

Table of Contents

1. Introduction to C	2
2. Decision and Loops.....	23
3. Functions.....	48
4. Array	63
5. Pointers	79
6. Strings	90
7. Structure	96
8. Files.....	107
9. Example of C programs	114
10. Conclusion	131
11. References.....	131

Introduction to C

C programming is a popular computer programming language which is widely used for system and application software. Despite being fairly old programming language, C programming is widely used because of its efficiency and control. This tutorial is intended for beginners who does not have any prior knowledge or have very little knowledge of computer programming. All basic features of C programming language are included in detail with explanation and output to give you solid platform to understand C programming.

Keywords and Identifiers

Character set

Character set are the set of alphabets, letters and some special characters that are valid in C language.

Alphabets:

Uppercase: A B C X Y Z

Lowercase: a b c x y z

Digits:

0 1 2 3 4 5 6 8 9

Special Characters:

Special Characters in C language

,	<	>	.	_	()	;	\$:	%	[
'	&	{	}	"	^	!	*	/		-	\	

White space Characters:

blank space, new line, horizontal tab, carriage return and form feed

Keywords:

Keywords are the reserved words used in programming. Each keywords has fixed meaning and that cannot be changed by user. For example:

```
int money;
```

Here, int is a keyword that indicates, '*money*' is of type integer.

As, C programming is case sensitive, all keywords must be written in lowercase.

Keywords in C Language

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

Besides these keywords, there are some additional keywords supported by Turbo C.

Additional Keywords for Borland C

asm	far	interrupt	pascal	near	huge	cdecl
-----	-----	-----------	--------	------	------	-------

Identifiers

In C programming, identifiers are names given to C entities, such as variables, functions, structures etc. Identifier are created to give unique name to C entities to identify it during the execution of program. For example:

```
int money;
```

```
int mango_tree;
```

Here, *money* is a identifier which denotes a variable of type integer. Similarly, *mango_tree* is another identifier, which denotes another variable of type integer.

Rules for writing identifier

An identifier can be composed of letters (both uppercase and lowercase letters), digits and underscore '_' only.

The first letter of identifier should be either a letter or an underscore. But, it is discouraged to start an identifier name with an underscore though it is legal. It is because, identifier that starts with underscore can conflict with system names. In such cases, compiler will complain about it. Some system names that start with underscore are *_fileno*, *_iob*, *_wopen* etc.

There is no rule for the length of an identifier. However, the first 31 characters of an identifier are discriminated by the compiler. So, the first 31 letters of two identifiers in a program should be different.

Variables and Constans

Variables

Variables are memory location in computer's memory to store data. To indicate the memory location, each variable should be given a unique name called identifier. Variable names are just the symbolic representation of a memory location. Examples of variable name: *sum*, *car_no*, *count* etc.

```
int num;
```

Here, *num* is a variable of integer type.

Rules for writing variable name in C

Variable name can be composed of letters (both uppercase and lowercase letters), digits and underscore '_' only.

The first letter of a variable should be either a letter or an underscore. But, it is discouraged to start variable name with an underscore though it is legal. It is because, variable name that starts with underscore can conflict with system names and compiler may complain.

There is no rule for the length of length of a variable. However, the first 31 characters of a variable are discriminated by the compiler. So, the first 31 letters of two variables in a program should be different.

In C programming, you have to declare variable before using it in the program.

Constants

Constants are the terms that can't be changed during the execution of a program. For example: 1, 2.5, "Programming is easy." etc. In C, constants can be classified as:

Integer constants

Integer constants are the numeric constants(constant associated with number) without any fractional part or exponential part. There are three types of integer constants in C language: decimal constant(base 10), octal constant(base 8) and hexadecimal constant(base 16) .

Decimal digits: 0 1 2 3 4 5 6 7 8 9

Octal digits: 0 1 2 3 4 5 6 7

Hexadecimal digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F.

For example:

Decimal constants: 0, -9, 22 etc

Octal constants: 021, 077, 033 etc

Hexadecimal constants: 0x7f, 0x2a, 0x521 etc

Notes:

You can use small caps *a, b, c, d, e, f* instead of uppercase letters while writing a hexadecimal constant.

Every octal constant starts with 0 and hexadecimal constant starts with 0x in C programming.

Floating-point constants

Floating point constants are the numeric constants that has either fractional form or exponent form. For example:

-2.0

0.0000234

-0.22E-5

Note: Here, E-5 represents 10^{-5} . Thus, $-0.22E-5 = -0.0000022$.

Character constants

Character constants are the constant which use single quotation around characters. For example: 'a', 'l', 'm', 'F' etc.

Escape Sequences

Sometimes, it is necessary to use newline(enter), tab, quotation mark etc. in the program which either cannot be typed or has special meaning in C programming. In such cases, escape sequence are used. For example: \n is used for newline. The backslash(\) causes "escape" from the normal way the characters are interpreted by the compiler.

Escape Sequences	
Escape Sequences	Character
\b	Backspace
\f	Form feed

Escape Sequences	
Escape Sequences	Character
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character

String constants

String constants are the constants which are enclosed in a pair of double-quote marks. For example:

```
"good"           //string constant
""               //null string constant
"  "            //string constant of six white space
"x"             //string constant having single character.
"Earth is round\n"  //prints string with newline
```

Enumeration constants

Keyword enum is used to declare enumeration types. For example:

```
enum color {yellow, green, black, white};
```

Here, the variable name is color and yellow, green, black and white are the enumeration constants having value 0, 1, 2 and 3 respectively by default.

Data Types:

In C, variable(data) should be declared before it can be used in program. Data types are the keywords, which are used for assigning a type to a variable.

Data types in C

Fundamental Data Types

- Integer types
- Floating Type
- Character types
- Derived Data Types
- Arrays
- Pointers
- Structures
- Enumeration

Syntax for declaration of a variable

```
data_type variable_name;
```

Integer data types

Keyword int is used for declaring the variable with integer type. For example:

```
int var1;
```

Here, *var1* is a variable of type integer.

The size of int is either 2 bytes(In older PC's) or 4 bytes. If you consider an integer having size of 4 byte(equal to 32 bits), it can take 2^{32} distinct states as: -2^{31} , $-2^{31}+1$, ..., -2 , -1 , 0 , 1 , 2 , ..., $2^{31}-2$, $2^{31}-1$

Similarly, int of 2 bytes, it can take 2^{16} distinct states from -2^{15} to $2^{15}-1$. If you try to store larger number than $2^{31}-1$, i.e., +2147483647 and smaller number than -2^{31} , i.e., -2147483648, program will not run correctly.

Floating types

Variables of floating types can hold real values(numbers) such as: 2.34, -9.382 etc. Keywords either float or double is used for declaring floating type variable. For example:

```
float var2;
```

```
double var3;
```

Here, both *var2* and *var3* are floating type variables.

In C, floating values can be represented in exponential form as well. For example:

```
float var3=22.442e2
```

Difference between float and double

Generally the size of float(Single precision float data type) is 4 bytes and that of double(Double precision float data type) is 8 bytes. Floating point variables has a precision of 6 digits whereas the the precision of double is 14 digits.

Note: Precision describes the number of significant decimal places that a floating values carries.

Character types

Keyword char is used for declaring the variable of character type. For example:

```
char var4='h';
```

Here, *var4* is a variable of type character which is storing a character 'h'.

The size of char is 1 byte. The character data type consists of ASCII characters. Each character is given a specific value. For example:

For, 'a', value =97

For, 'b', value=98

For, 'A', value=65

For, '&', value=33

For, '2', value=49

Input/Output

ANSI standard has defined many library functions for input and output in C language. Functions printf() and scanf() are the most commonly used to display out and take input respectively. Let us consider an example:

```
#include <stdio.h>    //This is needed to run printf() function.
void main()
{
    printf("C Programming"); //displays the content inside quotation
}
```

Output:

C Programming

I/O of integers in C

```
#include<stdio.h>
void main()
{
    int c=5;
    printf("Number=%d",c);
}
```

Output:

Number=5

```
#include<stdio.h>
void main()
{
    int c;
    printf("Enter a number\n");
    scanf("%d",&c);
    printf("Number=%d",c);
}
```

Output:

Enter a number

4

Number=4

I/O of float in C

```
#include <stdio.h>
void main(){
    float a;
    printf("Enter value: ");
    scanf("%f",&a);
    printf("Value=%f",a);  //%f is used for floats instead of %d
}
```

Output:

Enter value: 23.45

Value=23.450000

I/O of characters

```
#include <stdio.h>
void main(){
    char var1;
    printf("Enter character: ");
    scanf("%c",&var1);
    printf("You entered %c.",var1);
}
```

Output:

Enter character: g

You entered g.

I/O of ASCII code

```
#include <stdio.h>
void main(){
    char var1;
    printf("Enter character: ");
    scanf("%c",&var1);
    printf("You entered %c.\n",var1);
    /* \n prints the next line(performs work of enter). */
    printf("ASCII value of %d",var1);
}
```

Output:

Enter character:

g

103

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

```
int var1=69;
printf("Character of ASCII value 69: %c",var1);
return 0;
}
```

Output:
Character of ASCII value 69: E

Operators

Operators are the symbol which operates on value or a variable. For example: + is a operator to perform addition.

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

Operators in C programming
<u>Arithmetic Operators</u>
<u>Increment and Decrement Operators</u>
<u>Assignment Operators</u>
<u>Relational Operators</u>
<u>Logical Operators</u>
<u>Conditional Operators</u>
<u>Bitwise Operators</u>
<u>Special Operators</u>

Arithmetic Operators

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division(modulo division)

Example of working of arithmetic operators.

```
/* Program to demonstrate the working of arithmetic operators in C. */
#include <stdio.h>
int main(){
    int a=9,b=4,c;
    c=a+b;
    printf("a+b=%d\n",c);
    c=a-b;
    printf("a-b=%d\n",c);
    c=a*b;
    printf("a*b=%d\n",c);
    c=a/b;
    printf("a/b=%d\n",c);
    c=a%b;
    printf("Remainder when a divided by b=%d\n",c);
    return 0;
}
a+b=13
a-b=5
a*b=36
a/b=2
Remainder when a divided by b=1
```

Explanation

Here, the operators `+`, `-` and `*` performed normally as you expected. In normal calculation, $9/4$ equals to 2.25. But, the output is 2 in this program. It is because, `a` and `b` are both integers. So, the output is also integer and the compiler neglects the term after decimal point and shows answer 2 instead of 2.25. And, finally `a%b` is 1, i.e., when `a=9` is divided by `b=4`, remainder is 1.

Suppose `a=5.0`, `b=2.0`, `c=5` and `d=2`

In C programming,

`a/b=2.5`

`a/d=2.5`

`c/b=2.5`

`c/d=2`

Note: `%` operator can only be used with integers.

Increment and decrement 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` and `b=10`

`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 only increment it. This can be demonstrated by an example:

```
#include <stdio.h>
int main(){
    int c=2,d=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

The most common assignment operator is `=`. This operator assigns the value in 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
<code>=</code>	<code>a=b</code>	<code>a=b</code>
<code>+=</code>	<code>a+=b</code>	<code>a=a+b</code>
<code>-=</code>	<code>a-=b</code>	<code>a=a-b</code>
<code>*=</code>	<code>a*=b</code>	<code>a=a*b</code>

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

Relational Operator

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

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

Operator	Meaning of Operator	Example
&&	Logial 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.

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

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

A bitwise operator works on each bit of data. Bitwise operators are used in bit level programming.

Operators	Meaning of operators
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Bitwise operator is advance topic in programming .

Other Operators

Comma Operator

Comma operators are used to link related expressions together. For example:

```
int a,c=5,d;
```

The sizeof operator

It is a unary operator which is used in finding the size of data type, constant, arrays, structure etc.

For example:

```
#include <stdio.h>
int main(){
    int a;
    float b;
    double c;
    char d;
    printf("Size of int=%d bytes\n",sizeof(a));
    printf("Size of float=%d bytes\n",sizeof(b));
    printf("Size of double=%d bytes\n",sizeof(c));
    printf("Size of char=%d byte\n",sizeof(d));
    return 0;
}
```

Output

Size of int=4 bytes

Size of float=4 bytes

Size of double=8 bytes

Size of char=1 byte

Conditional operators (?:)

Conditional operators are used in decision making in C programming, i.e, executes different statements according to test condition whether it is either true or false.

Syntax of conditional operators

conditional_expression?expression1:expression2

If the test condition is true, expression1 is returned and if false expression2 is returned.

Example of conditional operator.

