CHAPTER 3

3. PROGRAM STRUCTURE

STRUCTURE OF A C PROGRAM

The **C programming language** was designed by Dennis Ritchie as a systems programming language for Unix.

A C program basically has the following structure:

- Preprocessor Commands
- Functions
- Variable declarations
- Statements & Expressions
- Comments

Example:

```
#include <stdio.h>

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

   return 0;
}
```


Preprocessor Commands

These commands tell the compiler to do **preprocessing before doing actual compilation**. Like *#include <stdio.h>* is a preprocessor command which tells a C compiler to include **stdio.h** file before going to actual compilation. The standard input and output header file (stdio.h) allows the program to interact with the screen, keyboard and file system of the computer.

```
# include <stdio.h>  }
# include <math.h>   } header files
# include <stdlib.h> }
```

NB/ Preprocessor directives are not actually part of the C language, but rather instructions from you to the compiler.

Functions

These are main building blocks of any C Program. Every C Program will have one or more functions and there is one mandatory function which is called *main()* function. When this function is prefixed with keyword *int*, it means this function returns an integer value when it exits. This integer value is retuned using *return* statement.

The C Programming language provides a set of built-in functions. In the above example printf() is a C built-in function which is used to print anything on the screen.

A function is a group of statements that together perform a task. A C program can be divide up into separate functions but logically the division usually is so each function performs a specific task. A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

The general form of a function definition in C programming language is as follows:

return_type function_name(parameter list)
{
body of the function/Function definition
}

Variable Declarations

In C, all variables must be declared before they are used. Thus, C is a **strongly typed** programming language. Variable declaration ensures that appropriate memory space is reserved for the variables. Variables are used to hold numbers, strings and complex data for manipulation e.g.
Int x;
Int num; int z;

Statements & Expressions

Expressions combine variables and constants to create new values e.g.
x + y;

Statements in C are **expressions, assignments, function calls, or control flow** statements which make up C programs.
An assignment statement uses the assignment operator "=" to give a **variable** on the operator's left side the **value** to the operator's right or the result of an expression on the right.
z = x + y;

Comments

These are **non-executable program statements** meant to **enhance program readability** and allow **easier program maintenance**- they **document the program**. They are **ignored by the compiler**. These are used to give additional useful information inside a C Program. All the comments will be put inside /*...*/ or // for single line comments as given in the example above. A comment can span through multiple lines.

```
/* Author: Mzee Moja */
```

or

```
/*
 * Author: Mzee Moja
 * Purpose: To show a comment that spans multiple lines.
 * Language:  C
 */
```

or
Fruit = apples + oranges; // get the total fruit

Escape Sequences

Escape sequences (also called back slash codes) are character combinations that begin with a backslash symbol used to format output and represent difficult-to-type characters.

They include:

\a	Alert/bell
\b	Backspace
\n	New line
\v	Vertical tab
\t	Horizontal tab
\\	Back slash
\'	Single quote

\” Double quote
\0 Null

Note the following

- C is a **case sensitive** programming language. It means in C *printf* and *Printf* will have different meanings.
- End of each C statement must be marked with a semicolon.
- Multiple statements can be on the same line.
- Any combination of spaces, tabs or newlines is called a **white space**. C is a free-form language as the C compiler chooses to ignore whitespaces. Whitespaces are allowed in any format to improve readability of the code. **Whitespace is the term used in C to describe blanks, tabs, newline characters and comments.**
- Statements can continue over multiple lines.
- 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 or a to z or an underscore _ followed by zero or more letters, underscores, and digits (0 to 9). C does not allow punctuation characters such as @, $, and % within identifiers.
- **A keyword is a reserved word** in C. Reserved words may not be used as constants or variables or any other identifier names

SAMPLE PROGRAM

```
//First program
#include<stdio.h>
    main()
    {
        int num; // Declaration

        num =1; // Assignment statement
        printf(" My favorite number is %d because", num);
        printf(" it is first.\n");
        return 0;
    }
```


The program will output (print on screen) the statement “**My favorite number is 1 because it is first**”.
The %d instructs the computer where and in what form to print the value. %d is a **type specifier** used to specify the output format for integer numbers.

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			

SOURCE CODE FILES

When you write a program in C language, **your instructions** form the source code/file. C files have an extension **.c**. The part of the name before the period is called the **extension**.

Object Code, Executable Code and Libraries

An **executable file is a file containing ready to run machine code**. C accomplishes this in two steps.

- ✓ Compiling –The compiler converts the source code to produce the intermediate **object code**.
- ✓ The linker combines the intermediate code with other code to produce the executable file. You can compile individual modules and then combine modules later.

