COMPUTER PROGRAMMING USING C

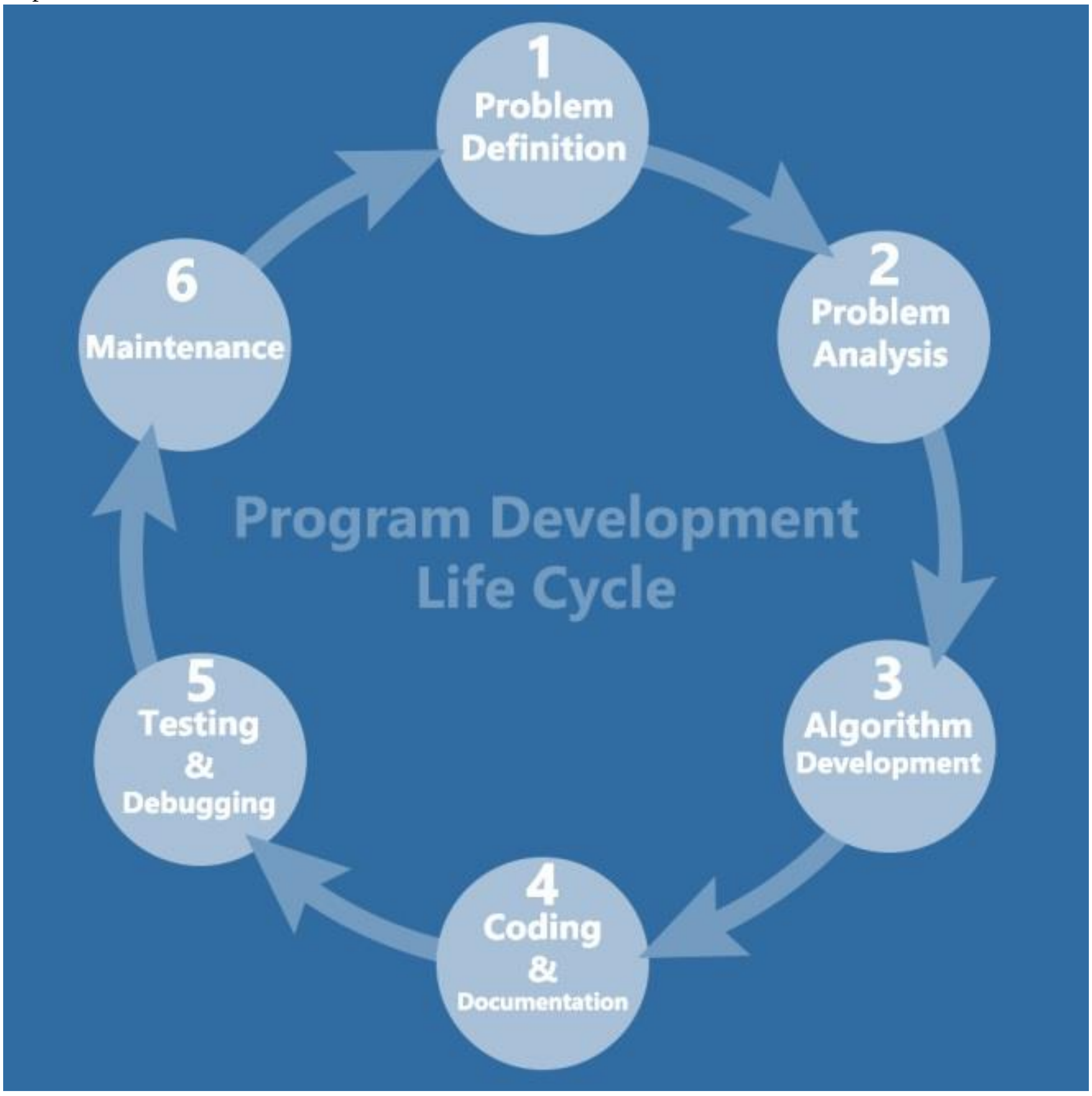
1. Algorithm and Programming Development

C programming language works as an interface between the programmer and the computer.

