STUDY C LANGUAGE

WITH

ARAFAT BIN REZA

INTRODUCTION OF C LANGUAGE

- What is C Language?
- C is a general-purpose, procedural, imperative computer programming language.
- C is the most widely used computer language.
- If you are new to programming, C is a good choice to start your programming journey.

INTRODUCTION OF C LANGUAGE

- C has now become a widely used professional language for various reasons −
- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities
- It can be compiled on a variety of computer platforms

INTRODUCTION OF C LANGUAGE

- Different parts of C program.

- Pre-processor
- Header file
- Function
- Variables
- expression
- Comment

MY FIRST PROGRAM IN C LANGUAGE .


```
#include <stdio.h>

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

   return 0;
}
```

MY FIRST PROGRAM IN C LANGUAGE .

1. Pre-processor :

#include, the first word of any C program. It is also known as pre-processor. The main work of pre-processor is to initialize the environment of program, i.e to link the program with the header file <stdio.h>.

2. Header file:

Header file is a collection of built-in functions that help us in our program. Header files contain definitions of functions and variables which can be incorporated into any C program by pre-processor #include statement. Standard header files are provided with each compiler, and cover a range of areas like string handling, mathematical functions, data conversion, printing and reading of variables.

To use any of the standard functions, the appropriate header file must be included. This is done at the beginning of the C source file.

For example, to use the printf() function in a program, the line #include <stdio.h> is responsible.

MY FIRST PROGRAM IN C LANGUAGE .

3. main() function:

main() function is a function that must be used in every C program. A function is a sequence of statement required to perform a specific task. main() function starts the execution of C program. In the above example, int in front of main() function is the return type of main() function. we will discuss about it in detail later. The curly braces { } just after the main() function encloses the body of main() function.

4. Compile and Run:

There are many different ways to compile and run a C program. All that is required is a C compiler.

BASIC SYNTAX

- Tokens in C
- the following C statement consists of five tokens −
- printf("Hello, World! \n");

1. Bracket- a.() b.{} c.[] d.<>
2. Semicolons- The semicolon is a statement terminator. each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

Given below are two different statements −

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

return 0;

BASIC SYNTAX

3. Comments- Comments are like helping text in your C program and they are ignored by the compiler.

There are three types of Comments , they are :

a. /* */ b. // c.///

4. Identifiers- A C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores, and digits.

C does not allow punctuation characters such as @, $, and % within identifiers. C is a case-sensitive programming language. Thus, Manpower and manpower are two different identifiers in C.

Here are some examples of acceptable identifiers −

mohd zara abc move_name a_123

myname50 _temp j a23b9 retVal

BASIC SYNTAX

5. Keywords- The following list shows the reserved words in C. These reserved words may not be used as constants or variables or any other identifier names.

auto	else	long	switch	break
enum	register	typedef	case	extern
return	union	char	float	short
unsigned	const	for	signed	void
continue	goto	sizeof	volatile	default
if	static	while	do	int
struct	Packed	double		

BASIC SYNTAX

6. Whitespace - A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it.

Whitespace is the term used in C to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as int, ends and the next element begins. Therefore, in the following statement −

int age;

DATA TYPES

1. Char
2. unsigned char
3. signed char
4. Int
5. unsigned int
6. Short
7. unsigned short
8. Long
9. unsigned long
10. float
11. double
12. long double

VARIABLES

- Type Description
- Char-Typically a single octet(one byte). This is an integer type.
- Int-The most natural size of integer for the machine.
- Float-A single-precision floating point value.
- Double-A double-precision floating point value.
- Void-Represents the absence of type.

VARIABLE DEFINITION

1. type variable_list;
2. type variable_name = value;

1.EXAMPLE:

- int i, j, k;
- char c, ch;
- float f, salary;
- double d;

2. EXAMPLE:

```
extern int d = 3, f = 5;    // declaration of d and f.
int d = 3, f = 5;           // definition and initializing d and f.
byte z = 22;                // definition and initializes z.
char x = 'x';               // the variable x has the value 'x'.
```

VARIABLE DECLARATION

- Example:

```
// Variable declaration:
extern int a, b;
extern int c;
extern float f;
```

EXAMPLE:


```
#include <stdio.h>

// Variable declaration:
extern int a, b;
extern int c;
extern float f;

int main () {

   /* variable definition: */
   int a, b;
   int c;
   float f;

   /* actual initialization */
   a = 10;
   b = 20;

   c = a + b;
   printf("value of c : %d \n", c);

   f = 70.0/3.0;
   printf("value of f : %f \n", f);

   return 0;
}
```

FUNCTION


```
// function declaration
int func();

int main() {

  // function call
  int i = func();
}

// function definition
int func() {
  return 0;
}
```

CONSTANTS

- Defining Constants
- There are two simple ways in C to define constants −
- Using #define preprocessor.
- Using const keyword.
- The #define Preprocessor:
- Given below is the form to use #define preprocessor to define a constant −

#define identifier value

EXAMPLE:


```
#include <stdio.h>

#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'

int main()
{

   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

THE CONST KEYWORD

- You can use const prefix to declare constants with a specific type as follows −
- #include <stdio.h>

- int main() {

- const int LENGTH = 10;
- const int WIDTH = 5;
- const char NEWLINE = '\n';
- int area;
-
- area = LENGTH * WIDTH;
- printf("value of area : %d", area);
- printf("%c", NEWLINE);

- return 0;
- }

STORAGE CLASSES

- A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. They precede the type that they modify.
- We have four different storage classes in a C program −

- auto
- register
- static
- extern

THE AUTO STORAGE CLASS

- The auto storage class is the default storage class for all local variables.

- {
- int mount;
- auto int month;
- }
- The example above defines two variables with in the same storage class. 'auto' can only be used within functions, i.e., local variables.

THE REGISTER STORAGE CLASS

- The register storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

- {
- register int miles;
- }
- The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

THE STATIC STORAGE CLASS

- The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

- The static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.

- In C programming, when static is used on a global variable, it causes only one copy of that member to be shared by all the objects of its class.

THE EXTERN STORAGE CLASS

- The extern storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern', the variable cannot be initialized however, it points the variable name at a storage location that has been previously defined.

- When you have multiple files and you define a global variable or function, which will also be used in other files, then extern will be used in another file to provide the reference of defined variable or function. Just for understanding, extern is used to declare a global variable or function in another file.

OPERATORS

- Arithmetic Operators(+,−,*,/,% ,++,--)
- Relational Operators(==,!= ,>,<,>=,<=)
- Logical Operators(&& ,||,!)
- Bitwise Operators(&, |, ^, ~,<<,>>)
- Assignment Operators(=,+=,-=,*=,/=,%=,<<=,>>=,&=,^=,|=)
- Misc Operators(sizeof(),&,*,? :)

DECISION MAKING

- Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.