```
#include <stdio.h>
int main(){
    char feb;
    int days;
    printf("Enter 1 if the year is leap year otherwise enter 0: ");
    scanf("%c",&feb);
    days=(feb=='1')?29:28;
    /*If test condition (feb=='1') is true, days will be equal to 29. */
    /*If test condition (feb=='1') is false, days will be equal to 28. */
    printf("Number of days in February = %d",days);
    return 0;
}
```

Output

Enter 1 if the year is leap year otherwise enter n: 1

Number of days in February = 29

Examples:

C Program to add two integers

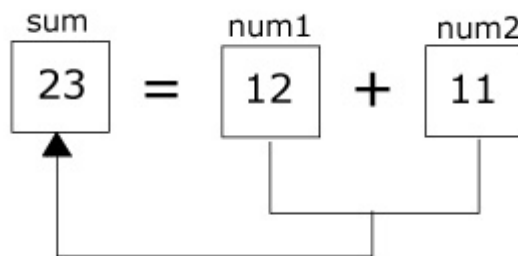
In this program, user is asked to enter two integers and this program will add these two integers and display it.

Source Code

```
/*C programming source code to add and display the sum of two integers entered by user */  
  
#include <stdio.h>  
int main( )  
{  
    int num1, num2, sum;  
    printf("Enter two integers: ");  
    scanf("%d %d",&num1,&num2); /* Stores the two integer entered by user in variable num1  
and num2 */  
  
    sum=num1+num2; /* Performs addition and stores it in variable sum */  
    printf("Sum: %d",sum); /* Displays sum */  
    return 0;  
}  
Output  
Enter two integers: 12  
11  
Sum: 23
```

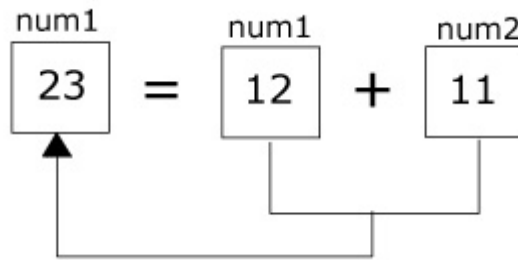
Explanation

In this program, user is asked to enter two integers. The two integers entered by user will be stored in variables *num1* and *num2* respectively. This is done using `scanf()` function. Then, `+` operator is used for adding variables *num1* and *num2* and this value is assigned to variable *sum*.



Finally, the sum is displayed and program is terminated.

There are three variables used for performing this task but, you can perform this same task using 2 variables only. It can be done by assigning the sum of *num1* and *num2* to one of these variables as shown in figure below.



/* C programming source code to add and display the sum of two integers entered by user using two variables only. */

```
#include <stdio.h>
int main( )
{
    int num1, num2, sum;
    printf("Enter a two integers: ");
    scanf("%d %d",&num1,&num2);
    num1=num1+num2; /* Adds variables num1 and num2 and stores it in num1 */
    printf("Sum: %d",num1); /* Displays value of num1 */
    return 0;
}
```

This source code calculates the sum of two integers and displays it but, this program uses only two variables.

C program to multiply two floating point numbers.

In this program, user is asked to enter two floating point numbers and this program will multiply these two numbers and display it.

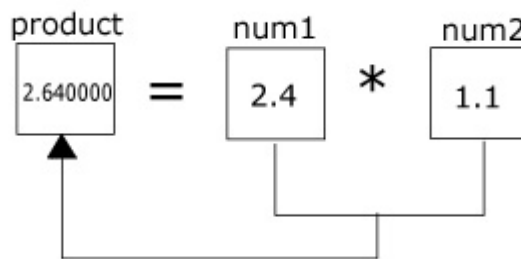
Source Code

```
/*C program to multiply and display the product of two floating point numbers entered by user.
*/

#include <stdio.h>
int main( )
{
    float num1, num2, product;
    printf("Enter two numbers: ");
    scanf("%f %f",&num1,&num2);
    /* Stores the two floating point numbers entered by user in variable num1 and num2
    respectively */
    product=num1*num2; /* Performs multiplication and stores it */
    printf("Product: %f",product);
    return 0;
}
Output
Enter two numbers: 2.4
1.1
Sum: 2.640000
```

Explanation

In this program, user is asked to enter two floating point numbers. These two numbers entered by user will be stored in variables *num1* and *num2* respectively. This is done using `scanf()` function. Then, `*` operator is used for multiplying variables and this value is stored in variable *product*.



Then, the product is displayed and program is terminated.

C Program to swap two numbers:

This program asks user to enter two numbers and this program will swap the value of these two numbers.

Source Code to Swap Two Numbers

```
#include <stdio.h>
int main(){
    float a, b, temp;
    printf("Enter value of a: ");
    scanf("%f",&a);
    printf("Enter value of b: ");
    scanf("%f",&b);
    temp = a; /* Value of a is stored in variable temp */
    a = b;    /* Value of b is stored in variable a */
    b = temp; /* Value of temp(which contains initial value of a) is stored in variable b*/
    printf("\nAfter swapping, value of a = %.2f\n", a);
    printf("After swapping, value of b = %.2f", b);
    return 0;
}
```

Output

Enter value of a: 1.20
Enter value of b: 2.45

After swapping, value of a = 2.45
After swapping, value of b = 1.2

Decision and Loops

The if, if...else and nested if...else statement are used to make one-time decisions in C Programming, that is, to execute some code/s and ignore some code/s depending upon the test expression.

C if Statement

```
if (test expression) {
    statement/s to be executed if test expression is true;
```

}

The if statement checks whether the text expression inside parenthesis () is true or not. If the test expression is true, statement/s inside the body of if statement is executed but if test is false, statement/s inside body of if is ignored.

Flowchart of if statement

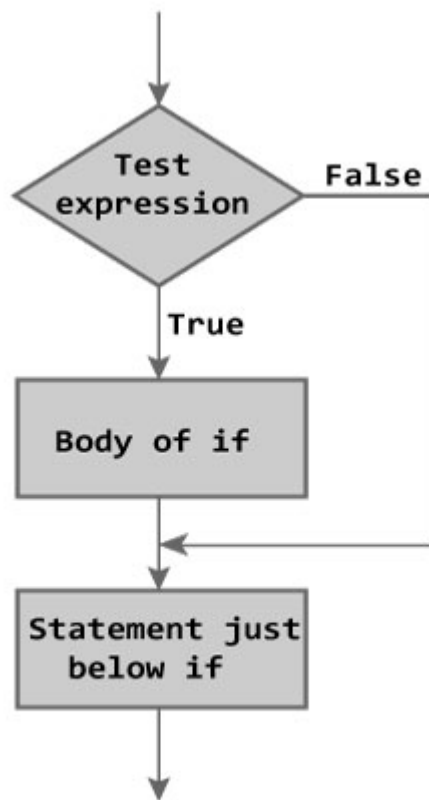


Figure: Flowchart of if Statement

Example 1: C if statement

Write a C program to print the number entered by user only if the number entered is negative.

```
#include <stdio.h>
int main(){
    int num;
    printf("Enter a number to check.\n");
    scanf("%d",&num);
    if(num<0) { /* checking whether number is less than 0 or not. */
        printf("Number = %d\n",num);
    }
    /*If test condition is true, statement above will be executed, otherwise it will not be executed */
    printf("The if statement in C programming is easy.");
    return 0;
}
Output 1
Enter a number to check.
-2
Number = -2
The if statement in C programming is easy.
When user enters -2 then, the test expression (num<0) becomes true. Hence, Number = -2 is displayed in the screen.
Output 2
Enter a number to check.
5
```

The if statement in C programming is easy.

When the user enters 5 then, the test expression (num<0) becomes false. So, the statement/s inside body of if is skipped and only the statement below it is executed.

C if...else statement

The if...else statement is used if the programmer wants to execute some statement/s when the test expression is true and execute some other statement/s if the test expression is false.

Syntax of if...else

```
if (test expression) {  
    statements to be executed if test expression is true;  
}  
else {  
    statements to be executed if test expression is false;  
}
```

Flowchart of if...else statement

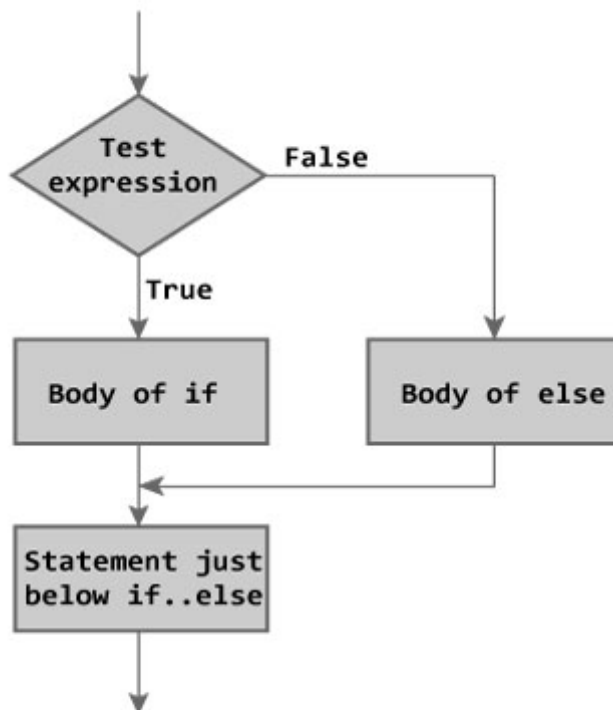


Figure: Flowchart of if...else Statement

Example 2: C if...else statement

Write a C program to check whether a number entered by user is even or odd

```
#include <stdio.h>
int main() {
    int num;
    printf("Enter a number you want to check.\n");
    scanf("%d",&num);
    if((num%2)==0)    //checking whether remainder is 0 or not.
        printf("%d is even.",num);
    else
        printf("%d is odd.",num);
    return 0;
}
```

Output 1

Enter a number you want to check.

25

25 is odd.

Output 2

Enter a number you want to check.

2

2 is even.

Nested if...else statement (if...elseif....else Statement)

The nested if...else statement is used when program requires more than one test expression.

Syntax of nested if...else statement.

```
if (test expression1) {
    statement/s to be executed if test expression1 is true;
}
else if (test expression2) {
    statement/s to be executed if test expression1 is false and 2 is true;
}
else if (test expression 3) {
    statement/s to be executed if text expression1 and 2 are false and 3 is true;
}
.
else {
    statements to be executed if all test expressions are false;
}
```

How nested if...else works?

The nested if...else statement has more than one test expression. If the first test expression is true, it executes the code inside the braces{ } just below it. But if the first test expression is false, it checks the second test expression. If the second test expression is true, it executes the statement/s inside the braces{ } just below it. This process continues. If all the test expression are false, code/s inside else is executed and the control of program jumps below the nested if...else

The ANSI standard specifies that 15 levels of nesting may be continued.

Example 3: C nested if else statement

Write a C program to relate two integers entered by user using = or > or < sign.

```
#include <stdio.h>
int main(){
    int numb1, numb2;
    printf("Enter two integers to check.\n");
    scanf("%d %d",&numb1,&numb2);
    if(numb1==numb2) //checking whether two integers are equal.
        printf("Result: %d = %d",numb1,numb2);
    else
        if(numb1>numb2) //checking whether numb1 is greater than numb2.
            printf("Result: %d > %d",numb1,numb2);
        else
            printf("Result: %d > %d",numb2,numb1);
    return 0;
}
```

Output 1

Enter two integers to check.

5

3

Result: 5 > 3

Output 2

Enter two integers to check.

-4

-4

Result: -4 = -4

C Programming Loops

Loops cause program to execute the certain block of code repeatedly until test condition is false.

Loops are used in performing repetitive task in programming. Consider these scenarios:

You want to execute some code/s 100 times.

You want to execute some code/s certain number of times depending upon input from user.

These types of task can be solved in programming using loops.

There are 3 types of loops in C programming:

- for loop
- while loop
- do...while loop

for Loop Syntax