Linking is the process where the object code, the start up code and the code for library routines used in the program (all in machine language) are combined into a single file- the executable file.

- ➢ **NB/** An **interpreter** unlike a **compiler** is a computer program that directly executes, i.e. performs, instructions written in a programming, without previously compiling them into a machine language program.
- ➢ If the compiled program can run on a computer whose CPU or operating system is different from the one on which the compiler runs, the compiler is known as a **cross-compiler**.
- ➢ A program that translates from a low level language to a higher level one is a **decompiler**.
- ➢ A program that translates between high-level languages is usually called a **source-to-source compiler** or **transpiler**.

Library Functions

There is a minimal set of library functions that should be supplied by all C compilers, which your program may use. This collection of functions is called the C standard library. The standard library contains functions to perform disk I/O (input/ output), string manipulations, mathematics and much more. When your program is compiled, the code for library functions is automatically added to your program. One of the most common library functions is called **printf()** which is a general purpose output function. The quoted string between the parenthesis of the printf() function is called an argument.

Printf("This is a C program\n")

The \n at the end of the text is an **escape sequence** tells the program to print a new line as part of the output.

C DATA TYPES

In the C programming language, data types refer to a system used for declaring variables or functions of different types. **A data type is, therefore, a data storage format that can contain a specific type or range of values**. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The basic data types in C are as follows:

Type	Description
Char	Character data and is used to hold a single character. A character can be a letter, number, space, punctuation mark, or symbol - 1 byte long
Int	A signed whole number in the range -32,768 to 32,767 - 2 bytes long
Float	A real number (that is, a number that can contain a fractional part) – 4 bytes
Double	A double-precision floating point value. Has more digits to the right of the decimal point than a float – 8 bytes
Void	Represents the absence of type. i.e. represents "no data"

USING C'S DATA TYPE MODIFIERS

The five basic types (int, float, char,double and void) can be modified to your specific need using the following specifiers.

- **Signed**
  Signed Data Modifier implies that the data type variable can store positive values as well as negative values.
  The use of the modifier with integers is redundant because the default integer declaration assumes a signed number. The signed modifier is used with char to create a small signed integer. Specified as signed, a char can hold numbers in the range -128 to 127.
- **Unsigned**
  If we need to change the data type so that it can only store positive values, "unsigned" data modifier is used.
  This can be applied to char and int. When char is unsigned, it can hold positive numbers in the range 0 to 255.

- **Long**
  Sometimes while coding a program, we need to increase the Storage Capacity of a variable so that it can store values higher than its maximum limit which is there as default.
  This can be applied to both **int** and **double.** When applied to **int,** it doubles its **length**, in bits, of the base type that it modifies. For example, an integer is usually 16 bits long. Therefore a **long int** is 32 bits in length. When **long** is applied to a double, it roughly doubles the precision.
- **Short**
  A "short" type modifier does just the opposite of "long". If one is not expecting to see high range values in a program.
  For example, if we need to store the "age" of a student in a variable, we will make use of this type qualifier as we are aware that this value is not going to be very high

The type modifier precedes the type name. For example this declares a **long integer.**

long int age;

Integer Types

Following table gives you details about standard integer types with its storage sizes and value ranges:

Type	Storage size	Value range
Char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
Int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
Short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
Long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Floating-Point Types

Following table gives you details about standard floating-point types with storage sizes and value ranges and their precision:

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The void Type

The void type specifies that no value is available. It is used in three kinds of situations:

	Types and Description
1	Function returns as void. There are various functions in C which do not return value or you can say they return void. A function with no return value has the return type as void. For example, void exit (int status);
2	Function arguments as void. There are various functions in C which do not accept any parameter. A function with no parameter can accept as a void. For example, int rand(void);
3	Pointers to void A pointer of type void * represents the address of an object, but not its type. For example, a memory allocation function void *malloc(size_t size); returns a pointer to void which can be casted to any data type.

VARIABLES

A variable is a memory location whose value can change during program execution. In C a variable must be declared before it can be used.

Variable Declaration

Declaring a variable tells the compiler to reserve space in memory for that particular variable. A **variable definition** specifies a data type and the variable name and contains a list of one or more variables of that type .Variables can be declared at the start of any block of code. A declaration begins with the **type,** followed by the name of one or more variables. For example,

Int high, low;

```
int    i, j, k;
char   c, ch;
float f, salary;
```

Variables can be initialized when they are declared. This is done by adding an equals sign and the required value after the declaration.

```
Int high = 250;          /*Maximum Temperature*/
Int low = -40;           /*Minimum Temperature*/
Int results[20];         /* series of temperature readings*/
```

TYPES OF VARIABLES

The Programming language C has two main variable types

- Local Variables
- Global Variables

Local Variables

A local variable is a variable that is declared inside a function.

