Basic Structure of C Program

Basic Structure of C Programs
Documentation Section
Link Section
Definition Section
Global Declaration Section

main() Function Section
{
Declaration Part
Executable Part
}

Subprogram Section Function 1 Function 2 Function 3 - - - Function n

The **documentation section** contains a set of comment including the name of the program other necessary details. Comments are ignored by compiler and are used to provide documentation to people who reads that code. Comments are be giving in C programming in two different ways:

1 Single Line Comment

2 Multi Line Comment

```
// This is a single line comment
/* This is
Multi-line
Comment */
```

The **link section** consists of header files while contains function prototype of Standard Library functions which can be used in program. Header file also consists of macro declaration. Example:

```
#include <stdio.h>
```

In the above code, stdio.h is a header file which is included using the preprocessing directive #include

The **definition section** defines all symbolic constants. A symbolic constant is a constant value given to a name which can't be changed in program.

Example:

#define PI 3.14
In **global declaration** section, global variables and user defined functions are declared.
The **main () function section** is the most important section of any C program. The compiler start executing C program from main() function. The main() function is mandatory in C programming. It has two parts:

Declaration Part - All the variables that are later used in the executable part are declared in this part.
Executable Part - This part contains the statements that are to be executed by the compiler

```
void main()
{
    ……….
    ……….
}
```

The **subprogram section** contains all the user defined functions.
Printing a Message

```
void main()
{
    /*…….printing begins ……*/
    printf(“I see, I remember”);
    /*…….printing ends……*/
}
```

printf is a predefined standard C function for printing output.

Basic C Program to print Hello World (using vi editor)

```
//This is a very simple program to print Hello World
#include <stdio.h>
int main()
{
     printf(“Hello World”);
     return 0;
}
```

CHARACTER SET
The characters that can be used to form words, numbers, and expressions depend upon the computer the computer on which the program is run. The characters in C are grouped into the following categories:

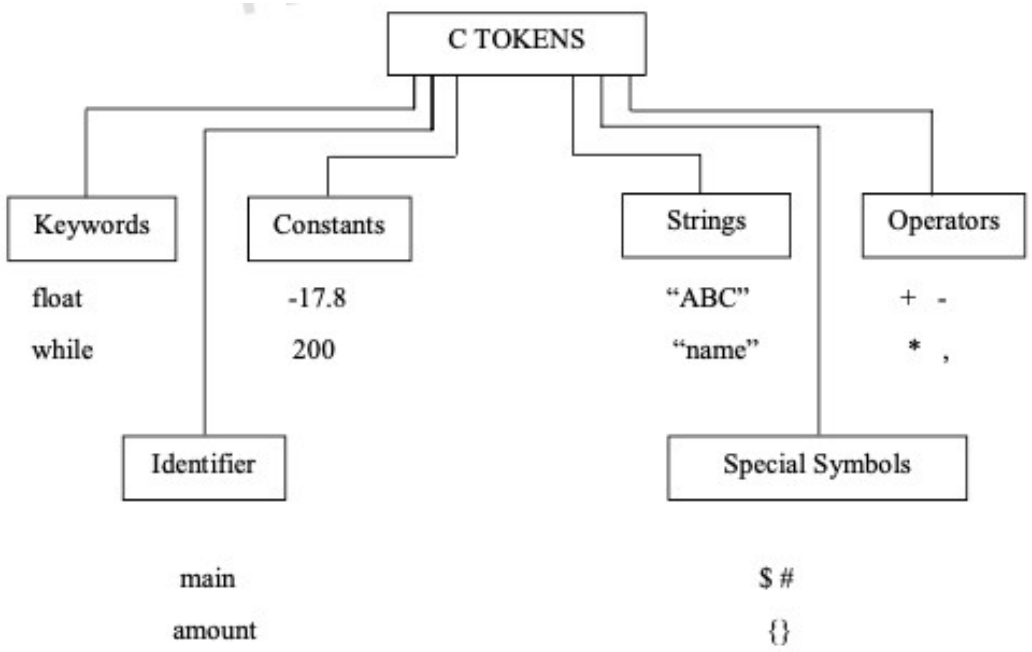
1. Letters- **Uppercase A-Z , Lower case a-z**
2. Digits- **0,1,2,3,4,5,6,7,8,9**

3 Special characters **-/,|,% etc**

4 White space **-\b,\t etc**

C Tokens

In a passage of text, individual words and punctuation marks are called tokens. Similarly, in a C program the **smallest individual units** are known as C tokens.