PDLC: Program Development Life Cycle

PDLC consists of seven steps:

- Problem Definition/Analysis
- Program Designing
- Algorithm Development and Flowcharting
- Program Coding
- Debugging and Compilation
- Program Testing
- Implementation and Documentation



1). Problem Definition/Analysis

Before writing a program, it is must to have proper understanding of the problem.

2). Program Designing

It consists of five steps:

- Design Input
- Design Output
- Code Design
- Form Design
- Module Design

3). Algorithm Development and Flowcharting

Algorithm:

It is a step by step solution of the problem written in simple language.

Algorithm Example 1: Write an algorithm to add two nos.

Step1: start

Step2: Read a, b

Step3: sum = a + b

Step4: write sum

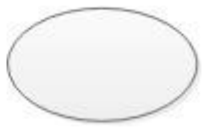
Step5: stop

Flowchart:

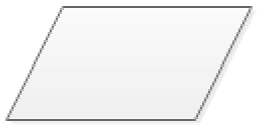
It is the graphical representation of an algorithm.

Several standard graphics are applied in a flowchart:

- Terminal Box - Start / End

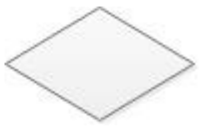


- Input / Output

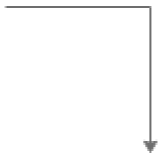


- Process / Instruction

- Decision



- Connector / Arrow



The graphics above represent different part of a flowchart. The process in a flowchart can be expressed through boxes and arrows with different sizes and colors. In a flowchart, we can easily highlight a certain element and the relationships between each part.

How to Use Flowcharts to Represent Algorithms

Now that we have the definitions of algorithm and flowchart, how do we use a flowchart to represent an algorithm?

Algorithms are mainly used for mathematical and computer programs, whilst flowcharts can be used to describe all sorts of processes: business, educational, personal and of course algorithms. So flowcharts are often used as a program planning tool to visually organize the step-by-step process of a program. Here are some examples:

Print numbers from 1 to 20:

Algorithm:

Step 1: Initialize X as 0,

Step 2:

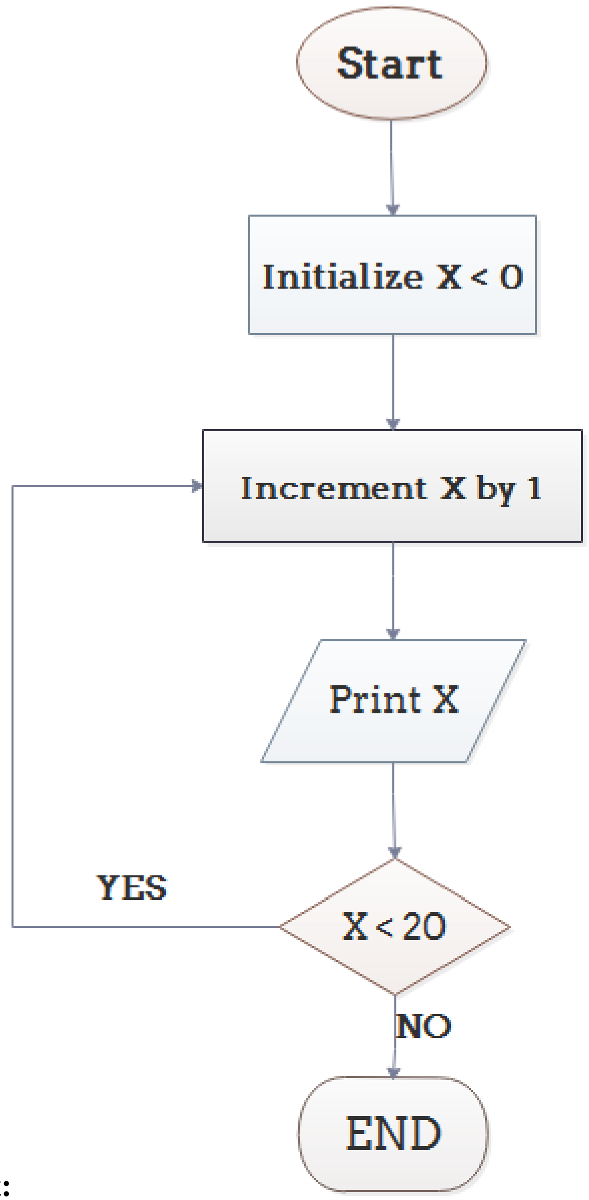
Increment X

by 1,

Step 3: Print

X,

Step 4: If X is less than 20 then go back to step 2.



Flowchart:

Example 2: Convert Temperature from Fahrenheit (°F) to

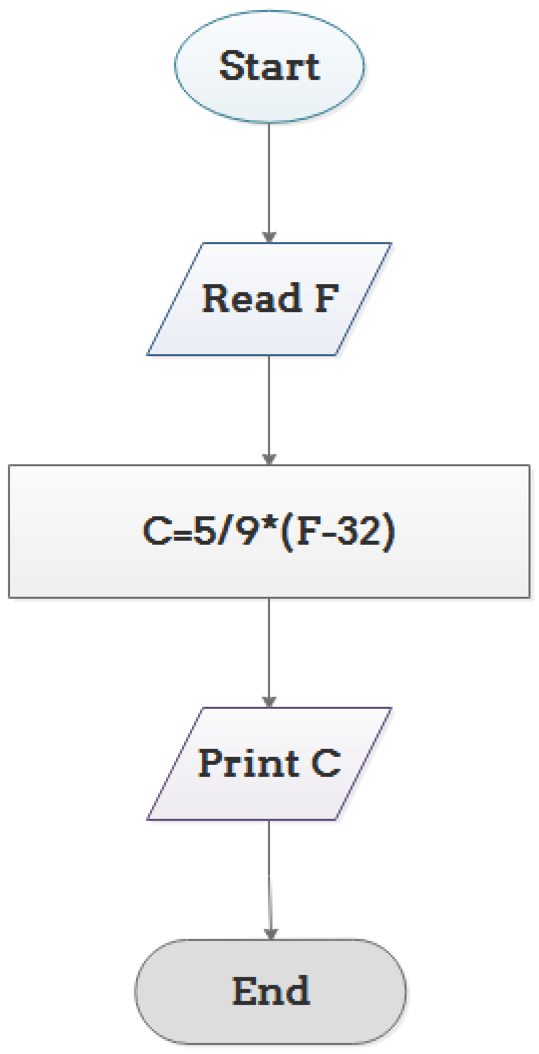
Celsius (°C)Algorithm:

Step 1: Read temperature in Fahrenheit,

Step 2: Calculate temperature with formula

C=5/9*(F-32),Step 3: Print C

Flowchart:



Conclusion

From the above we can come to a conclusion that a flowchart is pictorial representation of analgorithm, an algorithm can be expressed and analyzed through a flowchart.

An algorithm shows you every step of reaching the final solution, while a flowchart shows youhow to carry out the process by connecting each step. An algorithm uses mainly words to

describe the steps while a flowchart uses the symbols, shapes and arrows to make the process more logical.

4). Program Coding:

When the algorithm is developed and tested; it needs implementation using that language. This is known as program coding.

5). Debugging and Compilation:

Once the program is ready, it must be checked for errors. An error is also known as bug.

Error/Bug: there are three types of errors or bugs:

- Syntax error
- Semantics error/Logical error
- Execution error

a).Syntax Error:

A grammatical error is known as syntax error.