```
for(initialization statement; test expression; update statement) {  
    code/s to be executed;  
}
```

How for loop works in C programming?

The initialization statement is executed only once at the beginning of the for loop. Then the test expression is checked by the program. If the test expression is false, for loop is terminated. But if test expression is true then the code/s inside body of for loop is executed and then update expression is updated. This process repeats until test expression is false.

This flowchart describes the working of for loop in C programming.

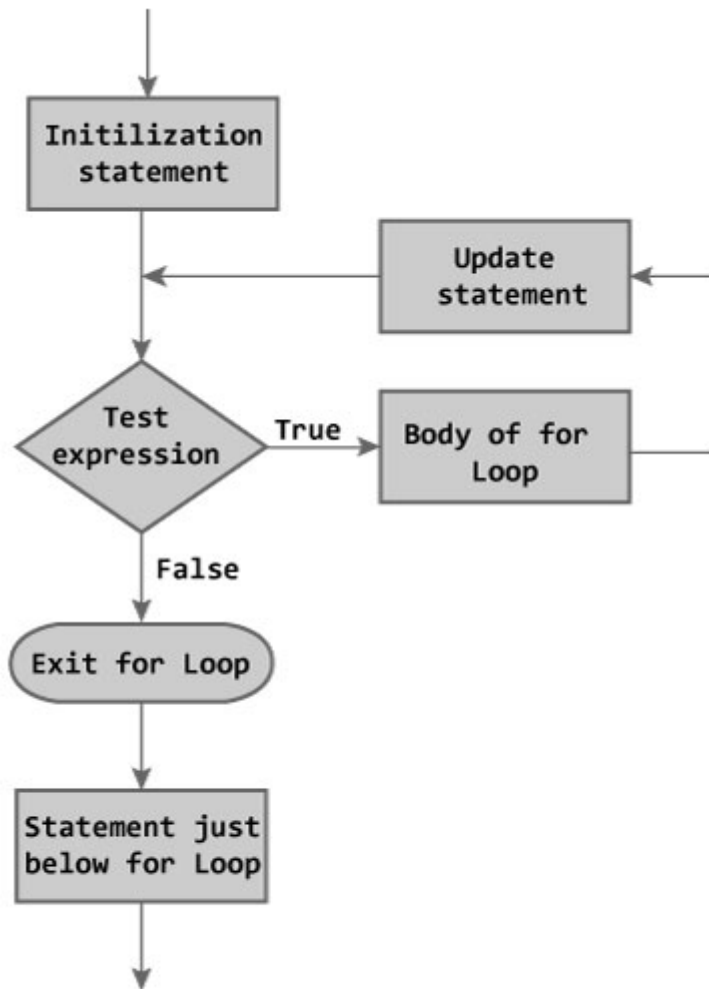


Figure: Flowchart of for Loop

for loop example

Write a program to find the sum of first n natural numbers where n is entered by user. Note: 1,2,3... are called natural numbers.

```

#include <stdio.h>
int main(){
    int n, count, sum=0;
    printf("Enter the value of n.\n");
    scanf("%d",&n);
    for(count=1;count<=n;++count) //for loop terminates if count>n
    {

```

```
        sum+=count; /* this statement is equivalent to sum=sum+count */
    }
    printf("Sum=%d",sum);
    return 0;
}
```

Output

```
Enter the value of n.
19
Sum=190
```

In this program, the user is asked to enter the value of n . Suppose you entered 19 then, `count` is initialized to 1 at first. Then, the test expression in the for loop, i.e., $(count \leq n)$ becomes true. So, the code in the body of for loop is executed which makes `sum` to 1. Then, the expression `++count` is executed and again the test expression is checked, which becomes true. Again, the body of for loop is executed which makes `sum` to 3 and this process continues. When `count` is 20, the test condition becomes false and the for loop is terminated.

C programming while loops

Loops causes program to execute the certain block of code repeatedly until some conditions are satisfied, i.e., loops are used in performing repetitive work in programming.

Suppose you want to execute some code/s 10 times. You can perform it by writing that code/s only one time and repeat the execution 10 times using loop.

Syntax of while loop

```
while (test expression) {
    statement/s to be executed.
}
```

The while loop checks whether the test expression is true or not. If it is true, code/s inside the body of while loop is executed, that is, code/s inside the braces `{ }` are executed. Then again the test expression is checked whether test expression is true or not. This process continues until the test expression becomes false.

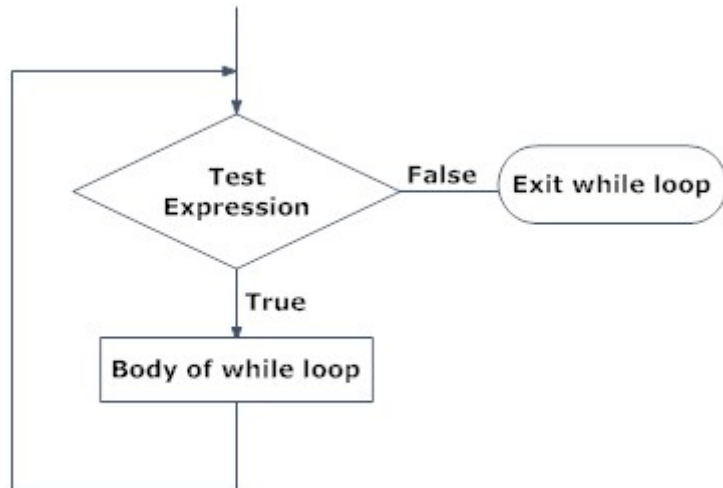


Figure: Flowchart of while loop

Example of while loop

Write a C program to find the factorial of a number, where the number is entered by user. (Hints: factorial of $n = 1*2*3*...*n$)

```

/*C program to demonstrate the working of while loop*/
#include <stdio.h>
int main(){
    int number,factorial;
    printf("Enter a number.\n");
    scanf("%d",&number);
    factorial=1;
    while (number>0){    /* while loop continues until test condition number>0 is true */
        factorial=factorial*number;
        --number;
    }
    printf("Factorial=%d",factorial);
    return 0;
}
Output
Enter a number.
5
Factorial=120
  
```


do...while loop

In C, do...while loop is very similar to while loop. Only difference between these two loops is that, in while loops, test expression is checked at first but, in do...while loop code is executed at first then the condition is checked. So, the code are executed at least once in do...while loops.

Syntax of do...while loops

```
do {  
    some code/s;  
}  
while (test expression);
```

At first codes inside body of do is executed. Then, the test expression is checked. If it is true, code/s inside body of do are executed again and the process continues until test expression becomes false(zero).

Notice, there is semicolon in the end of while (); in do...while loop.

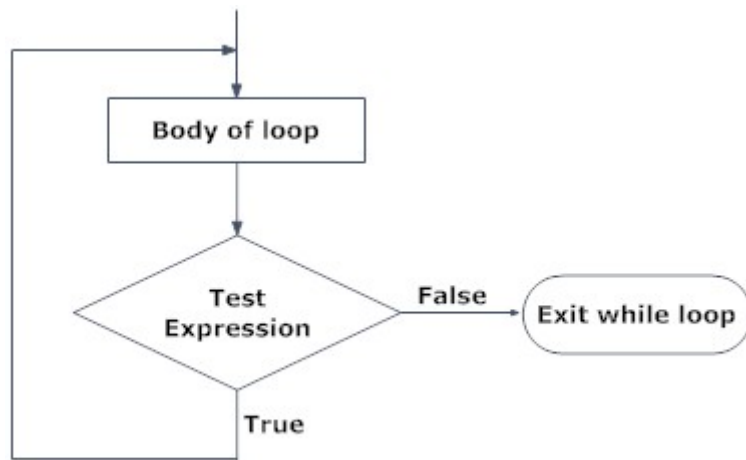


Figure: Flowchart of do...while loop

Example of do...while loop

Write a C program to add all the numbers entered by a user until user enters 0.

```
/*C program to demonstrate the working of do...while statement*/  
#include <stdio.h>  
int main(){  
    int sum=0,num;  
    do        /* Codes inside the body of do...while loops are at least executed once. */  
    {
```

```
printf("Enter a number\n");
scanf("%d",&num);
sum+=num;
}
while(num!=0);
printf("sum=%d",sum);
return 0;
}
```

Output

```
Enter a number
3
Enter a number
-2
Enter a number
0
sum=1
```

In this C program, user is asked a number and it is added with *sum*. Then, only the test condition in the do...while loop is checked. If the test condition is true,i.e, *num* is not equal to 0, the body of do...while loop is again executed until *num* equals to zero.

Break and continue statement

There are two statements built in C programming, `break`; and `continue`; to alter the normal flow of a program. Loops perform a set of repetitive task until text expression becomes false but it is sometimes desirable to skip some statement/s inside loop or terminate the loop immediately without checking the test expression. In such cases, `break` and `continue` statements are used. The `break`; statement is also used in switch statement to exit switch statement.

break Statement

In C programming, `break` is used in terminating the loop immediately after it is encountered. The `break` statement is used with conditional if statement.

Syntax of break statement

```
break;
```

The `break` statement can be used in terminating all three loops for, while and do...while loops.

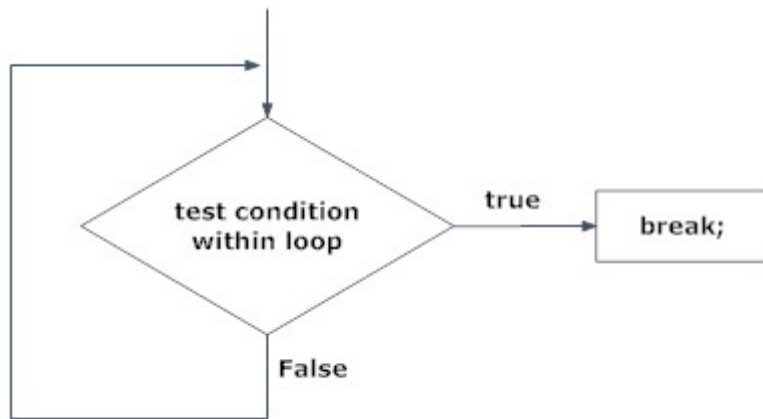


Figure: Flowchart of break statement

The figure below explains the working of break statement in all three type of loops.

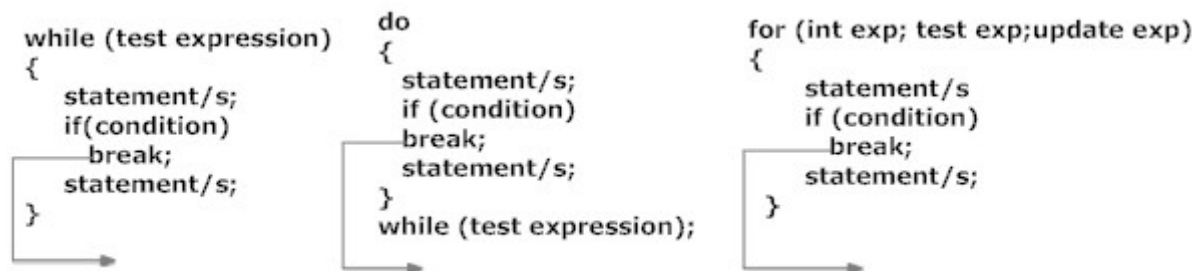


Fig: Working of break statement in different loops

Example of break statement

Write a C program to find average of maximum of n positive numbers entered by user. But, if the input is negative, display the average(excluding the average of negative input) and end the program.

```

/* C program to demonstrate the working of break statement by terminating a loop, if user inputs
negative number*/
# include <stdio.h>
int main(){
    float num,average,sum;
    int i,n;
    printf("Maximum no. of inputs\n");
    scanf("%d",&n);
    for(i=1;i<=n;++i){
        printf("Enter n%d: ",i);
        scanf("%f",&num);
        if(num<0.0)
            break;           //for loop breaks if num<0.0
        sum=sum+num;
    }
}
  
```

```

average=sum/(i-1);
printf("Average=%.2f",average);
return 0;
}

```

Output

Maximum no. of inputs

4

Enter n1: 1.5

Enter n2: 12.5

Enter n3: 7.2

Enter n4: -1

Average=7.07

In this program, when the user inputs number less than zero, the loop is terminated using break statement with executing the statement below it i.e., without executing `sum=sum+num`.

In C, break statements are also used in switch...case statement.

continue Statement

It is sometimes desirable to skip some statements inside the loop. In such cases, continue statements are used.

Syntax of continue Statement

`continue;`

Just like break, continue is also used with conditional if statement.

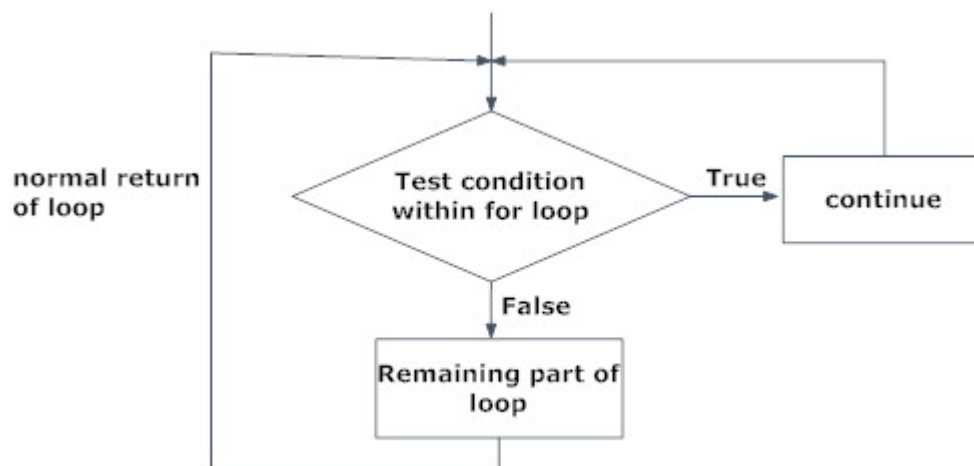


Fig: Flowchart of continue statement

For better understanding of how continue statements works in C programming. Analyze the figure below which bypasses some code/s inside loops using continue statement.

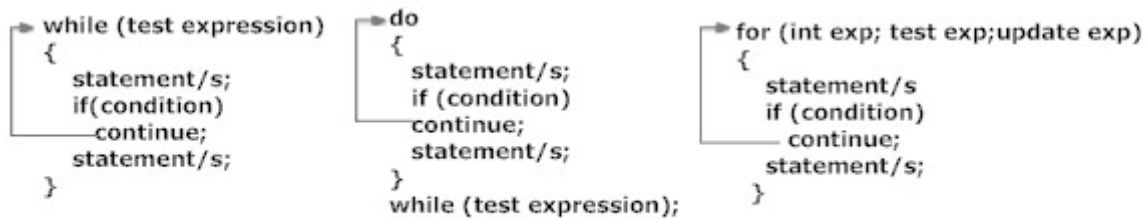


Fig: Working of continue statement in different loops

Example of continue statement

Write a C program to find the product of 4 integers entered by a user. If user enters 0 skip it.

