Programming With C

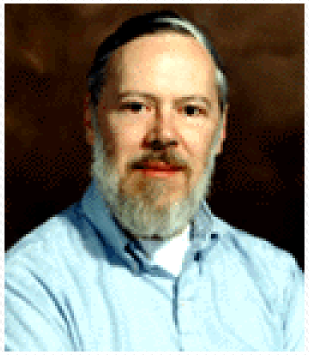
Presented by- Saubhik Goswami

M.C.A Department

M.S.I.T.

The C Language

- The C programming language was designed by Dennis Ritchie at Bell Laboratories in the early 1970s



- Influenced by:
  - ALGOL (1960),
  - BCPL (Martin Richard, 1967),
  - B (Ken Thompson, 1970)

- In a Line:
  - C is basically a structured, procedural, case-sensitive middle level programming language.

- There are mainly three types of language:

  i) low level language---machine language and assembly language

  ii) middle level language---C

  iii) High level language----Fortran, Cobol, Basic.

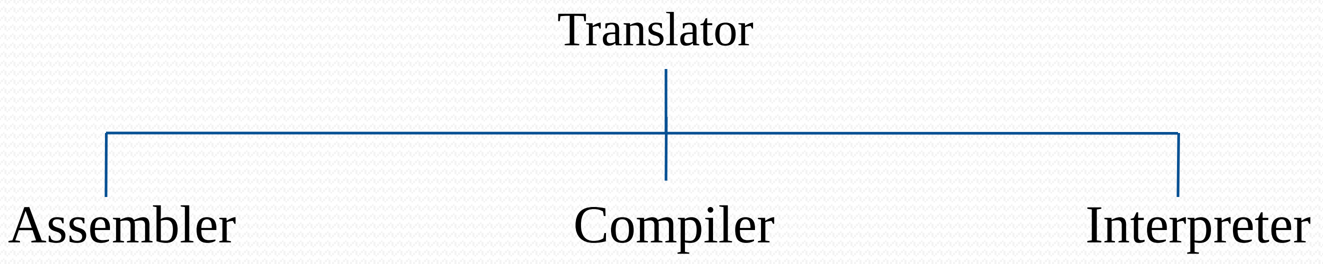
- Why C Programming language is called as Middle Level Language ?

  (i) it gives or behaves as High Level Language through *Functions* - gives a modular programming and breakup, increased efficiency for reusability

  (ii)it gives access to the low level memory through *Pointers*. Moreover it does support the Low Level programming i.e, Assembly Language.

  As its a combination of these two aspects, **its neither a High Level nor a Low level language** but a *Middle Level Language*.

Translators: There are different types of translators which convert the high level language into its equivalent machine language(low level language).



Structure of a C program:

```
#include<header file name>
  ...................................
  < return type > main( )
     {
       declarations
...................................
       body of the program
...................................
     }
```

HelloWorld.C

```
#include  <stdio.h>
#include<conio.h>
void main( )
   {
      clrscr( );
      printf("Hello, world!\n");
      getch( );
   }
```

It stands for standard input output header file, which provides function like printf(), scanf()…

A line starting with a # is a preprocessor directive, it inserts the specified file in place of that line.

HelloWorld.C

```
#include <stdio.h>
#include<conio.h>
void main( )

{

   clrscr( );

   printf("Hello, world! \n");

   getch( );

}
```

Program mostly a collection of functions
"main" function special: It is a predefined function without which no c program would run. It is the place where the execution of every program starts.
"void" qualifier indicates function does not return anything

clrscr( ) and getch( ) are predefined functions stored in the conio.h header file

I/O performed by a library function: not included in the language

Semicolon is the statement terminator.

It is a new line character. Generally it is used in the printf() statement to format the output.

Square.C

```
#include<stdio.h>
#include<conio.h>
void main( )
    {
        int x, ans;

        clrscr( );

        printf("Enter a Number:");

        scanf("%d", &x);

        ans = x *  x;

        printf("The square of %d is %d",x, ans);

        getch( );
        }
```

This is the declaration section it tells the compiler that x is a variable and supposed to deal with integer data. Here int stands for integer data type.