Keywords in C

Every C word is classified either as a keyword or as an identifier. **Keywords are identifiers that are reserved by the language for special use**. All keywords must be written in lower case.

ANSI C Keywords

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

Identifiers

Identifiers refer to the **names of variables, functions and arrays.** These are user defined names and consist of a sequence of letters and digits, with a letter as the first character. Both upper case and lower case letters are permitted, although lower case is commonly used. An underscore is usually used as a link between long identifiers.

Rules for Identifiers

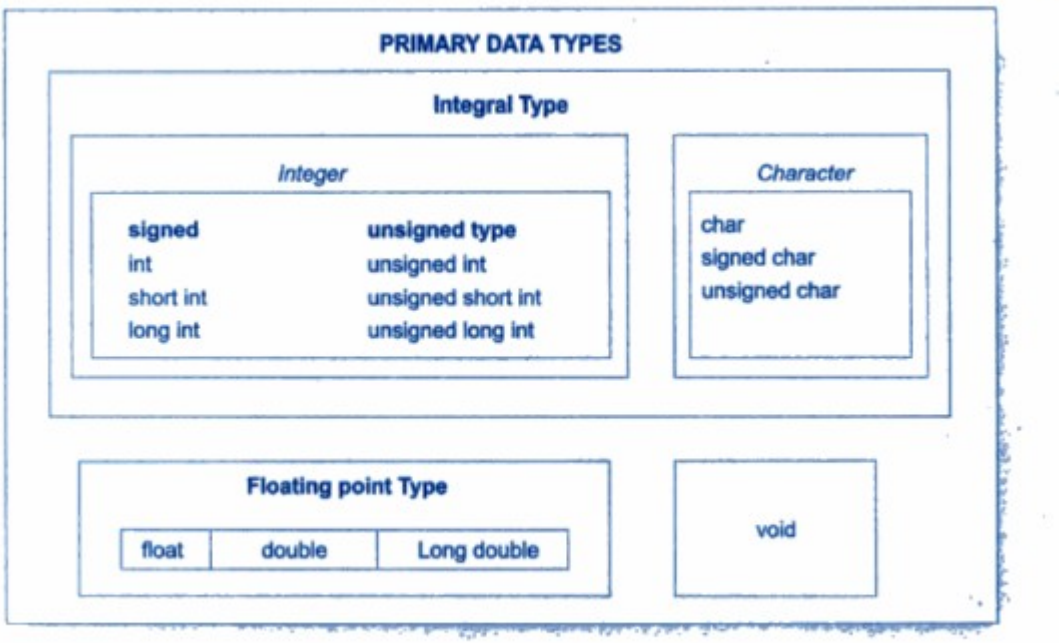
1. First character must be an alphabet (or underscore).
2. Must consist of only letters, digits or underscore.
3. Only first 31 characters are significant.
4. Cannot use a keyword.
5. Must not contain white space.

Variables

A variable is a data name that may be used to store a data value. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during execution. Variable names can be chosen in a meaningful manner.
Eg:
Average, height, Total, Counter_1, classStrength
Length of variables should not be normally more than 8 characters although 31 characters are recognized.

Data types in C

All C compilers support five fundamental data types, namely **integer (int), character (char), floating point (float), double precision floating point (double) and void**. There are also extended data types such as long int and long double.