```

//program to demonstrate the working of continue statement in C programming
# include <stdio.h>
int main(){
    int i,num,product;
    for(i=1,product=1;i<=4;++i){
        printf("Enter num%d:",i);
        scanf("%d",&num);
        if(num==0)
            continue; / *In this program, when num equals to zero, it skips the statement
product*=num and continue the loop. */
        product*=num;
    }
    printf("product=%d",product);
    return 0;
}

```

Output

```

Enter num1:3
Enter num2:0
Enter num3:-5
Enter num4:2
product=-30

```

Switch case

Decision making are needed when, the program encounters the situation to choose a particular statement among many statements. If a programmer has to choose one among many alternatives if...else can be used but, this makes programming logic complex. This type of problem can be handled in C programming using switch...case statement.

Syntax of switch...case

```
switch (expression)
{
case constant1:
    codes to be executed if expression equals to constant1;
    break;
case constant2:
    codes to be executed if expression equals to constant3;
    break;
.
.
.
default:
    codes to be executed if expression doesn't match to any cases;
}
```

In switch...case, expression is either an integer or a character. If the value of switch expression matches any of the constant in case, the relevant codes are executed and control moves out of the switch...case statement. If the expression doesn't matches any of the constant in case, then the default statement is executed.

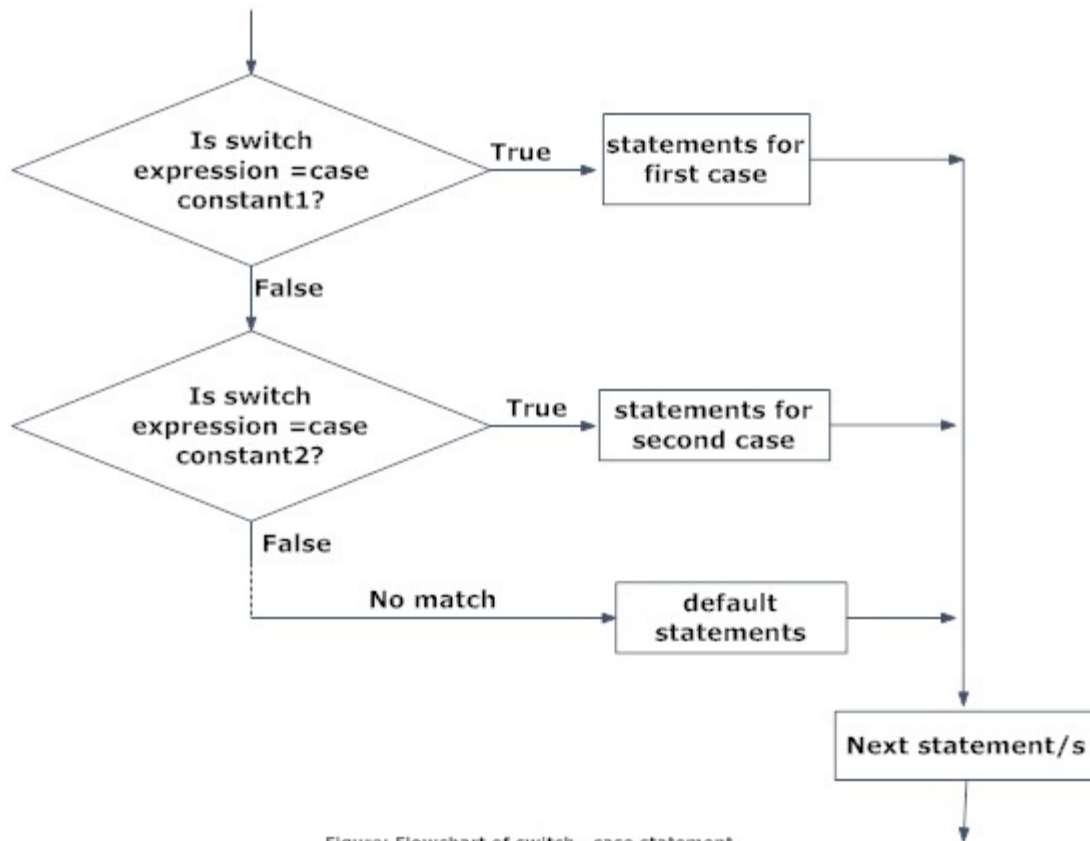


Figure: Flowchart of switch...case statement

Example of switch...case statement

Write a program that asks user an arithmetic operator('+','-','*' or '/') and two operands and perform the corresponding calculation on the operands.

```

/* C program to demonstrate the working of switch...case statement */
/* Program to create a simple calculator for addition, subtraction, multiplication and division */
# include <stdio.h>
int main()
{
    char o;
    float num1,num2;
    printf("Enter operator either + or - or * or divide : ");
    scanf("%c",&o);
    printf("Enter two operands: ");
    scanf("%f%f",&num1,&num2);
    switch(o) {
        case '+':
            printf("%.1f + %.1f = %.1f",num1, num2, num1+num2);
            break;
        case '-':
            printf("%.1f - %.1f = %.1f",num1, num2, num1-num2);

```

```

        break;
    case '*':
        printf("%.1f * %.1f = %.1f", num1, num2, num1 * num2);
        break;
    case '/':
        printf("%.1f / %.1f = %.1f", num1, num2, num1 / num2);
        break;
    default:
        /* If operator is other than +, -, * or /, error message is shown */
        printf("Error! operator is not correct");
        break;
    }
    return 0;
}

```

Output

Enter operator either + or - or * or divide : *

Enter two operands: 2.3

4.5

2.3 * 4.5 = 10.3

Notice break statement at the end of each case, which cause switch...case statement to exit. If break statement are not used, all statements below that case statement are also executed.

Goto statement

In C programming, goto statement is used for altering the normal sequence of program execution by transferring control to some other part of the program.

Syntax of goto statement

```

goto label;
.....
.....
.....
label:
statement;

```

In this syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code/s after it.

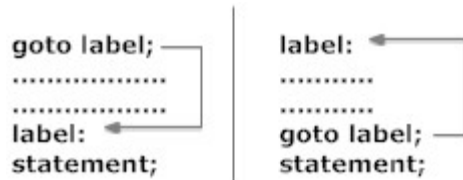


Figure: Working of goto statement

Example of goto statement

```

/* C program to demonstrate the working of goto statement.*/
# include <stdio.h>
int main(){
    float num,average,sum;
    int i,n;
    printf("Maximum no. of inputs: ");
    scanf("%d",&n);
    for(i=1;i<=n;++i){
        printf("Enter n%d: ",i);
        scanf("%f",&num);
        if(num<0.0)
            goto jump;      /* control of the program jumps to label jump */
        sum=sum+num;
    }
jump:
    average=sum/(i-1);
    printf("Average: %.2f",average);
    return 0;
}

```

Output

```

Maximum no. of inputs: 4
Enter n1: 1.5
Enter n2: 12.5
Enter n3: 7.2
Enter n4: -1
Average: 7.07

```

Though goto statement is included in ANSI standard of C, use of goto statement should be reduced as much as possible in a program.

Reasons to avoid goto statement

Though, using goto statement give power to jump to any part of program, using goto statement makes the logic of the program complex and tangled. In modern programming, goto statement is considered a harmful construct and a bad programming practice.

The goto statement can be replaced in most of C program with the use of break and continue statements. In fact, any program in C programming can be perfectly written without the use of goto statement. All programmer should try to avoid goto statement as possible as they can.

Decision making and loop examples:

Source Code to Check Whether a Number is Even or Odd

```
/*C program to check whether a number entered by user is even or odd. */  
  
#include <stdio.h>  
int main(){  
    int num;  
    printf("Enter an integer you want to check: ");  
    scanf("%d",&num);  
    if((num%2)==0)    /* Checking whether remainder is 0 or not. */  
        printf("%d is even.",num);  
    else  
        printf("%d is odd.",num);  
    return 0;  
}
```

Output 1

Enter an integer you want to check: 25
25 is odd.

Output 2

Enter an integer you want to check: 12
12 is even.

In this program, user is asked to enter an integer which is stored in variable *num*. Then, the remainder is found when that number is divided by 2 and checked whether remainder is 0 or not. If remainder is 0 then, that number is even otherwise that number is odd. This task is performed using if...else statement in C programming and the result is displayed accordingly.

This program also can be solved using conditional operator [?:] which is the shorthand notation for if...else statement.

```
/* C program to check whether an integer is odd or even using conditional operator */  
  
#include <stdio.h>  
int main(){  
    int num;  
    printf("Enter an integer you want to check: ");  
    scanf("%d",&num);  
    ((num%2)==0) ? printf("%d is even.",num) : printf("%d is odd.",num);  
    return 0;  
}
```

Source Code to Check Whether a Character is Vowel or consonant

```
/* C program to check whether an a Character is Vowel or consonant */  
  
#include <stdio.h>  
int main(){  
    char c;  
    printf("Enter an alphabet: ");  
    scanf("%c",&c);  
    if(c=='a' || c=='A' || c=='e' || c=='E' || c=='i' || c=='I' || c=='o' || c=='O' || c=='u' || c=='U')  
        printf("%c is a vowel.",c);  
    else  
        printf("%c is a consonant.",c);  
    return 0;  
}
```

Output 1

Enter an alphabet: i

i is a vowel.

Output 2

Enter an alphabet: G

G is a consonant.

In this program, user is asked to enter a character which is stored in variable c. Then, this character is checked, whether it is any one of these ten characters a, A, e, E, i, I, o, O, u and U using logical OR operator ||. If that character is any one of these ten characters, that alphabet is a vowel if not that alphabet is a consonant.

```
/* C Program to find largest number using nested if...else statement */
```

```
#include <stdio.h>
int main(){
    float a, b, c;
    printf("Enter three numbers: ");
    scanf("%f %f %f", &a, &b, &c);
    if(a>=b && a>=c)
        printf("Largest number = %.2f", a);
    else if(b>=a && b>=c)
        printf("Largest number = %.2f", b);
    else
        printf("Largest number = %.2f", c);
    return 0;
}
```

Output:

```
Enter three numbers: 12.2
13.452
10.193
Largest number = 13.45
```

Source Code to Find Roots of Quadratic Equation

```
/* Program to find roots of a quadratic equation when coefficients are entered by user. */
/* Library function sqrt() computes the square root. */
```

```
#include <stdio.h>
#include <math.h> /* This is needed to use sqrt() function.*/
int main()
{
    float a, b, c, determinant, r1, r2, real, imag;
    printf("Enter coefficients a, b and c: ");
    scanf("%f%f%f", &a, &b, &c);
    determinant=b*b-4*a*c;
    if (determinant>0)
    {
        r1= (-b+sqrt(determinant))/(2*a);
        r2= (-b-sqrt(determinant))/(2*a);
        printf("Roots are: %.2f and %.2f", r1, r2);
    }
    else if (determinant==0)
    {
        r1 = r2 = -b/(2*a);
        printf("Roots are: %.2f and %.2f", r1, r2);
    }
}
```

```

}
else
{
    real= -b/(2*a);
    imag = sqrt(-determinant)/(2*a);
    printf("Roots are: %.2f+%.2fi and %.2f-%.2fi", real, imag, real, imag);
}
return 0;
}

```

Output 1

Enter coefficients a, b and c: 2.3

4

5.6

Roots are: -0.87+1.30i and -0.87-1.30i

Output 2

Enter coefficients a, b and c: 4

1

0

Roots are: 0.00 and -0.25

Leap year

All years which are perfectly divisible by 4 are leap years except for century years(years ending with 00) which is a leap year only if it is perfectly divisible by 400. For example: 2012, 2004, 1968 etc are leap year but, 1971, 2006 etc are not leap year. Similarly, 1200, 1600, 2000, 2400 are leap years but, 1700, 1800, 1900 etc are not.

This program asks user to enter a year and this program checks whether that year is leap year or not.

Source Code to Check Leap Year

```

/* C program to check whether a year is leap year or not using if else statement.*/

#include <stdio.h>
int main() {
    int year;
    printf("Enter a year: ");
    scanf("%d",&year);
}

```

```

if(year%4 == 0)
{
    if( year%100 == 0) /* Checking for a century year */
    {
        if ( year%400 == 0)
            printf("%d is a leap year.", year);
        else
            printf("%d is not a leap year.", year);
    }
    else
        printf("%d is a leap year.", year );
}
else
    printf("%d is not a leap year.", year);
return 0;
}

```

Output 1:

Enter year: 2000
2000 is a leap year.

Output 2:

Enter year: 1900
1900 is not a leap year.

C program to check whether a number is positive or negative.