- printf displays information on screen.
- printf displays the text you put inside the double quotes.
- printf can display variables by using the % conversion character.
- printf returns the number of characters printed.

- The scanf function: read information from a standard input device (keyboard).
- The conversion specifier argument tells scanf how to convert the incoming data.
- scanf returns the number of successful inputs

<u>%d</u> is a conversion specification and is used in both functions printf() and scanf(). It means that a number integer type to be converted into decimal number before printing out or a decimal number is coming at the time of reading from the keyboard.
<u>&x:</u> It is the address operator and extracts the address of the memory location which is assigned to a and b.

What would be the output of the following programs?

```
#include<stdio.h>
#include<conio.h>
  void main( )
    {
      int x;
      clrscr( );
      x=printf("hello");
      printf("\n%d",x);
      getch( );
    }
```

Output: 5

```
#include<stdio.h>
 #include<conio.h>
      void main()
          {
            int x, int y;
            clrscr( );
            printf("enter the value of  x:");
            y=scanf("%d",&x);
            printf("\n%d",y);
            getch( );
          }
```

Output: 1

What is the difference between declaration and definition?

- **Declaring a variable** means describing its type to the compiler but not allocating any space for it.

- **Defining a variable** means declaring it and also allocating space to hold the variable.

- Example: extern int y;----- declaration

  int y = 3;   ----- definition

What is the difference between #include <stdio.h> and #include “stdio.h"

- **#include “stdio.h":** This command looks for the file stdio.h in the current directory as well as in the specified list of directories as mentioned in the include search path .

- **#include <stdio.h>:** This command looks for the file stdio.h in the specified list of directories only.

What is a Macro?

- A macro is a preprocessor directive that provides a mechanism for token replacement in our source code. Macros are created by using the #define statement.

Program without #define:

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
    float r, area;
    clrscr( );
    printf("Enter the radius of the circle:");
    scanf("%f",&r);
    area= 3.14*r*r;
    printf("Area of the circle=%f",area);
    getch( );
  }
```

Program with #define:

```
#include<stdio.h>
#include<conio.h>
#define PI 3.1415
void main( )
  {
    float r, area;
    clrscr( );
    printf("Enter the radius of circle:");
    scanf("%f",&r);
    area= PI*r*r;
    printf("Area of the circle=%f",area);
    getch( );
  }
```

Macro Template

Macro Expansion

How a C program compile

$$\begin{tikzcd}
\text{C source code (prg.C)} \arrow[d, "\text{Preprocessor}"] & & \\
\text{Expanded Source code (prg.I)} \arrow[d, "\text{Compiler}"] & & \\
\text{Assembly code (prg.ASM)} \arrow[d, "\text{Assembler}"] & & \\
\text{Relocatable Object Code (prg.obj)} \arrow[dr] & & \text{Obj code of Library fn} \arrow[dl, "\text{Linker}"] \\
& \text{Executable code(prg.EXE)} &
\end{tikzcd}$$

Data Types

- In C language , data types are broadly classified as:

1. Primitive data types

	Byte	Format
void	0	0
int	2	%d
char	1	%c
float	4	%f
double	8	%lf
long double	10	%Lf

2. Derived data types: i) array, ii) pointer, iii) function

3. User-defined data types: i) structure, ii) union, iii) enumeration

Example of different data types with their conversion specifications:

```
void main( )
     {
         int code;
         char grade;
         float price;
         double size;
         clrscr( );
         printf(“\n Enter the code no of the item:”);
         scanf(“%d”, &code);
         printf(“\n Enter the grade of the item:”);
         scanf(“%c”, &grade);
         printf(“\n Enter the cost of the item:”);
        scanf(“%f”, &price);
```

```
 printf(“\n Enter the length of the item:”);
 scanf(“%lf”, &size);
 printf(“\n\n The description of the item as follows:”);
 printf(“\n Item code=  %d”, code);
 printf(“\n Item grade= %c”, grade);
 printf(“\n Item cost=  %f”, cost);
 printf(“\n Item size=  %lf”, size);
 getch( );
}
```

What would be the output of the following programs?