Example of syntax error:

Correct: printf("Welcome to the world of C");

Incorrect: printf("Welcome to the world of C");

The incorrect statement is not terminated by a semicolon

b).Semantics error/Logical error:

There is requirement to write a program to add two numbers; but the programmer did subtraction.

c=a+b; (correct)

c=a-b; (incorrect)

c).Execution Error:

execution error is also known as runtime error.

Example:

1. divide by zero
2. infinite loop

Debugging:

Isolation of errors and their removal is known as debugging.

Compilation:

Compilation is the process of converting source program into an object program.

Source program:

The program written in C (middle level language).

Object program:

The program written in machine language (0’s & 1’s).

Shortcut for compilation in C is: Alt+F9

6).Program Testing:

The program is tested before implementation.

For testing a program, some sample data is calculated manually.

Then, same data is given as input to the program.

The output of program is compared to the manual output.

If the result is ok, then there are no logical errors present in the program; but, if the output is incorrect, it means that some logical errors are present in the program. To locate these errors, one has to go back third step i.e. check algorithms and flowcharts and locate problems. Correct problems, recode the program, recompile program and test program again. These steps are repeated time and again; until, output during testing becomes correct.

7).Implementation and documentation:

There are three main steps:

1. Installation of software
2. Maintenance of software
3. Documentation of software

Characteristics of good program:

1. Efficient
2. Reliable
3. Flexible
4. Extensible
5. Portable
6. Integrity
7. Robustness
8. Clarity

Program Approaches:

- Top down approach
- Modular approach or structured programming
- Bottom up approach

C language follows top down approach.

C program structure:

- C is a middle level language as it has features of both Low Level Language & High Level Language.

BRIEF HISTORY OF C:

C was developed in 1972 by Ken Thompson and Dennis Ritchie at Bell Laboratories, USA.

PREDESSOR OF C:

- **B/BCPL**(Basic Combined Programming Language)

SUCESSOR OF C:

- **C++**

Working With C:

The most commonly used C compilers are:

Borland C (BC)

Turbo C (TC)

Turbo C is more followed than Borland C.

Source Program (written in C) Alt+F9 (compilation) **program.c**	Compiled Program (object program) Ctrl+F9 (run/execution) **program.obj**	Executable Program Alt+F5 (to view the exe file) **Program.exe**

Compiler Linker

Compiler:

The compiler converts source program to object program.

Linker:

The linker converts object program to executable program.

The C character set:

Type of character	Description	characters
Lowercase alphabets	a to z	A,b,…z
Uppercase alphabets	A to Z	A,B…Z
Digits	0 to 9	0,1,…9
Special characters	` to ?	`,~,…?
White spaces	Blank spaces to new line	Blank spaces, carriage return,…new line

Special characters:(32)

`	**Backtick**
~	**Tilde**
@	**At the rate of**
!	**exclamation mark**
$	**Dollar**
#	**Hash**
^	**Caret**
*	**Asterisk**
%	**percentage**
&	**ampersand**
(	**Left parenthesis**
)	**right parenthesis**
[	**left bracket**
]	**Right bracket**
{	**Left brace**
}	**Right brace**
<	**Less than**
>	**Greater than**
+	**Add**
=	**Equal to**
_	**Underscore**
-	**Minus**
|	**Vertical bar**
/	**Slash**
\	**Backslash**
;	**Semicolon**
:	**Colon**
‘	**apostrophe**
“	**Quotation mark**
,	**Comma**
.	**Period**
?	**Question mark**

White spaces:

Blank spaces	Single spacebar
Carriage return	Enter

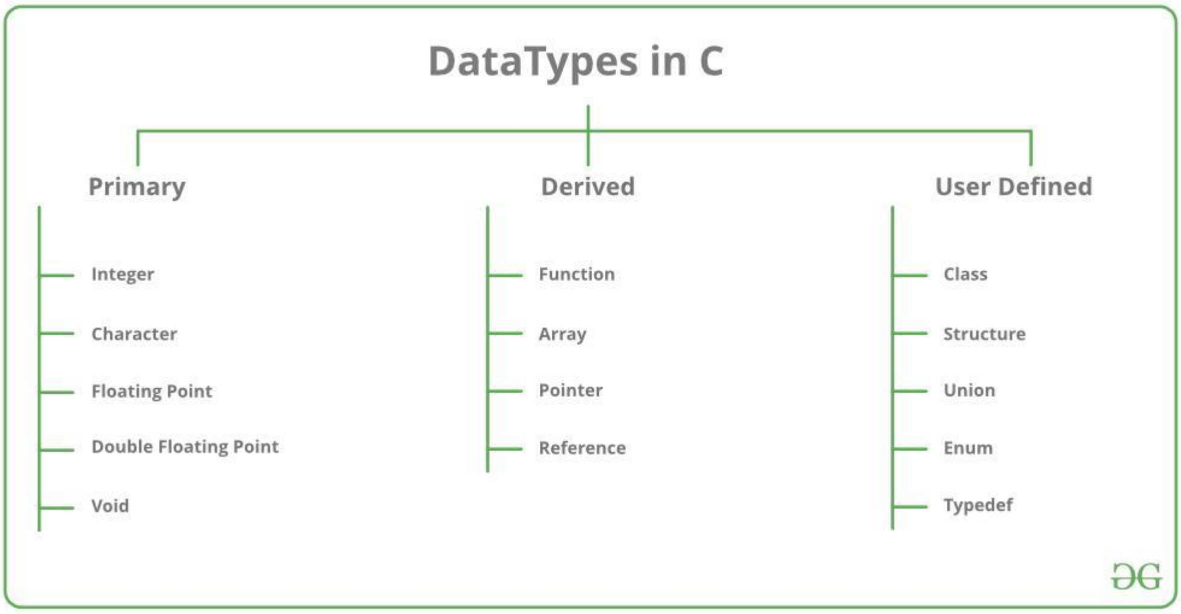
Horizontal tab	Eight horizontal spaces
Vertical tab/Form feed	Eight vertical spaces
New line	Next line

Escape sequences:

Escape Sequences	
Escape Sequences	Character
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark

\?	Question mark
\0	Null character

Data types in C:



Primary data types in C:

Data type	Format specifier
integer	“%d”
character	“c”
float	“%f”
double	“%lf”
void	void

Data Type	Format Specifier	Minimal Range	Typical Bit Size
unsigned char	%c	0 to 255	8

char	%c	-127 to 127	8
signed char	%c	-127 to 127	8
int	%d, %i	-32,767 to 32,767	16 or 32
unsigned int	%u	0 to 65,535	16 or 32
signed int	%d, %i	Same as int	Same as int 16 or 32
short int	%hd	-32,767 to 32,767	16
unsigned short int	%hu	0 to 65,535	16
signed short int	%hd	Same as short int	16
long int	%ld, %li	-2,147,483,647 to 2,147,483,647	32
long long int	%lld, %lli	-(263 – 1) to 263 – 1 (It will be added by the C99 standard)	64
signed long int	%ld, %li	Same as long int	32
unsigned long int	%lu	0 to 4,294,967,295	32
float	%f	1E-37 to 1E+37 along with six digits of the precisions here	32
double	%lf	1E-37 to 1E+37 along with six digits of the precisions here	64
long double	%Lf	1E-37 to 1E+37 along with six digits of the precisions here	80

Storage classes in C:

Class	**Name of Class**	**Place of Storage**	**Scope**	**Default Value**	**Lifetime**
auto	Automatic	RAM	Local	Garbage Value	Within a function
extern	External	RAM	Global	Zero	Till the main program ends. One can declare it anywhere in a program.

static	Static	RAM	Local	Zero	Till the main program ends. It retains the available value between various function calls.
register	Register	Register	Local	Garbage Value	Within the function

Derived types

(a) Pointer types,

(b) Array types,

(c) Structure types,

(d) Union types and

(e) Function types.