```

/* C programming code to check whether a number is negative or positive or zero using nested
if...else statement. */
#include <stdio.h>
int main()
{
    float num;
    printf("Enter a number: ");
    scanf("%f",&num);
    if (num<0) /* Checking whether num is less than 0*/
        printf("%.2f is negative.",num);
    else if (num>0) /* Checking whether num is greater than zero*/
        printf("%.2f is positive.",num);
    else
        printf("You entered zero.");
    return 0;
}

```

Output 1:

Enter a number: 12.3
12.30 is positive.

Output 1:
Enter a number: -12.3
-12.30 is negative.

C program to check whether a character is alphabet or not.

```
/* C programming code to check whether a character is alphabet or not.*/
```

```
#include <stdio.h>
int main()
{
    char c;
    printf("Enter a character: ");
    scanf("%c",&c);
    if( (c>='a' && c<='z') || (c>='A' && c<='Z'))
        printf("%c is an alphabet.",c);
    else
        printf("%c is not an alphabet.",c);
    return 0;
}
```

Output 1:
Enter a character: *
* is not an alphabet

Output 1:
Enter a character: K
K is an alphabet

Functions

Function in programming is a segment that groups a number of program statements to perform specific task.

A C program has at least one function `main()`. Without `main()` function, there is technically no C program.

Types of C functions

Basically, there are two types of functions in C on basis of whether it is defined by user or not.

Library function

User defined function

Library function

Library functions are the in-built function in C programming system. For example:

`main()`

- The execution of every C program starts from this `main()` function.

`printf()`

- `printf()` is used for displaying output in C.

`scanf()`

- `scanf()` is used for taking input in C.

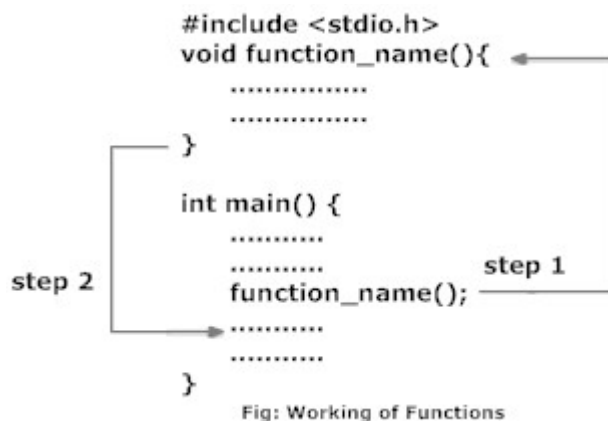
User defined function

C provides programmer to define their own function according to their requirement known as user defined functions. Suppose, a programmer wants to find factorial of a number and check whether it is prime or not in same program. Then, he/she can create two separate user-defined functions in that program: one for finding factorial and other for checking whether it is prime or not.

How user-defined function works in C Programming?

```
#include <stdio.h>
void function_name(){
.....
.....
}
int main(){
.....
.....
function_name();
.....
.....
}
```

As mentioned earlier, every C program begins from main() and program starts executing the codes inside main() function. When the control of program reaches to function_name() inside main() function. The control of program jumps to void function_name() and executes the codes inside it. When, all the codes inside that user-defined function are executed, control of the program jumps to the statement just after function_name() from where it is called. Analyze the figure below for understanding the concept of function in C programming. Visit this page to learn in detail about user-defined functions.



Remember, the function name is an identifier and should be unique.

Advantages of user defined functions

User defined functions helps to decompose the large program into small segments which makes programmer easy to understand, maintain and debug.

If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.

Programmer working on large project can divide the workload by making different functions.

User defined function

Example of user-defined function

Write a C program to add two integers. Make a function add to add integers and display sum in main()function.

```
/*Program to demonstrate the working of user defined function*/
#include <stdio.h>
int add(int a, int b);      //function prototype(declaration)
int main(){
    int num1,num2,sum;
    printf("Enter two number to add\n");
    scanf("%d %d",&num1,&num2);
    sum=add(num1,num2);     //function call
    printf("sum=%d",sum);
    return 0;
}
int add(int a,int b)        //function declarator
{
/* Start of function definition. */
    int add;
    add=a+b;
    return add;             //return statement of function
/* End of function definition. */
}
```

Function prototype(declaration):

Every function in C programming should be declared before they are used. These type of declaration are also called function prototype. Function prototype gives compiler information about function name, type of arguments to be passed and return type.

Syntax of function prototype

`return_type function_name(type(1) argument(1),...,type(n) argument(n));`

In the above example, `int add(int a, int b);` is a function prototype which provides following information to the compiler:

name of the function is `add()`

return type of the function is `int`.

two arguments of type `int` are passed to function.

Function prototype are not needed if user-definition function is written before `main()` function.

Function call

Control of the program cannot be transferred to user-defined function unless it is called (invoked).

Syntax of function call

`function_name(argument(1),...,argument(n));`

In the above example, function call is made using statement `add(num1,num2);` from `main()`. This make the control of program jump from that statement to function definition and executes the codes inside that function.

Function definition

Function definition contains programming codes to perform specific task.

Syntax of function definition

`return_type function_name(type(1) argument(1),...,type(n) argument(n))`

`{`

`//body of function`

`}`

Function definition has two major components:

1. Function declarator

Function declarator is the first line of function definition. When a function is invoked from calling function, control of the program is transferred to function declarator or called function.

Syntax of function declarator

return_type function_name(type(1) argument(1),...,type(n) argument(n))

Syntax of function declaration and declarator are almost same except, there is no semicolon at the end of declarator and function declarator is followed by function body.

In above example, `int add(int a,int b)` in line 12 is a function declarator.

2. Function body

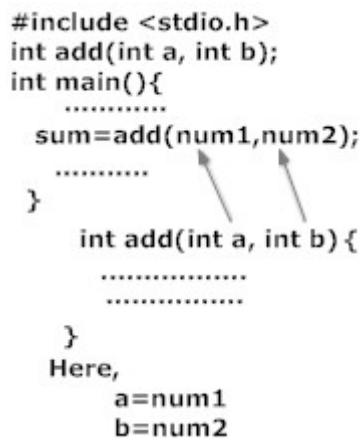
Function declarator is followed by body of function which is composed of statements.

Passing arguments to functions

In programming, argument/parameter is a piece of data(constant or variable) passed from a program to the function.

In above example two variable, `num1` and `num2` are passed to function during function call and these arguments are accepted by arguments `a` and `b` in function definition.

```
#include <stdio.h>
int add(int a, int b);
int main(){
    .....
    sum=add(num1,num2);
    .....
}
    int add(int a, int b){
        .....
        .....
    }
Here,
    a=num1
    b=num2
```

A diagram illustrating argument passing. In the `main` function, the call `sum=add(num1,num2);` has two arrows pointing from `num1` and `num2` to the parameters `a` and `b` in the `int add(int a, int b){` function definition. Below the function definition, the text 'Here, a=num1 b=num2' indicates the mapping of arguments to parameters.

Arguments that are passed in function call and arguments that are accepted in function definition should have same data type. For example:

If argument *num1* was of int type and *num2* was of float type then, argument variable *a* should be of type int and *b* should be of type float, i.e., type of argument during function call and function definition should be same.

A function can be called with or without an argument.

Return Statement

Return statement is used for returning a value from function definition to calling function.

Syntax of return statement

```
return (expression);  
OR  
return;  
For example:  
return;  
return a;  
return (a+b);
```

In above example, value of variable *add* in *add()* function is returned and that value is stored in variable *sum* in *main()* function. The data type of expression in return statement should also match the return type of function.

```
#include <stdio.h>  
int add(int a,int b);  
int main(){  
.....  
sum=add(num1, num2);  
.....  
}  
return type of function  
data type of add  
int add(int a, int b)  
{  
int add;  
.....  
return add;  
}
```

sum = add

Function Types

For better understanding of arguments and return in functions, user-defined functions can be categorised as:

- Function with no arguments and no return value
- Function with no arguments and return value
- Function with arguments but no return value
- Function with arguments and return value.

Let's take an example to find whether a number is prime or not using above 4 categories of user defined functions.

Function with no arguments and no return value.

```
/*C program to check whether a number entered by user is prime or not using function with no arguments and no return value*/
#include <stdio.h>
void prime();
int main(){
    prime();    //No argument is passed to prime().
    return 0;
}
void prime(){
    /* There is no return value to calling function main(). Hence, return type of prime() is void */
    int num,i,flag=0;
    printf("Enter positive integer enter to check:\n");
    scanf("%d",&num);
    for(i=2;i<=num/2;++i){
        if(num%i==0){
            flag=1;
        }
    }
    if (flag==1)
        printf("%d is not prime",num);
    else
        printf("%d is prime",num);
}
```

Function prime() is used for asking user a input, check for whether it is prime or not and display it accordingly. No argument is passed and returned from prime() function.

Function with no arguments but return value

```
/*C program to check whether a number entered by user is prime or not using function with no arguments but having return value */
#include <stdio.h>
int input();
int main(){
    int num,i,flag;
    num=input();    /* No argument is passed to input() */
    for(i=2,flag=i;i<=num/2;++i,flag=i){
        if(num%i==0){
            printf("%d is not prime",num);
            ++flag;
            break;
        }
    }
    if(flag==i)
        printf("%d is prime",num);
    return 0;
}
int input(){ /* Integer value is returned from input() to calling function */
    int n;
    printf("Enter positive enter to check:\n");
    scanf("%d",&n);
    return n;
}
```

There is no argument passed to input() function But, the value of *n* is returned from input() to main()function.

Function with arguments and no return value

```
/*Program to check whether a number entered by user is prime or not using function with arguments and no return value */
#include <stdio.h>
void check_display(int n);
int main(){
    int num;
    printf("Enter positive enter to check:\n");
    scanf("%d",&num);
    check_display(num); /* Argument num is passed to function. */
    return 0;
}
void check_display(int n){
```

```

/* There is no return value to calling function. Hence, return type of function is void. */
int i,flag;
for(i=2,flag=i;i<=n/2;++i,flag=i){
    if(n%i==0){
        printf("%d is not prime",n);
        ++flag;
        break;
    }
}
if(flag==i)
    printf("%d is prime",n);
}

```

Here, check_display() function is used for check whether it is prime or not and display it accordingly. Here, argument is passed to user-defined function but, value is not returned from it to calling function.

Function with argument and a return value

```

/* Program to check whether a number entered by user is prime or not using function with
argument and return value */
#include <stdio.h>
int check(int n);
int main(){
    int num,num_check=0;
    printf("Enter positive enter to check:\n");
    scanf("%d",&num);
    num_check=check(num); /* Argument num is passed to check() function. */
    if(num_check==1)
        printf("%d in not prime",num);
    else
        printf("%d is prime",num);
    return 0;
}
int check(int n){
    /* Integer value is returned from function check() */
    int i;
    for(i=2;i<=n/2;++i){
        if(n%i==0)
            return 1;
    }
    return 0;
}

```


Here, check() function is used for checking whether a number is prime or not. In this program, input from user is passed to function check() and integer value is returned from it. If input the number is prime, 0 is returned and if number is not prime, 1 is returned.

Recursion

A function that calls itself is known as recursive function and the process of calling function itself is known as recursion in C programming.

Example of recursion in C programming

Write a C program to find sum of first n natural numbers using recursion. Note: Positive integers are known as natural number i.e. 1, 2, 3....n