Primary data types in C

Size and Range of Data Types on a 16-bit Machine

Type	*Size (bits)*	*Range*
char or signed char	8	–128 to 127
unsigned char	8	0 to 255
int or signed int	16	–32,768 to 32,767
unsigned int	16	0 to 65535
short int or signed short int	8	–128 to 127
unsigned short int	8	0 to 255
long int or signed long int	32	–2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
float	32	3.4E – 38 to 3.4E + 38
double	64	1.7E – 308 to 1.7E + 308
long double	80	3.4E – 4932 to 1.1E + 4932

32 bit machine

short int , signed short, unsigned short – 2 bytes
int, signed int, unsigned int – 4 bytes
long int, unsigned long int, signed long int – 4 bytes
float -4 bytes
double -8 bytes
long double – 16 bytes

Declaration of Variables

The syntax for declaring a variable is as follows:
data-type v1, v2….vn
where v1, v2…vn are the names of the variables.
Eg:
int count;
int number, total;
double ratio;

```
main() /*.........Program Name..................... */
{
  /*................Declaration.....................*/
    float       x, y;
    int         code;
    short int   count;
    long int    amount;
    double      deviation;
    unsigned    n;
    char        c;
  /*...............Computation..................... */
    . . . .
    . . . .
    . . . .
} /*.............Program ends.....................*/
```

Declaration of variables

Enumerated data types
enum identifier {value1,value2,.....valuen};
Then declare variables
enum identifier v1,v2.....vn;

v1=value1;
v2=value3;

Example
enum day {Monday, Tuesday,.....Sunday};
enum day week_st, week_end;

week_st=Monday;
week_end=Friday;

Compiler assigns integer digits beginning from zero , automatic assignment can be overridden by assigning values explicitly.

Operators and Expressions

C operators can be classified into a number of categories

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment and decrement Operators
- Conditional Operators
- Bitwise Operators

Arithmetic Operators

Operator	*Meaning*
+	Addition or unary plus
–	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

a and b are integers, a=14 and b=4
a-b= 10

a+b = 18

a*b = 56

a/b = 3 (decimal part truncated)

a%b = 2 (remainder)

Relational Operators

Operator	*Meaning*
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

Logical Operators

&& meaning LOGICAL AND
|| meaning LOGICAL OR
! meaning LOGICAL NOT

Assignment Operators

Statement with simple assignment operator	*Statement with shorthand operator*
a = a + 1	a += 1
a = a – 1	a –= 1
a = a * (n+1)	a *= n+1
a = a / (n+1)	a /= n+1
a = a % b	a %= b

Increment and Decrement Operators

increment operator ++
decrement operator --
The operator ++ adds 1 to the operand while – subtracts 1. Both are unary operators.
++m is equivalent to m=m+1
--m is equivalent to m=m-1

m= 5;
y= ++m; // equivalent to m=m+1 and y=m
after these two statements m=6, y=6 (pre-increment operation)

m=5;

y=m++; after these two statements m=6, y=5. (post-increment operation)
y=m;
++m;

Conditional Operator

A ternary operator is available in C to construct conditional expression
exp1 ? exp2 : exp3

where exp1, exp2 and exp3 are expressions
a=10;
b=5;
x= (a > b) ? a : b;
If a is greater than b, assign x=a, otherwise assign x=b.

Bitwise Operators

These operators are used for testing the bits or shifting them right or left.

Bitwise Operators

Operator	*Meaning*
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right

AND operation

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
  00001100
& 00011001
  ________
  00001000  = 8 (In decimal)
```

XOR operation

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25
  00001100
^ 00011001
  ________
  00010101  = 21 (In decimal)
```

Shift operation

```
212 = 11010100 (In binary)
212>>2 = 00110101 (In binary) [Right shift by two bits]
212>>7 = 00000001 (In binary)
212>>8 = 00000000
212>>0 = 11010100 (No Shift)
```

sizeof Operator

The sizeof is a compile time operator and, when used with an operand, it returns the number of bytes the operand occupies. The operand may be variable, a constant or a data type qualifier. Examples:

m = sizeof (sum);
n = sizeof (long int);

The sizeof operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer. It is also used to allocate memory space dynamically to variables during execution of a program.