C Operators:

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Special Operators

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200

/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

Operator	Description
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

p	q	p & q	p | q
0	0	0	0
0	1	0	1
1	1	1	1
1	0	0	1

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

Show Examples

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary EXOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. ~00111100 11000011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 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 = 15 i.e., 0000 1111

Assignment Operators

The following table lists the assignment operators supported by the C language −

Show Examples

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2

|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Increment & Decrement operators:

There are two types of Increment operators in C:

a).prefix increment

b=++a;

In prefix increment, the current value of a will be incremented first by 1. Then the new value will be assigned to b for the expression in which it is used in the same line.

b).postfix increment

b=a++;

The postfix increment operator allows the usage of the current value of a variable in an expression and then increments its value by 1 in next line.

c).prefix decrement:

b=--a;

In prefix decrement, the current value of a will be decremented first by 1. Then the new value will be assigned to b for the expression in which it is used in the same line.

d).postfix decrement:

b=a--;

The postfix decrement operator allows the usage of the current value of a variable in an expression and then decrements its value by 1 in next line.

Syntax of conditional operator in C

variable = condition? statement 1: statement 2;

If the condition is true(1), then statement 1 will be executed & its value will be assigned to the variable.

If the condition is false(0), then statement 2 will be executed & its value will be assigned to the variable.

Special Operators in C:

There are 4 special operators in C:

a).Comma operator

b).& operator

c).* operator

d).sizeof operator

a).Comma operator

it is used to combine many operators together

example:

for(a=0,b=0;a<=10;a++)

b).& operator

it is also known as address operator. It is used to print address of any memory location.

c).* operator

it is also known as indirection operator.

Example:

int *ptr;

float *ptr1;

d).sizeof operator

it is used to measure size of memory occupied by any variable.

C Keywords:

There are 32 keywords in C.

These are reserved words and can’t be used for variable or identifier name.

C Keywords

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

Control structures in C:

Control structures are of four types:

- Sequence control statements
- Decision control statements/ Conditional control statements
- Case control statements
- Loop control statements

Sequence control statements:

These statements are executed in same order in which they appear in the program. Such a program is known as monolithic program. No control structure is needed in such programs.

Decision control statements/ Conditional control statements:

- **If statement**
- **If-else statement**
- **Nested if statement**

a). If statement:

it is used when a set of statements are to be executed depending upon a specified condition.

Syntax of if statement:

```
If (condition)

{

Set of statements to be executed when if condition is true.

}
```

Example:

```
#include<stdio.h>

void main( )

{

int a,b;

printf("enter a & b");

scanf("%d %d",&a, &b);

if (a>b)

{

printf("a is greater");

printf("b is smaller");

}
}
```

b).If-else statement:

this statement is used when there are two possible results of a question.

Syntax of if-else statement:

```
if (condition)
{
Statements;
}
else
{
Statements;
}
```

Example:

```
#include<stdio.h>
void main( )
{
int a,b;
printf("enter a & b");
scanf("%d %d",&a, &b);
if (a>b)
{
printf("a is greater");
}
else
printf("b is greater");
}
}
```

c).Syntax of nested if:

```
if(condition1)
{statements}
```

else{if (condition2)

{statements}

else

}

}

Switch statement:

Syntax of switch statement:

Switch(expression)

{

Case constant 1:

Statements;

Case constant 2:

Statements;

Default :

Statements;

}

Break statement:

It is used to transfer control out of the current loop.

Syntax of break statement:

break;

exit function:

syntax:

exit( );

it transfers the control out of the program. It stops the execution of the program.

