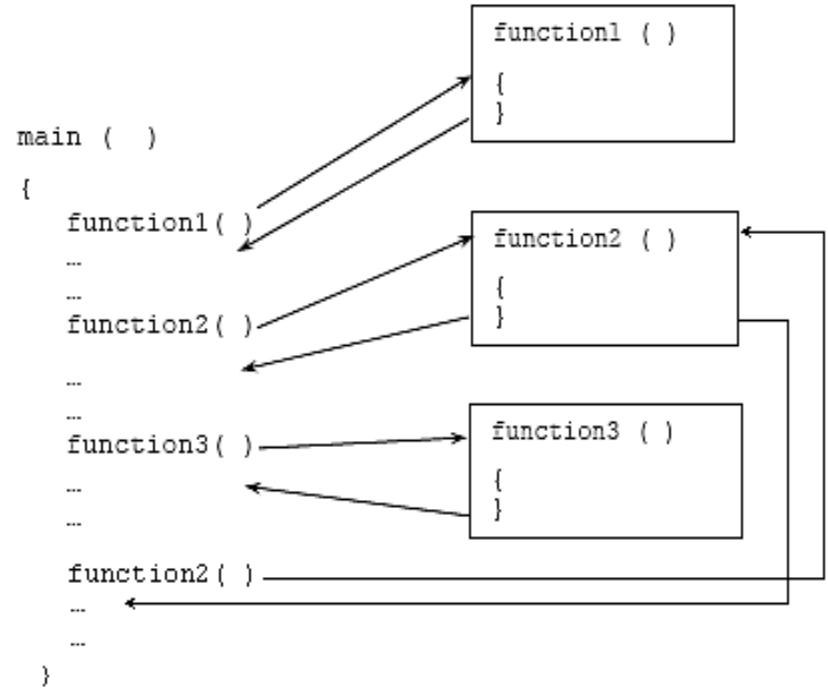
Sequence of Execution

- *The flow of a program*
 - the steps and branches can be represented in graphically
- Represented using Flow chart
 - Example: **a for loop** \Rightarrow



Functions

- *A program segment that carries out some specific, well-defined task*
- Examples:
 - 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**
 - A function can be **called** multiple times



Function Definition

- 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  
}
```

Function Prototypes

- `void print_msg ();`
- `int get_hour (void);`
- `void print_num (int);`
- `int increment (int x);`
- `int sum (int a, int b, int c);`
- `float add (float, float);`

```
double power (double, int);
```

```
int main ()  
{... printf ("%lf", power(2, 10)); ...}
```

```
double power (double base, int expo)  
{  
    int i; double result=1;  
    for(i=0; i<expo; i++)  
        result *= base;  
    return result;  
}
```


Function Prototypes (contd.)

- Compiler needs to know some details of a function(see list below) before it is being used (called) in a program
 1. **Name** of the function
 2. **Return type** of the function
 3. The **sequence of the parameters-types** (*parameter names are optional*) of that function
 4. The **definition/body** of the function **is optional**
- The collection of these minimum requirements is known as *function prototype*

Example

- **Function prototype**
- **Function Header**
- **Start** of function body
 - Local variables
 - A while loop
- **Start** of the loop block
 - Statement
 - Statement
 - Statement
- **End** of loop block
- Return statement
- **End** of function body

```
int gcd (int, int);  
int gcd (int A, int B)  
{  
    int temp;  
    while ((B % A) != 0)  
    {  
        temp = B % A;  
        B = A;  
        A = temp;  
    }  
    return (A);  
}
```

Functions (Two ways of writing)

```
#include<stdio.h>
void print_msg ()
{
    printf ("inside print_msg function\n");
}
int main ()
{
    printf ("inside main function\n");
    print_msg ();
    printf ("inside main function again\n");
    return 0;
}
```

```
#include<stdio.h>
void print_msg ();
int main ()
{
    printf ("inside main function\n");
    print_msg ();
    printf ("inside main function again\n");
    return 0;
}
void print_msg ()
{
    printf ("inside print_msg function\n");
}
```

For both the above styles
The output will be the same >>>

inside main function
inside print_msg function
inside main function again

Functions (Two more examples)

```
#include<stdio.h>
int get_result ()
{
    printf ("inside get_result\n");
    return 1000;
}
int main ()
{
    int result = get_result();
    printf ("value returned = %d\n", result);
    // printf ("value returned = %d\n", get_result());
    // you can also directly call here ~~~~~~
    return 0;
}
```

Output>>> inside get_result
 value returned = 1000

```
#include<stdio.h>
float add_num (float a, float b)
{
    float result = a + b;
    return result;
}
int main ()
{
    float x=100, y=200;
    printf ("sum of x and y = %f\n", add_num (x, y));
    return 0;
}
```

Output>>> sum of x and y = 300.0

Functions - Passing of variables

Passing of variables

- Variables values are copied when then are passed (by calling) to a function
- The actual variables are not passed
- So, a change made to a variable within a function will not reflect in the variable at the end of the caller

The return statement

- Return statement is optional
- But, the return type in the function prototype must be present
- Return statement causes the sequence of execution to return to the caller

Functions (Another example)

```
void swap (int a, int b)
{
    printf ("a=%d b=%d\n", a, b); //a=10 b=20

    int tmp = a; // copies 10 into tmp
    a = b;       // copies 20 into a
    b = tmp;     // copies 10 into b

    printf ("a=%d b=%d\n", a, b); //a=20 b=10
}
```

```
void swap (int, int);
int main ()
{
    int a=10, b=20;

    printf ("a=%d b=%d\n", a, b); //a=10 b=20
    swap (a, b);
    printf ("a=%d b=%d\n", a, b); //a=? b=?
    return 0;
}
```

Scope of Variables

- Part of the program from which the value of the variable can be used (seen)
- Scope of a variable - Within the block in which the variable is defined
 - Block = group of statements enclosed within { }
- Local variable – scope is usually the function in which it is defined
 - So two local variables of two functions can have the same name, but they are different variables
- Global variables – declared outside all functions (even main)
 - scope is entire program by default, but can be hidden in a block if local variable of same name defined

In The Next Class...

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