Arithmetic Expressions

An arithmetic expression is a combination of variables, constants and operators arranged as per the syntax of the language

Expressions

Algebraic expression	*C expression*
a x b - c	a * b - c
(m+n) (x+y)	(m+n) * (x+y)
$\left(\frac{ab}{c}\right)$	a * b/c
$3x^2 + 2x + 1$	3 * x * x + 2 * x + 1
$\left(\frac{x}{y}\right) + c$	x/y+c

Operator Precedence and Associativity

Operator precedence is used to determine how an expression involving more than one operator is evaluated. There are distinct levels of precedence and operator may belong to one of these levels. The operators at the higher level of precedence are evaluated first.

When the operators have same precedence level, they are evaluated either from left to right or from right to left depending on the level. This is known as associative property of an operator.

If (x==25 && y<10) is evaluated to FALSE if x=20 and y=5.

Summary of C Operators

Operator	*Description*	*Associativity*	*Rank*
( ) [ ]	Function call Aray element reference	Left to right	1
+ – ++ – – ! ~ * & sizeof (type)	Unary plus Unary minus Increment Decrement Logical negation Ones complement Pointer reference (indirection) Address Size of an object Type cast (conversion)	Right to left	2
* / %	Multiplication Division Modulus	Left to right	3
+ –	Addition Subtraction	Left to right	4
<< >>	Left shift Right shift	Left to right	5
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left to right	6
== |=	Equality Inequality	Left to right	7
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
|	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
||	Logical OR	Left to right	12
?:	Conditional expression	Right to left	13
= * = /= %= += –= &= ^= |= <<= >>=	Assignment operators	Right to left	14
,	Comma operator	Left to right	15

Input and Output functions

All input/output operations are carried out through function calls such as printf and scanf.
The general form of scanf is
scanf(“control string”, arg1, arg2,……argn)
control string specifies the field format in which data is entered and the arguments arg1,arg2….argn specify the address of locations where data is stored.
Control string contains field specifications which direct the interpretation of input data. It may include

- Field (or format) specifications, consisting of the conversion character %, a data type character (or type specifier) and an optional number specifying the field width.
- Blanks, tabs or newlines

Eg: scanf(“%d %d”, &num1, &num2);
scanf reads real numbers using the specification %f for decimal and exponential notation.
scanf uses %c to read a single character. And to read character strings %wc or %ws is used.
Eg:

```
main()
{
int no;   //no is a variable of type integer
char name[15];// name is an array of type character
printf(“enter a number and a string”);
scanf(“%d %s”, &no, name);
}
```

no=5
printf(“the number entered is %d \n”, no);
the number entered is 5
The general form of printf is
printf(“control string”, arg1, arg2,……argn)
Control string consists of three types of items

- Characters that will be printed on the screen as they appear
- Format specifications that define the output format for display of each item
- Escape sequence characters such as \n,\t and \b.

1e2 = 1*10^2

%c	Character
%d	Signed integer
%e or %E	Scientific notation of floats
%f	Float values
%g or %G	equivalent to either %f or %e - whichever would produce shorter output for the provided value.

%hi	Signed integer (short)
%hu	Unsigned Integer (short)
%i	signed integer
%l or %ld or %li	Long
%lf	Double
%Lf	Long double
%lu	Unsigned int or unsigned long
%lli or %lld	Long long
%llu	Unsigned long long
%o	Octal representation
%p	Pointer
%s	String
%u	Unsigned int
%x or %X	Hexadecimal representation
%n	Prints nothing
%%	Prints % character

- A minus symbol (-) sign tells left alignment
- A number after % specifies the minimum field width. If string is less than the width, it will be filled with spaces
- A period (.) is used to separate field width and precision

Examples

```
printf("programming in C");
printf(" ");
printf("%d", x);
printf("a= %f\n b=%f", a, b);
printf("\n\n");
```

document print_format.c explanation

Other input and output functions:

getchar() reads one character from standard input
putchar() writes one character to standard output
gets()
puts()

Examples

1
```
char name;
name = getchar();
```

If we press the key H on the keyboard, this will assign character 'H' to the variable name.
Note the parantheses for getchar() function.

2 char answer;
answer = `Y`;
putchar (answer)
This will display character Y on the screen.