- Local variables scope is confined within the block or function where it is defined. Local variables must always be defined at the top of a block.
- When execution of the block starts the variable is available, and when the block ends the variable 'dies'.

Global Variables

Global variable is defined at the top of the program file and it can be visible and modified by any function that may reference it. Global variables are declared outside **all** functions.

Sample Program.

```
#include <stdio.h>
int area; //global variable

int main ()
{
  int a, b; //local variable

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

  printf("\t Side a is %d cm and side b is %d cm long\n",a,b);

  area = a*b;
  printf("\t The area of your rectangle is : %d \n", area);

  return 0;
}
```

Variable Names

Every variable has a name and a value. The name identifies the variable and the value stores data. Every variable name in C must start with a letter; the rest of the name can consist of letters, numbers and underscore characters. C is case sensitive i.e. it recognizes upper and lower case characters as being different. You cannot use any of C's keywords like main, while, switch etc as variable names,

Examples of legal variable names:
X result outfile x1 out_file etc

It is conventional in C not to use capital letters in variable names. These are used for names of constants.

Declaration vs Definition

A declaration provides basic attributes of a symbol: its type and its name. A definition provides all of the details of that symbol--if it's a function, what it does; if it's a class, what fields and methods it has; if it's a variable, where that variable is stored. Often, the compiler only needs to have a declaration for something in order to compile a file into an object file, expecting that the linker can find the definition from another file. If no source file ever defines a symbol, but it is declared, you will get errors at link time complaining about undefined symbols. In the following short code, the definition of variable x means that the storage for the variable is that it is a global variable.

```
int x;
int main()
{
    x = 3;
}
```


Inputting Numbers From The Keyboard Using Scanf()

Variables can also be initialized during program execution (run time). The **scanf()** function is used to read values from the keyboard. For example, to read an integer value use the following general form:
scanf("%d", &var_name)
As in
scanf("%d", &num)

The %d is a format specifier which tells the compiler that the second argument will be receiving an integer value.
The & preceding the variable name means "address of". The function allows the function to place a value into one of its arguments.
The table below shows format specifiers or codes used in the scanf() function and their meaning.

%c	Read a single character
%d	Read an integer
%f	Read a floating point number
%lf	Read a double
%s	Read a string
%u	Read a an unsigned integer

When used in a printf() function, a type specifier informs the function that a different type item is being displayed.

SAMPLE PROGRAM USING SCANF()

```
#include <stdio.h>
int area; //global variable

int main ()
{
  int a, b; //local variables

  /* actual initialization */
  printf("Enter the value of side a: ");
  scanf("%d", &a);

  printf("Enter the value of side b: ");
  scanf("%d", &b);
  printf("\n");
  printf("\t You have entered %d for side a and %d for side b\n", a, b);

  area = a*b;
  printf("\t The area of your rectangle is : %d \n", area);


  return 0;
}
```


CONSTANTS

C allows you to declare *constants*. When you declare a constant it is a bit like a variable declaration except the value cannot be changed during program execution.
The const keyword is used to declare a constant, as shown below:

```
int const A = 1;
const int A =2;
```

These fixed values are also called **literals**.
Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are also enumeration constants as well.
The constants are treated just like regular variables except that their values cannot be modified after their definition.

TYPE CASTING

Type casting is a way to convert a variable from one data type to another. For example, if you want to store a long value into a simple integer then you can type cast long to int. You can convert values from one type to another explicitly using the **cast operator** as follows:

```
(type_name) expression
```

Consider the following example where the cast operator causes the division of one integer variable by another to be performed as a floating-point operation:

```
#include <stdio.h>

main()
{
   int sum = 17, count = 5;
   double mean;

   mean = (double) sum / count;
   printf("Value of mean is %d \n", mean );

}
```

When the above code is compiled and executed, it produces the following result:

```
Value of mean : 3.400000
```

It should be noted here that the cast operator has precedence over division, so the value of **sum** is first converted to type **double** and finally it gets divided by count yielding a double value.

Type conversions can be implicit which is performed by the compiler automatically, or it can be specified **explicitly** through the use of the **cast operator**. It is considered good programming practice to use the cast operator whenever type conversions are necessary.

C PROGRAMMING OPERATORS

Operator is the symbol which operates on a value or a variable (operand). For example: + is an operator to perform addition.