```
#include<stdio.h>
#include<conio.h>
void main( )
 {
   char ch;
   int a=10;
   ch = ‘A’;
   clrscr( );
   printf(“Integer value is: %d”, a);
   printf(“\n The ASCII value of %c is %d”, ch, ch);
   getch( );
 }
```

Output:
Integer value is: 10
The ASCII value of A is 65

```
#include<stdio.h>
#include<conio.h>
void main()
      {
        int x = 66;
        clrscr( );
        printf(“ %c“, x);
        getch( );
       }
```

Output: B

Operators

Operators in C are classified into two categories:

- <u>Classification Based on Number of Operands:</u>

1. Unary Operators:

    x +

    - x

    + + x

2. Binary Operators:

    x + y

    x < y

    x % y

3. Ternary Operator:

    exp1 ? exp2 : exp3

Consider the following statement:

```
a = 10;
b = 15;
x = (a > b) ? a : b;
```

It's equivalent statement:

```
if ( a > b)
    x = a;
else
    x = b;
```

Classification Based on Role of Operator:

1. Arithmetic operators:

Unary Arithmetic Operator:

- ++x(pre-increment)

  -- Increment ‘x’ before use
- x++(post-increment)

  -- Increment ‘x’ after use
- --x(pre-decrement)

  -- Decrement ‘x’ before use
- x--(post-decrement)

  -- Decrement ‘x’ after use

Example:

```
n = 10;
x = n++;  /* x becomes 10, n  11 */
```

```
x = n;
n = n+1;
```

```
x = ++n; /* x becomes 12, n  12 */
```

```
n = n +1;
x = n;
```

```
x = n--; /* x becomes 12, n  11 */
```

```
x = n;
n = n -1;
```

```
x = --n; /* x becomes 10, n  10 */
```

```
n = n-1;
x = n ;
```

Increment & Decrement Operator Cont'd:

Find the output:

```
void main()
{
int x=4,y,z;
y=--x;
z=x--;
printf("x=%d,y=%d,z=%d",x,y,z);
}
```

Output: x=2
y=3
z=3

Find the output:

```
void main()
{
int i=5;
printf("%d%d%d%d%d",i++,i--,++i,--i,i);
}
```

Answer: 4 5 5 4 5

Explanation:

- The arguments in a function call are pushed into the stack from left to right. The evaluation is by popping out from the stack. and the evaluation is from right to left, hence the result.

Increment & Decrement Operator Cont'd:

Find the output:

```
 void main()
  {
     int i=5,j=6;
     printf("%d",i+++j);
    }
```

Answer: 11

Explanation:

- the expression i+++j is treated as

  (i++ + j)

Find the output:

```
 void main()
 {
         int i=5;
         printf("%d",i+++++i);
 }
```

Answer: Compiler Error

Explanation:

- The expression i+++++i is parsed as **i ++ ++ + i** which is an illegal combination of operators.

Binary Arithmetic Operator:

x + y, x – y, x * y, x / y, x % y

Ex:
```
int a = 2, b = 4;
a = ((a/b) * b) + (a % b);
printf("%d", a);
```

Output: 2

2. Assignment Operator:

```
a = 5;
c = a + b;
```

Complex Assignment Operator:

+ =, - =, * =, / =

```
x + = y;
    x = x + y;
a * = b;
    a = a * b;
```

3. Relational Operator:

x < y, x <= y, x > y, x >= y, x != y

Ex:
```
#include<stdio.h>
   void main()
      {
        int a;
        a = 2 < 3 != 2;
        printf(" The value of a= %d", a);
      }
```

Output: 1

4. Logical Operator:

1. Logical AND (&&)
2. Logical OR (||)
3. Logical NOT (!)

Logical Operator Cont'd

- **Short Circuit Operator:**

Logical AND(**&&**) and logical OR(||) are called Short Circuit Operator.

- Here (op-1 && op-2) is true if both op-1 and op-2 are true and false otherwise.

  Consider an example:

  (a<x && x<b) ? 1:0;

- (op-1 || op-2) is false if both op-1 and op-2 are false and true otherwise.

  Consider an example:

  (a+b) > c || a!= 0 || a == b;

What would be the output of the following program:

```
void main()
  {
     int  x=0, y=0, z;
       for(z=0;z<5;z++)
          {
               if((++x>2) || (++y>2))
                 {
                       x++;
                 }
          }
       printf("x=%d,y=%d",x,y);
}
```

Output: x = 8

y = 2

What would be the output of the following program:

```
void main()
 {
    int  x=0, y=0, z;
     for(z=0;z<5;z++)
        {
           if((++x>2) && (++y>2))
            {
                x++;
            }
        }
     printf("x=%d,y=%d",x,y);


 }
```

Output: x= 6

y = 3

- Ternary Operator:

```
        exp1 ? exp2 : exp3
```

Consider the following statement:

```
        a = 10;
        b = 15;
        x = (a > b) ? a : b;
```

It’s equivalent statement:

```
        if ( a > b)
            x = a;
        else
          x = b;
```

Find the output:

```
void main( )
{
  int x, y=30;
  x=(y >5 ? (y <= 10 ? 100:200):500);
  printf(“%d”, x);
}
```

Output: 500

Control Flow

Branching

- if
- if-else
- nested if
- else if
- switch
- break
- goto
- continue

Looping

- for
- while
- do-while

Single statement if block

```
if(condition)
   do this;
```

Multi statement if block

```
if(condition)
  {
    do this;
    and this;
    ……
   }
```

Single statement if-else block

```
if(condition)
   do this;
else
  do this;
```

Multi statement if-else block

```
if(condition)
   {
     do this;
     and this;
     ……
    }
else
     {
       do this;
       and this;
      }
```

Switch Statement

- Switch(expression)

```
          {
            case value1:  ……....
                          ……..
                          break;
             case value2:  ……..
                           ……..
                           break;
                    .
                    .
                    .
           }
```

- What is the difference between IF and SWITCH statement?

  In IF statement we can use conditional operator (like <, >, != etc) but in SWITCH we can’t use any conditional operator. Here we can only use the (=) operator.

- Find the Output:

```
void main()
     {
       int i=2;
       switch(i)
      {
       printf("hello");
       case 1:  printf("hi");
                break;
       case 2: printf (“good”);
               break;
        }
     }
```

Output: good

What would be the output of the following program:

```
void main()
   {
       int i=4;
       switch(i)
          {
             default:  printf("hello");
             case 1: printf(“\nhi");
                     break;
             case 2: printf(“\ngood");
                     break;
             case 3: printf(“bad");
           }
  }
```

Output:

```
hello
hi
```

Loop

```
while ( test-conditions)
        {
          body of code..
          ………………
        }

    do
      {
        body of code..
        ………………
      } while ( test-conditions)


for ( initialization; test-condition; increment)
        {
          body of code..
          ………………
        }
```

- What is the difference between FOR loop and WHILE or DO-WHILE loop?

In while and do-while loops, continue statement causes the control to go directly to the test-condition and then to continue the iteration process. But in the case of for loop, the increment section of the loop is executed before the test condition is evaluated.

Difference between FOR loop and WHILE loop Cont’d

Consider the example:

```
void main()
{
 int i=1;
 for(; ;)
  {
    printf(“%d”, i++);
    if(i>5)
         break;
  }
}
```

Output: 1 2 3 4 5

Consider the example:

```
void main()
{
 int i=1;
 while( )
  {
    printf(“%d”, i++);
    if(i>5)
         break;
  }
}
```

Output: the condition in the while loop is must.

Break and continue statement

Break:

```
int i;
for(i=1; i <= 10; i++)
    {
     if(i == 5)
     break;
     printf("%d", i);
    }
```

Output: 1, 2, 3, 4

Continue:

```
int i, j;
 for(i=1; i <= 2; i++)
     {
      for(j=1; j <= 2; j++)
         {
          if(i == j)
          continue;
          printf("%d %d", i, j);
         }
     }
```

Output: 1, 2, 2, 1