Problem Solving Through programming in C
Course Code: ONL1001

Basic structure of C program

Ms. Shubhra dwivedi
Assistant Professor
School - SCOPE
VIT-AP Amaravati
Shubhra.d@vitap.ac.in

Structure of a C program

```
#include <stdio.h>

void main ( ){

        printf("\nHello World\n");
}
```

Preprocessor directive (header file)

Program statement

```
#include <stdio.h>
#define VALUE 10

int global_var;

void main ( ){

        /* This is the beginning of the program */

        int local_var;
        local_var = 5;
        global_var = local_var + VALUE;

        printf ("Total sum is: %d\n", global_var); // Print out the result
}
```

Preprocessor directive

Global variable declaration

Comments

Local variable declaration

Variable definition

Preprocessor Directives

- The first statement to be checked by the compiler
- Preprocessor Directives always preceded with '#' sign
- They contain information to the compiler which are required by the compiler during compilation.
- There are a few compiler directives. But only 2 of them will be discussed here.
  - #include <stdio.h>
    - Tells the compiler to include the file stdio.h during compilation
    - Anything in the header file will be included a part of the program
  - #define VALUE 10
    - Tells the compiler to substitute the word VALUE with 10 during compilation

Preprocessor Directives

```
#define PI 3.141592654

main() {
      …..

      perimeter = 2*PI*radius;

      area = PI*radius*radius;

      ……

}
```

The result of the compilation is the same for both C program (One with #define and the other without it).

Which one is preferred (less typing)?

Which one is more readable?

The one with constant definition using #define preprocessor directive.

Before compilation, the pre-processor will replace all PI with 3.141592654.

```
      main() {
            …..

            perimeter = 2* 3.141592654 *radius;

            area = 3.141592654 *radius*radius;

            ……

      }
```

Comments

- Comment means explanations or annotations that are included in a program for documentation and clarification purpose.
- Comments are completely ignored by the compiler during compilation and have no effect on program execution.
- Comments starts with ‘/*’ and ends with ‘*/’
- Some compiler support comments starting with ‘//’

Basic Data Types

- There are 3 Basic data types in C:
  - int (used to declare numeric program variables of integer type)
  - char (used to declare character variable)
  - double (used to declare floating point variable)
- In addition, there are float, void, short, long, etc.
- Variables are declared before they are used in a program. Declaration specifies the type of a variable.
  - Example: int local_var;
- Once defined variables are used for storing a value.

Variable

- A variable can be declared globally or locally.
- A globally declared variable can be accessed from any part of the program.
- A locally declared variable can only be accessed from inside the function in which the variable is declared.

Statements

- A specification of an action to be taken by the computer as the program executes is called a Statement.
- In the previous example, there are 2 lines following variable declaration that terminate with semicolon ';' are statements:

```
global_var = local_var + VALUE;
printf ("Total sum is: %d\n", global_var);
```

- Each line that end with a semicolon is a statement.

Basic Functions

- A C program consists of one or more functions that contain a group of statements which perform a specific task.
- A C program must at least have one function: the function **main**.
- We can create our own function or use the functions that has been declared in C library (called Predefined function).
- In order to use Predefined functions, we have to include the appropriate header file (example: stdio.h).

- In this section, we will learn a few functions that are pre-defined in the header file **stdio.h**
- These functions are:
  - printf()
  - scanf()
  - getchar() & putchar()
- In addition to those functions, we will also learn about **Format Specifier** and **Escape Sequence** which are used with printf() and scanf().

printf()

- Used to send data to the standard output (usually the monitor) to be printed according to specific format.
- **General format:**
  - printf(“control string”, variables);
- Control string is a combination of text, format specifier and escape sequence.
- **Example:**
  - printf(“Thank you”);
  - printf (“Total sum is: %d\n”, global_var);
    - %d is a format Specifier
    - \n is an escape sequence

Format Specifier

No	Format Specifier	Output Type	Output Example
1	%d	Signed integer	76
2	%i	Unsigned integer	76
3	%o	Octal representation	134
4	%u	Unsigned decimal integer	76
5	%x	Hexadecimal representation(small letter)	9c
6	%X	Hexadecimal representation (capital letter)	9C
7	%f	Float values	76.0000
8	%e	Scientific notation of floats (using e)	7.6000e+01
9	%E	Scientific notation of floats (using E)	7.6000E+01
10	%g	The shorter between %f and %e	76
11	%G	The shorter between %f and %E	76
12	%c	Character	‘A’
13	%s	String	‘Hi'

The format specifier is used during input and output. It is a way to tell the compiler what type of data is in a variable during taking input using scanf() or printing using printf(). Some examples are %l ,%lf, %Lf for long, double and long double etc.

Escape Sequence

Escape Sequence	Effect
\a	Beep sound or alarm
\b	Backspace
\f	Formfeed (for printing)
\n	New line
\r	Carriage return
\t	HorizontalTab
\v	Vertical tab
\\	Backslash
\”	“ sign
\o	Octal decimal
\x	Hexadecimal
\O	NULL

Escape sequence is used in the printf() function to do something to the output.

scanf()

- Reads data from the standard input device (usually keyboard) and store it in a variable. The General format is:
  - scanf("Control string", &variable);
- The general format is pretty much the same as printf() except that it passes the address of the variable (notice the & sign) instead of the variable itself to the second function argument.
- Example:

```
int age;
printf("Enter your age: ");
scanf("%d", &age);
```

getchar() and putchar()

- getchar() - reads a character from standard input
- putchar() - writes a character to standard output
- **Example:**

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

    char my_char;
    printf("Please type a character: ");
    my_char = getchar();
    printf("\nYou have typed this character: ");
    putchar(my_char);
}
```

The End

Thank U