Loop structures in C:

- For loop
- While loop
- Do while loop

a).for loop:

syntax of for loop:

for(initialization part; conditional part; increment/decrement part)

{

statements;

}

b).while loop:

syntax of while loop:

while(condition)

{

Statements;

}

If condition is true, statements will be executed; otherwise statements will not be executed.

c).do-while loop:

syntax of do while loop:

do

{

Statements;

}while(condition);

In do while loop condition is checked at the end of the loop.

So, do while loop will be executed at least once; even though the condition is false.

Functions in C:

Function:

A function is a subroutine that performs a particular task.

Types of functions:

a). user defined functions

b). built-in functions

a). user defined functions:

these are the functions that are defined by the user.

b). built-in functions:

these functions are pre-defined in C header files.

For example:

The function clrscr( ) is defined in header file conio.h

Examples of some header files:

pow( ), printf( ), scanf( ), sin( ), cos( )

Argument:

It is used to pass the information between functions and main( ).

Types of Arguments:

1. formal arguments or dummy arguments

2. actual arguments

```
#include<stdio.h>

#include<conio.h>

/* program to add two numbers using function */

void main( )

{

int a, b;

clrscr();

printf("enter a and b");

scanf("%d %d",&a, &b);

add (a, b); /* call to add function. a and b are actual arguments */

}

void add(int a1, int b1) /*a1 and b1 are formal or dummy arguments */
```

```
{
int c;
c=a+b;
printf("%d",c);
}
```

Parameter passing in functions:

- Call by value
- Call by reference

Call by value:

In call by value, the value of actual parameter is passed to formal parameter.

```
/* program to swap two numbers using call by value */
#include<stdio.h>
#include<conio.h>
main()
{
int a, b;
clrscr();
printf("enter a and b");
scanf("%d %d", &a,&b);
swap(a,b);
printf("value in main after swapping");
printf("%d %d", a, b);
}
void swap (int a1, int b1)
{
Int c;
c=a1;
a1=b1;
b1=c;
printf("value in function after swapping");
printf("%d %d", a1, b1);
}
```

Output:

```
Enter a and b 5 6
value in function after swapping 6 5
value in main after swapping 5 6
```

Call by reference:

In call by reference, the address of actual parameters is passed to formal parameters.

```
/* program to swap two numbers using call by value */
#include<stdio.h>
#include<conio.h>
main()
{
int a, b;
clrscr();
printf("enter a and b");
scanf("%d %d", &a,&b);
swap(&a,&b);
printf("value in main after swapping");
printf("%d %d", a, b);
}
void swap (int *a1, int *b1)
{
Int c;
c=*a1;
*a1=*b1;
*b1=c;
printf("value in function after swapping");
printf("%d %d", *a1, *b1);
}
```

Output:

```
Enter a and b 5 6
value in function after swapping 6 5
value in main after swapping 6 5
```

Recursion:

It is the ability of a function to call itself.

ARRAY

Array:

An array is a finite, ordered set of homogeneous elements.

Types of array:

- One dimensional array
- Two dimensional array

One dimensional array:

Syntax:

<Type of elements> array name [no. of elements in row];

For example:

int num [10];

2bytes 2bytes

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

two dimensional array:

Syntax:

<Type of elements> array name [no. of elements in row][no. of elements in column];

For example:

int num [5][3];

num[0][0]	num[0][1]	num[0][2]	num[0][3]	num[0][4]
num[1][0]	num[1][1]	num[1][2]	num[1][3]	num[1][4]
num[2][0]	num[2][1]	num[2][2]	num[2][3]	num[2][4]

Array of characters:

char name[5]={‘a’, ‘b’, ‘c’, ‘d’, ‘e’};

1 byte	1byte	1byte	1byte	1byte
a	b	c	d	e
name[0]	name[1]	name[2]	name[3]	name[4]

POINTERS

Pointer:

A pointer is a variable that stores the address of another variable.

It is also known as indirection operator.

Pointer Declaration:

Syntax:

Data type *ptr1, *ptr2,……*ptrn;

Example:

int *a;

Indirection operator:

* is known as indirection operator.