C programming language has a wide range of operators to perform various operations. For better understanding of operators, these operators can be classified as:

OPERATORS IN C PROGRAMMING

1. Arithmetic Operators
2. Increment and Decrement Operators
3. Assignment Operators
4. Relational Operators
5. Logical Operators
6. Conditional Operators
7. Bitwise Operators
8. Special Operators

ARITHMETIC OPERATORS

Assume variable A holds 10 and variable B holds 20 then

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator - remainder of after an integer division	B % A will give 0

Note: % operator can only be used with integers.

INCREMENT AND DECREMENT OPERATORS – Unary Operators

In C, **++** and **--** are called increment and decrement operators respectively. Both of these operators are **unary operators**, i.e, used on single operand. ++ adds 1 to operand and -- subtracts 1 to operand respectively. For example:

```
Let a=5
a++;  //a becomes 6
a--;  //a becomes 5
++a;  //a becomes 6
--a;  //a becomes 5
```

Difference between ++ and -- operator as postfix and prefix

When i++ is used as prefix(like: ++var), ++var will increment the value of *var* and then return it but, if ++ is used as postfix(like: var++), operator will return the value of operand first and then increment it. This can be demonstrated by an example:

```
#include <stdio.h>
int main(){
    int c=2;
    printf("%d\n",c++); /*this statement displays 2 then,
                  only c incremented by 1 to 3.*/
    printf("%d",++c);   /*this statement increments 1 to
                  c then, only c is displayed.*/
    return 0;
}
```


Output

```
2
4
```

ASSIGNMENT OPERATORS – Binary Operators

The most common assignment operator is =. This operator assigns the value in the right side to the left side. For example:

```
var=5  //5 is assigned to var
a=c;   //value of c is assigned to a
5=c;   // Error! 5 is a constant.
```

Operator	Example	Same as
=	a=b	a=b
+=	a+=b	a=a+b
-=	a-=b	a=a-b
*=	a*=b	a=a*b

Operator	Example	Same as
/=	a/=b	a=a/b
%=	a%=b	a=a%b

NB/ += means Add and Assign etc.

RELATIONAL OPERATORS - Binary Operators

Relational operators check relationship between two operands. If the relation is true, it returns value 1 and if the relation is false, it returns value 0. For example:

a>b

Here, > is a relational operator. If *a* is greater than *b*, *a>b* returns 1 if not then, it returns 0.

Relational operators are used in decision making and loops in C programming.

Operator	Meaning of Operator	Example
= =	Equal to	5= =3 returns false (0)
>	Greater than	5>3 returns true (1)
<	Less than	5<3 returns false (0)
!=	Not equal to	5!=3 returns true(1)
>=	Greater than or equal to	5>=3 returns true (1)
<=	Less than or equal to	5<=3 return false (0)

LOGICAL OPERATORS - Binary Operators

Logical operators are used to combine expressions containing relational operators. In C, there are 3 logical operators:

Operator	Meaning of Operator	Example
&&	Logical AND	If c=5 and d=2 then,((c= =5) && (d>5)) returns false.
||	Logical OR	If c=5 and d=2 then, ((c= =5) || (d>5)) returns true.
!	Logical NOT	If c=5 then, !(c= =5) returns false.

The following table shows the result of operator && evaluating the expression a&&b:

&& OPERATOR (and)		
a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

The operator || corresponds to the Boolean logical operation OR, which yields true if either of its operands is true, thus being false only when both operands are false. Here are the possible results of a || b:

|| OPERATOR (or)		
a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Explanation

For expression, *((c==5) && (d>5))* to be true, both *c==5* and *d>5* should be true but, (d>5) is false in the given example. So, the expression is false. For expression `((c==5) || (d>5))` to be true, either the expression should be true.

Since, `(c==5)` is true. So, the expression is true. Since, expression `(c==5)` is true, `!(c==5)` is false.

CONDITIONAL OPERATOR – Ternary Operators

Conditional operator takes three operands and consists of two symbols **?** and **:** . Conditional operators are used for decision making in C. For example:

```
c=(c>0)?10:-10;
```

If *c* is greater than 0, value of *c* will be 10 but, if *c* is less than 0, value of *c* will be -10.

BITWISE OPERATORS

Bitwise operators work on bits and performs bit-by-bit operation.

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -60, which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15, which is 0000 1111

PRECEDENCE OF OPERATORS

If more than one operator is involved in an expression then, C language has a **predefined rule of priority of operators**. This rule of priority of operators is called **operator precedence**.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. **Within an expression, higher precedence operators will be evaluated first.**

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

ASSOCIATIVITY OF OPERATORS

Associativity indicates in which order two operators of same precedence (priority) executes. Let us suppose an expression:
a= =b!=c