```
#include <stdio.h>
int sum(int n);
int main(){
    int num,add;
    printf("Enter a positive integer:\n");
    scanf("%d",&num);
    add=sum(num);
    printf("sum=%d",add);
}
int sum(int n){
    if(n==0)
        return n;
    else
        return n+sum(n-1); /*self call to function sum() */
}
```

Output

Enter a positive integer:

5

15

In, this simple C program, sum() function is invoked from the same function. If n is not equal to 0 then, the function calls itself passing argument 1 less than the previous argument it was called with. Suppose, n is 5 initially. Then, during next function calls, 4 is passed to function and the value of argument decreases by 1 in each recursive call. When, n becomes equal to 0, the value of n is returned which is the sum numbers from 5 to 1.

For better visualization of recursion in this example:

```
sum(5)
=5+sum(4)
=5+4+sum(3)
=5+4+3+sum(2)
=5+4+3+2+sum(1)
=5+4+3+2+1+sum(0)
=5+4+3+2+1+0
=5+4+3+2+1
=5+4+3+3
=5+4+6
=5+10
=15
```

Every recursive function must be provided with a way to end the recursion. In this example when, n is equal to 0, there is no recursive call and recursion ends.

Advantages and Disadvantages of Recursion

Recursion is more elegant and requires few variables which make program clean. Recursion can be used to replace complex nesting code by dividing the problem into same problem of its sub-type.

In other hand, it is hard to think the logic of a recursive function. It is also difficult to debug the code containing recursion.

Examples:

C program to print Prime Numbers Between two Intervals by Making User-defined Function

```
#include<stdio.h>
int check_prime(int num);
int main(){
    int n1,n2,i,flag;
    printf("Enter two numbers(intervals): ");
    scanf("%d %d",&n1, &n2);
    printf("Prime numbers between %d and %d are: ", n1, n2);
    for(i=n1+1;i<n2;++i)
    {
        flag=check_prime(i);
        if(flag==0)
            printf("%d ",i);
    }
    return 0;
}
int check_prime(int num) /* User-defined function to check prime number*/
{
    int j,flag=0;
    for(j=2;j<=num/2;++j){
        if(num%j==0){
            flag=1;
            break;
        }
    }
    return flag;
}
```

Output:

Enter two numbers(intervals): 10

30

Prime numbers between 10 and 30 are: 11 13 17 19 23 29

C program to check prime and Armstrong number by Making User-defined Function

```
/* C program to check either prime number or Armstrong number depending upon the data
entered by user. */
#include <stdio.h>
int prime(int n);
int armstrong(int n);
int main()
```

```

{
    char c;
    int n,temp=0;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    printf("Enter P to check prime and A to check Armstrong number: ");
    c=getche();
    if (c=='p' || c=='P')
    {
        temp=prime(n);
        if(temp==1)
            printf("\n%d is a prime number.", n);
        else
            printf("\n%d is not a prime number.", n);
    }
    if (c=='a' || c=='A')
    {
        temp=armstrong(n);
        if(temp==1)
            printf("\n%d is an Armstrong number.", n);
        else
            printf("\n%d is not an Armstrong number.",n);
    }
    return 0;
}

int prime(int n)
{
    int i, flag=1;
    for(i=2; i<=n/2; ++i)
    {
        if(n%i==0)
        {
            flag=0;
            break;
        }
    }
    return flag;
}

int armstrong(int n)
{
    int num=0, temp, flag=0;
    temp=n;
    while(n!=0)
    {
        num+=(n%10)*(n%10)*(n%10);
        n/=10;
    }
}

```

```

    }
    if (num==temp)
        flag=1;
    return flag;
}

```

Output:

Enter a positive integer: 371

Enter P to check prime and A to check Armstrong number: p

371 is not a prime number.

Source Code to Calculated Sum using Recursion

```

#include<stdio.h>
int add(int n);
int main()
{
    int n;
    printf("Enter an positive integer: ");
    scanf("%d",&n);
    printf("Sum = %d",add(n));
    return 0;
}
int add(int n)
{
    if(n!=0)
        return n+add(n-1); /* recursive call */
}

```

Output:

Enter an positive integer: 10

Sum = 55

Source Code to Calculated factorial using Recursion

/* Source code to find factorial of a number. */

```

#include<stdio.h>
int factorial(int n);
int main()
{
    int n;
    printf("Enter an positive integer: ");
}

```

```

scanf("%d",&n);
printf("Factorial of %d = %ld", n, factorial(n));
return 0;
}
int factorial(int n)
{
    if(n!=1)
        return n*factorial(n-1);
}

```

Output:
Enter an positive integer: 6
Factorial of 6 = 720

Source code to reverse a sentence using recursion.

```

/* Example to reverse a sentence entered by user without using strings. */

#include <stdio.h>
void Reverse();
int main()
{
    printf("Enter a sentence: ");
    Reverse();
    return 0;
}
void Reverse()
{
    char c;
    scanf("%c",&c);
    if( c != '\n')
    {
        Reverse();
        printf("%c",c);
    }
}

```

Output:
Enter a sentence: margorp emosewa
awesome program

Source code to calculate power using recursion

```
/* Source Code to calculate power using recursive function */

#include <stdio.h>
int power(int n1,int n2);
int main()
{
    int base, exp;
    printf("Enter base number: ");
    scanf("%d",&base);
    printf("Enter power number(positive integer): ");
    scanf("%d",&exp);
    printf("%d^%d = %d", base, exp, power(base, exp));
    return 0;
}
int power(int base,int exp)
{
    if ( exp!=1 )
        return (base*power(base,exp-1));
}
```

Output:

Enter base number: 3

Enter power number(positive integer): 3

3^3 = 27

Array

In C programming, one of the frequently arising problem is to handle similar types of data. For example: If the user want to store marks of 100 students. This can be done by creating 100 variable individually but, this process is rather tedious and impracticable. These type of problem can be handled in C programming using arrays.

An array is a sequence of data item of homogeneous value(same type).

Arrays are of two types:

- One-dimensional arrays
- Multidimensional arrays

Declaration of one-dimensional array

```
data_type array_name[array_size];
```

For example:

```
int age[5];
```

Here, the name of array is *age*. The size of array is 5, i.e., there are 5 items (elements) of array *age*. All elements in an array are of the same type (int, in this case).

Array elements

Size of array defines the number of elements in an array. Each element of array can be accessed and used by user according to the need of program. For example:

```
int age[5];
```



Note that, the first element is numbered 0 and so on.

Here, the size of array *age* is 5 times the size of int because there are 5 elements.

Suppose, the starting address of `age[0]` is 2120d and the size of int be 4 bytes. Then, the next address (address of `a[1]`) will be 2124d, address of `a[2]` will be 2128d and so on.

Initialization of one-dimensional array:

Arrays can be initialized at declaration time in this source code as:

```
int age[5]={2,4,34,3,4};
```

It is not necessary to define the size of arrays during initialization.

```
int age[]={2,4,34,3,4};
```

In this case, the compiler determines the size of array by calculating the number of elements of an array.

age[0]	age[1]	age[2]	age[3]	age[4]
2	4	34	3	4

Initialization of one-dimensional array

Accessing array elements

In C programming, arrays can be accessed and treated like variables in C.

For example:

```
scanf("%d",&age[2]);
```

```
/* statement to insert value in the third element of array age[]. */
```

```
scanf("%d",&age[i]);
```

```
/* Statement to insert value in (i+1)th element of array age[]. */
```

```
/* Because, the first element of array is age[0], second is age[1], ith is age[i-1] and (i+1)th is age[i]. */
```

```
printf("%d",age[0]);
```

```
/* statement to print first element of an array. */
```

```
printf("%d",age[i]);
```

```
/* statement to print (i+1)th element of an array. */
```

Example of array in C programming

```
/* C program to find the sum marks of n students using arrays */
```

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

```
int main(){
```

```
    int marks[10],i,n,sum=0;
```

```
    printf("Enter number of students: ");
```

```
    scanf("%d",&n);
```

```
    for(i=0;i<n;++i){
```

```
        printf("Enter marks of student%d: ",i+1);
```

```

        scanf("%d",&marks[i]);
        sum+=marks[i];
    }
    printf("Sum= %d",sum);
return 0;
}

```

Output

```

Enter number of students: 3
Enter marks of student1: 12
Enter marks of student2: 31
Enter marks of student3: 2
sum=45

```

Multidimensional array

C programming language allows to create arrays of arrays known as multidimensional arrays.

For example:

```
float a[2][6];
```

Here, *a* is an array of two dimension, which is an example of multidimensional array. This array has 2 rows and 6 columns

For better understanding of multidimensional arrays, array elements of above example can be thought of as below:

	col 1	col 2	col 3	col 4	col 5	col 6
row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]
row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]

Figure: Multidimensional Arrays

Initialization of Multidimensional Arrays

In C, multidimensional arrays can be initialized in different number of ways.

```
int c[2][3]={{1,3,0}, {-1,5,9}};
```

OR

```
int c[][3]={{1,3,0}, {-1,5,9}};
```

OR

```
int c[2][3]={1,3,0,-1,5,9};
```

Initialization Of three-dimensional Array

```
double cprogram[3][2][4]={  
    {{-0.1, 0.22, 0.3, 4.3}, {2.3, 4.7, -0.9, 2}},  
    {{0.9, 3.6, 4.5, 4}, {1.2, 2.4, 0.22, -1}},  
    {{8.2, 3.12, 34.2, 0.1}, {2.1, 3.2, 4.3, -2.0}}  
};
```

Suppose there is a multidimensional array `arr[i][j][k][m]`. Then this array can hold $i*j*k*m$ numbers of data.

Similarly, the array of any dimension can be initialized in C programming.

Example of Multidimensional Array In C

Write a C program to find sum of two matrix of order 2*2 using multidimensional arrays where, elements of matrix are entered by user.

```
#include <stdio.h>  
int main(){  
    float a[2][2], b[2][2], c[2][2];  
    int i,j;  
    printf("Enter the elements of 1st matrix\n");  
    /* Reading two dimensional Array with the help of two for loop. If there was an array of 'n'  
    dimension, 'n' numbers of loops are needed for inserting data to array.*/  
    for(i=0;i<2;++i)  
        for(j=0;j<2;++j){  
            printf("Enter a%d%d: ",i+1,j+1);  
            scanf("%f",&a[i][j]);  
        }  
    printf("Enter the elements of 2nd matrix\n");  
    for(i=0;i<2;++i)  
        for(j=0;j<2;++j){  
            printf("Enter b%d%d: ",i+1,j+1);
```

```

scanf("%f",&b[i][j]);
}
for(i=0;i<2;++i)
for(j=0;j<2;++j){
/* Writing the elements of multidimensional array using loop. */
c[i][j]=a[i][j]+b[i][j]; /* Sum of corresponding elements of two arrays. */
}
printf("\nSum Of Matrix:");
for(i=0;i<2;++i)
for(j=0;j<2;++j){
printf("%.1f\t",c[i][j]);
if(j==1) /* To display matrix sum in order. */
printf("\n");
}
return 0;
}

```

Ouput

Enter the elements of 1st matrix

Enter a11: 2;

Enter a12: 0.5;

Enter a21: -1.1;

Enter a22: 2;

Enter the elements of 2nd matrix

Enter b11: 0.2;

Enter b12: 0;

Enter b21: 0.23;

Enter b22: 23;

Sum Of Matrix:

2.2 0.5

-0.9 25.0

Array and Function

In C programming, a single array element or an entire array can be passed to a function. Also, both one-dimensional and multi-dimensional array can be passed to function as argument.

Passing One-dimensional Array In Function

C program to pass a single element of an array to function

```
#include <stdio.h>
void display(int a)
{
    printf("%d",a);
}
int main(){
    int c[]={2,3,4};
    display(c[2]); //Passing array element c[2] only.
    return 0;
}
Output
4
```

Single element of an array can be passed in similar manner as passing variable to a function.

Passing entire one-dimensional array to a function

While passing arrays to the argument, the name of the array is passed as an argument(,i.e, starting address of memory area is passed as argument).

Write a C program to pass an array containing age of person to a function. This function should find average age and display the average age in main function.