3 char line[80]; // line is an array of characters
gets(line);

4 char line[80];
gets(line);
puts(line);

VI Editing commands

- i – Insert at cursor (goes into insert mode)
- a – Write after cursor (goes into insert mode)
- A – Write at the end of line (goes into insert mode)
- ESC – Terminate insert mode
- u – Undo last change
- U – Undo all changes to the entire line
- o – Open a new line (goes into insert mode)
- dd – Delete line

Saving and Closing the file

- Shift+zz – Save the file and quit
- :w – Save the file but keep it open
- :q – Quit without saving
- :wq – Save the file and quit

Coding exercises
1)

```
#include<stdio.h>
void main()
{
    int i=100;
    printf("%d\n",i);
    int *pointer = &i;
    printf("%p\n",i);
    printf("%p",pointer);
}
```

2)

```
#include <stdio.h>
int main()
{
    int a = 15;
    printf("%x\n", a);
    printf("%o\n", a);
    printf("%i\n", a);
    float b = 12.67;
    printf("%f\n", b);
    printf("%e\n", b);
    printf("%g\n", b);
    return 0;
}
```

3)

```
printf( "%.3f", 1.2 );
printf( "%.3f", 1.2348 );
printf( "%.3d", 10 );
printf( "%.5s\n", "abcdefg" );
printf( "%5s\n", "abc" );
printf( "%8.5f\n", 1.234 );
printf( "%05d\n", 10 );
printf( "%+d\n", 10 );
```

4)

```
#include <stdbool.h> // Header-file for boolean data-type.
#include <stdio.h>
int main()
{
    bool x=false;  // Declaration and initialization of boolean
    if (x==true) {  // Conditional statement.
        printf("The value of x is true");
    } else {
        printf("The value of x is false");
    }

    return 0;
    // Output: The value of x is false
}
```

5) Evaluate

```
x = 9 – 12 / 3 + 3 *2 - 1
x = 9 – 12 / (3 + 3) * (2 – 1)
y= 6*2/( 2+1 * 2/3 +6) +8 * (8/4)
```

6)

```
// Working of increment and decrement operators
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;

    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);

    return 0;
}
```

7)

```
// Working of assignment operators
#include <stdio.h>
int main()
{
    int a = 5, c;

    c = a;      // c is 5
    printf("c = %d\n", c);
    c += a;     // c is 10
    printf("c = %d\n", c);
    c -= a;     // c is 5
    printf("c = %d\n", c);
    c *= a;     // c is 25
    printf("c = %d\n", c);
    c /= a;     // c is 5
    printf("c = %d\n", c);
    c %= a;     // c = 0
    printf("c = %d\n", c);

    return 0;
}
```

8)

```
// Working of relational operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;

    printf("%d == %d is %d \n", a, b, a == b);
    printf("%d == %d is %d \n", a, c, a == c);
    printf("%d > %d is %d \n", a, b, a > b);
    printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
```

```
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);

    return 0;
}
```

9)

```
// Working of logical operators

#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) is %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) is %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) is %d \n", result);

    result = !(a != b);
    printf("!(a != b) is %d \n", result);

    result = !(a == b);
    printf("!(a == b) is %d \n", result);

    return 0;
}
```

10)

```
#include <stdio.h>
int main()
{
    int a;
    float b;
    double c;
    char d;
    printf("Size of int=%lu bytes\n",sizeof(a));
    printf("Size of float=%lu bytes\n",sizeof(b));
    printf("Size of double=%lu bytes\n",sizeof(c));
    printf("Size of char=%lu byte\n",sizeof(d));

    return 0;
```

```
}
```

11)

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a&b);
    return 0;
}
```

12)

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a^b);
    return 0;
}
```

13)

Check whether True or false (0 or not):

a. 4>5 && 5>4

b. 4>5 || 5>4

c. (232+23*1233) || 0

d. (232+23*1233) && 0

14)

What would be the value of 'a':

a. int a = 10/45*23%45/(45%4*21)

b. float a = 10+45.0*23-45+(4*21.0)

15)

Find the output of:

printf("%d\n",1==5==5);