```
#include <stdio.h>
float average(float a[]);
int main(){
    float avg, c[]={23.4, 55, 22.6, 3, 40.5, 18};
    avg=average(c); /* Only name of array is passed as argument. */
    printf("Average age=%.2f",avg);
    return 0;
}
```

```
float average(float a[]){
    int i;
    float avg, sum=0.0;
    for(i=0;i<6;++i){
        sum+=a[i];
    }
    avg =(sum/6);
    return avg;
}
```

Output

Average age=27.08

Passing Multi-dimensional Arrays to Function

To pass two-dimensional array to a function as an argument, starting address of memory area reserved is passed as in one dimensional array

Example to pass two-dimensional arrays to function

```
#include
void Function(int c[2][2]);
int main(){
    int c[2][2],i,j;
    printf("Enter 4 numbers:\n");
    for(i=0;i<2;++i)
        for(j=0;j<2;++j){
            scanf("%d",&c[i][j]);
        }
    Function(c); /* passing multi-dimensional array to function */
    return 0;
}
void Function(int c[2][2]){
    /* Instead to above line, void Function(int c[][2]){ is also valid */
    int i,j;
    printf("Displaying:\n");
    for(i=0;i<2;++i)
        for(j=0;j<2;++j)
            printf("%d\n",c[i][j]);
}
```

Output

Enter 4 numbers:

2

```
3
4
5
Displaying:
2
3
4
5
```

Examples

Source Code to Calculate Average Using Arrays

```
#include <stdio.h>
int main(){
    int n, i;
    float num[100], sum=0.0, average;
    printf("Enter the numbers of data: ");
    scanf("%d",&n);
    while (n>100 || n<=0)
    {
        printf("Error! number should in range of (1 to 100).\n");
        printf("Enter the number again: ");
        scanf("%d",&n);
    }
    for(i=0; i<n; ++i)
    {
        printf("%d. Enter number: ",i+1);
        scanf("%f",&num[i]);
        sum+=num[i];
    }
    average=sum/n;
    printf("Average = %.2f",average);
    return 0;
}
```

Source Code to Display Largest Element of an array

```
#include <stdio.h>
int main(){
    int i,n;
    float arr[100];
    printf("Enter total number of elements(1 to 100): ");
    scanf("%d",&n);
    printf("\n");
    for(i=0;i<n;++i) /* Stores number entered by user. */
    {
        printf("Enter Number %d: ",i+1);
        scanf("%f",&arr[i]);
    }
    for(i=1;i<n;++i) /* Loop to store largest number to arr[0] */
    {
        if(arr[0]<arr[i]) /* Change < to > if you want to find smallest element*/
            arr[0]=arr[i];
    }
    printf("Largest element = %.2f",arr[0]);
    return 0;
}
```

Source Code to Add Two Matrix in C programming

```
#include <stdio.h>
int main(){
    int r,c,a[100][100],b[100][100],sum[100][100],i,j;
    printf("Enter number of rows (between 1 and 100): ");
    scanf("%d",&r);
    printf("Enter number of columns (between 1 and 100): ");
    scanf("%d",&c);
    printf("\nEnter elements of 1st matrix:\n");

    /* Storing elements of first matrix entered by user. */

    for(i=0;i<r;++i)
        for(j=0;j<c;++j)
        {
            printf("Enter element a%d%d: ",i+1,j+1);
            scanf("%d",&a[i][j]);
        }

    /* Storing elements of second matrix entered by user. */
```



```

printf("Enter elements of 2nd matrix:\n");
for(i=0;i<r;++i)
    for(j=0;j<c;++j)
    {
        printf("Enter element a%d%d: ",i+1,j+1);
        scanf("%d",&b[i][j]);
    }

/*Adding Two matrices */

for(i=0;i<r;++i)
    for(j=0;j<c;++j)
        sum[i][j]=a[i][j]+b[i][j];

/* Displaying the resultant sum matrix. */

printf("\nSum of two matrix is: \n\n");
for(i=0;i<r;++i)
    for(j=0;j<c;++j)
    {
        printf("%d ",sum[i][j]);
        if(j==c-1)
            printf("\n\n");
    }

return 0;
}

```

Source code to multiply to matrix in C programming

```

#include <stdio.h>
int main()
{
    int a[10][10], b[10][10], mult[10][10], r1, c1, r2, c2, i, j, k;
    printf("Enter rows and column for first matrix: ");
    scanf("%d%d", &r1, &c1);
    printf("Enter rows and column for second matrix: ");
    scanf("%d%d",&r2, &c2);

    /* If colum of first matrix in not equal to row of second matrix, asking user to enter the size of
    matrix again. */
    while (c1!=r2)
    {

```

```

printf("Error! column of first matrix not equal to row of second.\n");
printf("Enter rows and column for first matrix: ");
scanf("%d%d", &r1, &c1);
printf("Enter rows and column for second matrix: ");
scanf("%d%d",&r2, &c2);
}

/* Storing elements of first matrix. */
printf("\nEnter elements of matrix 1:\n");
for(i=0; i<r1; ++i)
for(j=0; j<c1; ++j)
{
    printf("Enter elements a%d%d: ",i+1,j+1);
    scanf("%d",&a[i][j]);
}

/* Storing elements of second matrix. */
printf("\nEnter elements of matrix 2:\n");
for(i=0; i<r2; ++i)
for(j=0; j<c2; ++j)
{
    printf("Enter elements b%d%d: ",i+1,j+1);
    scanf("%d",&b[i][j]);
}

/* Initializing elements of matrix mult to 0.*/
for(i=0; i<r1; ++i)
for(j=0; j<c2; ++j)
{
    mult[i][j]=0;
}

/* Multiplying matrix a and b and storing in array mult. */
for(i=0; i<r1; ++i)
for(j=0; j<c2; ++j)
for(k=0; k<c1; ++k)
{
    mult[i][j]+=a[i][k]*b[k][j];
}

/* Displaying the multiplication of two matrix. */
printf("\nOutput Matrix:\n");
for(i=0; i<r1; ++i)
for(j=0; j<c2; ++j)
{
    printf("%d ",mult[i][j]);

```

```

        if(j==c2-1)
            printf("\n\n");
    }
    return 0;
}

```

Source Code to Find Transpose of a Matrix

```

include <stdio.h>
int main()
{
    int a[10][10], trans[10][10], r, c, i, j;
    printf("Enter rows and column of matrix: ");
    scanf("%d %d", &r, &c);

    /* Storing element of matrix entered by user in array a[][]. */
    printf("\nEnter elements of matrix:\n");
    for(i=0; i<r; ++i)
        for(j=0; j<c; ++j)
        {
            printf("Enter elements a%d%d: ", i+1, j+1);
            scanf("%d", &a[i][j]);
        }

    /* Displaying the matrix a[][] */
    printf("\nEntered Matrix: \n");
    for(i=0; i<r; ++i)
        for(j=0; j<c; ++j)
        {
            printf("%d ", a[i][j]);
            if(j==c-1)
                printf("\n\n");
        }

    /* Finding transpose of matrix a[][] and storing it in array trans[][]. */
    for(i=0; i<r; ++i)
        for(j=0; j<c; ++j)
        {
            trans[j][i]=a[i][j];
        }

    /* Displaying the transpose, i.e, Displaying array trans[][]. */
    printf("\nTranspose of Matrix:\n");
    for(i=0; i<c; ++i)
        for(j=0; j<r; ++j)
        {

```

```

        printf("%d ",trans[i][j]);
        if(j==r-1)
            printf("\n\n");
    }
    return 0;
}

```

Source Code to Multiply Matrix by Passing it to a Function

```

#include <stdio.h>
void take_data(int a[][10], int b[][10], int r1,int c1, int r2, int c2);
void multiplication(int a[][10],int b[][10],int mult[][10],int r1,int c1,int r2,int c2);
void display(int mult[][10], int r1, int c2);
int main()
{
    int a[10][10], b[10][10], mult[10][10], r1, c1, r2, c2, i, j, k;
    printf("Enter rows and column for first matrix: ");
    scanf("%d%d", &r1, &c1);
    printf("Enter rows and column for second matrix: ");
    scanf("%d%d",&r2, &c2);

    /* If colum of first matrix in not equal to row of second matrix, asking user to enter the size of
    matrix again. */

    while (c1!=r2)
    {
        printf("Error! column of first matrix not equal to row of second.\n");
        printf("Enter rows and column for first matrix: ");
        scanf("%d%d", &r1, &c1);
        printf("Enter rows and column for second matrix: ");
        scanf("%d%d",&r2, &c2);
    }
    take_data(a,b,r1,c1,r2,c2); /* Function to take matices data */
    multiplication(a,b,mult,r1,c1,r2,c2); /* Function to multiply two matrices. */
    display(mult,r1,c2); /* Function to display resultant matrix after multiplication. */
    return 0;
}

void take_data(int a[][10], int b[][10], int r1,int c1, int r2, int c2)
{
    int i,j;
    printf("\nEnter elements of matrix 1:\n");
    for(i=0; i<r1; ++i)
        for(j=0; j<c1; ++j)

```

```

    {
        printf("Enter elements a%d%d: ",i+1,j+1);
        scanf("%d",&a[i][j]);
    }

    printf("\nEnter elements of matrix 2:\n");
    for(i=0; i<r2; ++i)
    for(j=0; j<c2; ++j)
    {
        printf("Enter elements b%d%d: ",i+1,j+1);
        scanf("%d",&b[i][j]);
    }
}

void multiplication(int a[][10],int b[][10],int mult[][10],int r1,int c1,int r2,int c2)
{
    int i,j,k;
    /* Initializing elements of matrix mult to 0.*/
    for(i=0; i<r1; ++i)
    for(j=0; j<c2; ++j)
    {
        mult[i][j]=0;
    }
    /* Multiplying matrix a and b and storing in array mult. */
    for(i=0; i<r1; ++i)
    for(j=0; j<c2; ++j)
    for(k=0; k<c1; ++k)
    {
        mult[i][j]+=a[i][k]*b[k][j];
    }
}

void display(int mult[][10], int r1, int c2)
{
    int i, j;
    printf("\nOutput Matrix:\n");
    for(i=0; i<r1; ++i)
    for(j=0; j<c2; ++j)
    {
        printf("%d ",mult[i][j]);
        if(j==c2-1)
            printf("\n\n");
    }
}

```

Program To Sort Elements

```
/*C Program To Sort data in ascending order using bubble sort.*/
#include <stdio.h>
int main()
{
    int data[100],i,n,step,temp;
    printf("Enter the number of elements to be sorted: ");
    scanf("%d",&n);
    for(i=0;i<n;++i)
    {
        printf("%d. Enter element: ",i+1);
        scanf("%d",&data[i]);
    }

    for(step=0;step<n-1;++step)
    for(i=0;i<n-step-1;++i)
    {
        if(data[i]>data[i+1]) /* To sort in descending order, change > to < in this line. */
        {
            temp=data[i];
            data[i]=data[i+1];
            data[i+1]=temp;
        }
    }
    printf("In ascending order: ");
    for(i=0;i<n;++i)
        printf("%d ",data[i]);
    return 0;
}
```

Pointers

Pointers are the powerful feature of C and (C++) programming, which differs it from other popular programming languages like: java and Visual Basic.

Pointers are used in C program to access the memory and manipulate the address.

Reference operator(&)

If *var* is a variable then, *&var* is the address in memory.

```
/* Example to demonstrate use of reference operator in C programming. */
```

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

```
int main(){
```

```
    int var=5;
```

```
    printf("Value: %d\n",var);
```

```
    printf("Address: %d",&var); //Notice, the ampersand(&) before var.
```

```
    return 0;
```

```
}
```

Output

Value: 5

Address: 2686778

Note: You may obtain different value of address while using this code.

In above source code, value 5 is stored in the memory location 2686778. *var* is just the name given to that location.

You, have already used reference operator in C program while using `scanf()` function.

```
scanf("%d",&var);
```

Reference operator(*) and Pointer variables

Pointers variables are used for taking addresses as values, i.e., a variable that holds address value is called a pointer variable or simply a pointer.

Declaration of Pointer

Dereference operator(*) are used to identify an operator as a pointer.

```
data_type * pointer_variable_name;
```

```
int *p;
```

Above statement defines, *p* as pointer variable of type int.

Example To Demonstrate Working of Pointers

```
/* Source code to demonstrate, handling of pointers in C program */
#include <stdio.h>
int main(){
    int *pc,c;
    c=22;
    printf("Address of c:%d\n",&c);
    printf("Value of c:%d\n\n",c);
    pc=&c;
    printf("Address of pointer pc:%d\n",pc);
    printf("Content of pointer pc:%d\n\n",*pc);
    c=11;
    printf("Address of pointer pc:%d\n",pc);
    printf("Content of pointer pc:%d\n\n",*pc);
    *pc=2;
    printf("Address of c:%d\n",&c);
    printf("Value of c:%d\n\n",c);
    return 0;
}
```

Output

Address of c: 2686784

Value of c: 22

Address of pointer pc: 2686784

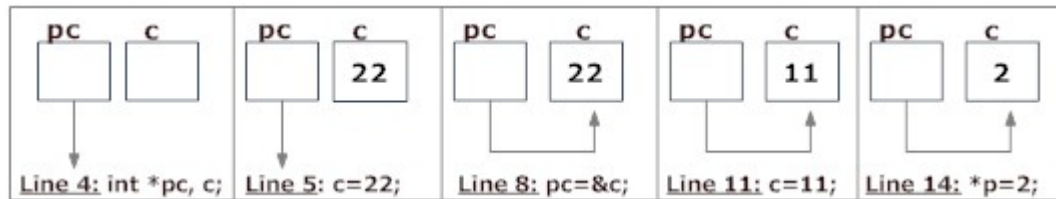
Content of pointer pc: 22

Address of pointer pc: 2686784

Content of pointer *pc*: 11

Address of *c*: 2686784

Value of *c*: 2



Explanation of program and figure

Code `int *pc, p;` creates a pointer *pc* and a variable *c*. Pointer *pc* points to some address and that address has garbage value. Similarly, variable *c* also has garbage value at this point.

Code `c=22;` makes the value of *c* equal to 22, i.e., 22 is stored in the memory location of variable *c*.

Code `pc=&c;` makes pointer, point to address of *c*. Note that, `&c` is the address of variable *c* (because *c* is normal variable) and *pc* is the address of *pc* (because *pc* is the pointer variable). Since the address of *pc* and address of *c* is same, `*pc` (value of pointer *pc*) will be equal to the value of *c*.

Code `c=11;` makes the value of *c*, 11. Since, pointer *pc* is pointing to address of *c*. Value of `*pc` will also be 11.

Code `*pc=2;` change the contents of the memory location pointed by pointer *pc* to change to 2. Since address of pointer *pc* is same as address of *c*, value of *c* also changes to 2.

Commonly done mistakes in pointers

Suppose, the programmer want pointer *pc* to point to the address of *c*. Then,

```
int c, *pc;
```

```
pc=c; /* pc is address whereas, c is not an address. */
```

```
*pc=&c; /* &c is address whereas, *pc is not an address. */
```

In both cases, pointer *pc* is not pointing to the address of *c*.

Pointers and array

Arrays are closely related to pointers in C programming. Arrays and pointers are synonymous in terms of how they use to access memory. But, the important difference between them is that, a pointer variable can take different addresses as value whereas, in case of array it is fixed. This can be demonstrated by an example:

```
#include <stdio.h>
int main(){
    char c[4];
    int i;
    for(i=0;i<4;++i){
        printf("Address of c[%d]=%x\n",i,&c[i]);
    }
    return 0;
}
```

Address of c[0]=28ff44
Address of c[1]=28ff45
Address of c[2]=28ff46
Address of c[3]=28ff47

Notice, that there is equal difference (difference of 1 byte) between any two consecutive elements of array.

Note: You may get different address of an array.

Relation between Arrays and Pointers

Consider and array:

```
int arr[4];
```

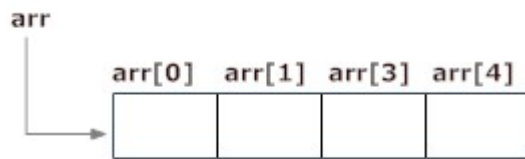


Figure: Array as Pointer

In arrays of C programming, name of the array always points to the first element of an array. Here, address of first element of an array is `&arr[0]`. Also, `arr` represents the address of the pointer where it is pointing. Hence, `&arr[0]` is equivalent to `arr`.

Also, value inside the address $\&arr[0]$ and address arr are equal. Value in address $\&arr[0]$ is $arr[0]$ and value in address arr is $*arr$. Hence, $arr[0]$ is equivalent to $*arr$.

Similarly,

$\&a[1]$ is equivalent to $(a+1)$ AND, $a[1]$ is equivalent to $*(a+1)$.

$\&a[2]$ is equivalent to $(a+2)$ AND, $a[2]$ is equivalent to $*(a+2)$.

$\&a[3]$ is equivalent to $(a+3)$ AND, $a[3]$ is equivalent to $*(a+3)$.

.

.

$\&a[i]$ is equivalent to $(a+i)$ AND, $a[i]$ is equivalent to $*(a+i)$.

In C, you can declare an array and can use pointer to alter the data of an array.

```
//Program to find the sum of six numbers with arrays and pointers.
```

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

```
int main(){
```

```
    int i,class[6],sum=0;
```

```
    printf("Enter 6 numbers:\n");
```

```
    for(i=0;i<6;++i){
```

```
        scanf("%d",class+i); // (class+i) is equivalent to &class[i]
```

```
        sum += *(class+i); // *(class+i) is equivalent to class[i]
```

```
    }
```

```
    printf("Sum=%d",sum);
```

```
    return 0;
```

```
}
```

Pointer and function

When, argument is passed using pointer, address of the memory location is passed instead of value.

Example of Pointer And Functions

Program to swap two number using call by reference.

```
/* C Program to swap two numbers using pointers and function. */
```

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

```
void swap(int *a,int *b);
```

```
int main(){
```

```
    int num1=5,num2=10;
```

```
    swap(&num1,&num2); /* address of num1 and num2 is passed to swap function */
```

```
    printf("Number1 = %d\n",num1);
```

```
    printf("Number2 = %d",num2);
```

```
    return 0;
```

```
}
```

```
void swap(int *a,int *b){ /* pointer a and b points to address of num1 and num2 respectively */
```

```
    int temp;
```

```
    temp=*a;
```

```
    *a=*b;
```

```
    *b=temp;
```

```
}
```

Output

Number1 = 10

Number2 = 5

Explanation

The address of memory location *num1* and *num2* are passed to function and the pointers **a* and **b* accept those values. So, the pointer *a* and *b* points to address of *num1* and *num2* respectively. When, the value of pointer are changed, the value in memory location also changed correspondingly. Hence, change made to **a* and **b* was reflected in *num1* and *num2* in main function.

This technique is known as call by reference in C programming.

Memory allocation

The exact size of array is unknown until the compile time, i.e., time when a compiler compiles code written in a programming language into an executable form. The size of array you have declared initially can be sometimes insufficient and sometimes more than required. Dynamic memory allocation allows a program to obtain more memory space, while running or to release space when no space is required.

Although, C language inherently does not have any technique to allocate memory dynamically, there are 4 library functions under "stdlib.h" for dynamic memory allocation.

Function	Use of Function
malloc()	Allocates requested size of bytes and returns a pointer first byte of allocated space
calloc()	Allocates space for an array elements, initializes to zero and then returns a pointer to memory
free()	deallocate the previously allocated space
realloc()	Change the size of previously allocated space

malloc()

The name malloc stands for "memory allocation". The function malloc() reserves a block of memory of specified size and return a pointer of type void which can be casted into pointer of any form.

Syntax of malloc()

```
ptr=(cast-type*)malloc(byte-size)
```

Here, *ptr* is pointer of cast-type. The malloc() function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.

```
ptr=(int*)malloc(100*sizeof(int));
```

This statement will allocate either 200 or 400 according to size of int 2 or 4 bytes respectively and the pointer points to the address of first byte of memory.

calloc()

The name calloc stands for "contiguous allocation". The only difference between malloc() and calloc() is that, malloc() allocates single block of memory whereas calloc() allocates multiple blocks of memory each of same size and sets all bytes to zero.

Syntax of calloc()

```
ptr=(cast-type*)calloc(n,element-size);
```

This statement will allocate contiguous space in memory for an array of *n* elements. For example:

```
ptr=(float*)calloc(25,sizeof(float));
```

This statement allocates contiguous space in memory for an array of 25 elements each of size of float, i.e, 4 bytes.

free()

Dynamically allocated memory with either `calloc()` or `malloc()` does not get return on its own. The programmer must use `free()` explicitly to release space.

syntax of free()

```
free(ptr);
```

This statement cause the space in memory pointer by `ptr` to be deallocated.

Examples of calloc() and malloc()

Write a C program to find sum of n elements entered by user. To perform this program, allocate memory dynamically using `malloc()` function.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)malloc(n*sizeof(int)); //memory allocated using malloc
    if(ptr==NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
    return 0;
}
```

Write a C program to find sum of n elements entered by user. To perform this program, allocate memory dynamically using calloc() function.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));
    if(ptr==NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
    return 0;
}
```

realloc()

If the previously allocated memory is insufficient or more than sufficient. Then, you can change memory size previously allocated using realloc().

Syntax of realloc()

```
ptr=realloc(ptr,newsize);
```

Here, *ptr* is reallocated with size of newsize.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *ptr,i,n1,n2;
    printf("Enter size of array: ");
```



```

scanf("%d",&n1);
ptr=(int*)malloc(n1*sizeof(int));
printf("Address of previously allocated memory: ");
for(i=0;i<n1;++i)
    printf("%u\t",ptr+i);
printf("\nEnter new size of array: ");
scanf("%d",&n2);
ptr=realloc(ptr,n2);
for(i=0;i<n2;++i)
    printf("%u\t",ptr+i);
return 0
;}
```

Example of pointer

Source Code to Calculate Average Using Arrays

```

#include <stdio.h>
int main(){
    int n, i;
    float num[100], sum=0.0, average;
    printf("Enter the numbers of data: ");
    scanf("%d",&n);
    while (n>100 || n<=0)
    {
        printf("Error! number should in range of (1 to 100).\n");
        printf("Enter the number again: ");
        scanf("%d",&n);
    }
    for(i=0; i<n; ++i)
    {
        printf("%d. Enter number: ",i+1);
        scanf("%f",&num[i]);
        sum+=num[i];
    }
    average=sum/n;
    printf("Average = %.2f",average);
    return 0;
}
```

Source Code to Access Array Elements Using Pointer

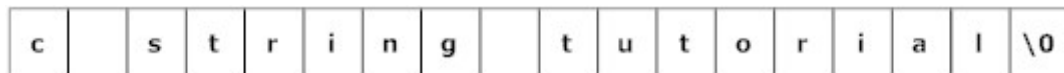
```
#include <stdio.h>
int main(){
    int data[5], i;
    printf("Enter elements: ");
    for(i=0;i<5;++i)
        scanf("%d",data+i);
    printf("You entered: ");
    for(i=0;i<5;++i)
        printf("%d\n",*(data+i));
    return 0;
}
```

Strings

In C programming, array of character are called strings. A string is terminated by null character `/0`. For example:

"c string tutorial"

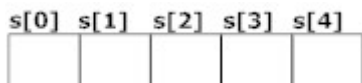
Here, "c string tutorial" is a string. When, compiler encounters strings, it appends null character at the end of string.



Declaration of strings

Strings are declared in C in similar manner as arrays. Only difference is that, strings are of char type.

char s[5];



Strings can also be declared using pointer.

char *p

Initialization of strings

In C, string can be initialized in different number of ways.

```
char c[]="abcd";
```

OR,

```
char c[5]="abcd";
```

OR,

```
char c[]={ 'a','b','c','d','\0' };
```

OR;

```
char c[5]= { 'a','b','c','d','\0' };
```

c[0]	c[1]	c[2]	c[3]	c[4]
a	b	c	d	\0

String can also be initialized using pointers

```
char *c="abcd";
```

Reading Strings from user.

Reading words from user.

```
char c[20];
```

```
scanf("%s",c);
```

String variable *c* can only take a word. It is because when white space is encountered, the `scanf()` function terminates.

Write a C program to illustrate how to read string from terminal.

```
#include <stdio.h>
int main() {
    char name[20];
    printf("Enter name: ");
    scanf("%s",name);
    printf("Your name is %s.",name);
    return 0;
```

```
}  
Output  
Enter name: Dennis Ritchie  
Your name is Dennis.
```

Here, program will ignore Ritchie because, `scanf()` function takes only string before the white space.

Reading a line of text

C program to read line of text manually.

```
#include <stdio.h>  
int main(){  
    char name[30],ch;  
    int i=0;  
    printf("Enter name: ");  
    while(ch!='\n') // terminates if user hit enter  
    {  
        ch=getchar();  
        name[i]=ch;  
        i++;  
    }  
    name[i]='\0'; // inserting null character at end  
    printf("Name: %s",name);  
    return 0;  
}
```

This process to take string is tedious. There are predefined functions `gets()` and `puts` in C language to read and display string respectively.

```
int main(){  
    char name[30];  
    printf("Enter name: ");  
    gets(name); //Function to read string from user.  
    printf("Name: ");  
    puts(name); //Function to display string.  
    return 0;  
}
```

Both, the above program has same output below:

Output

Enter name: Tom Hanks

Name: Tom Hanks

Passing Strings to Functions

String can be passed to function in similar manner as arrays as, string is also an array

```
#include <stdio.h>
void Display(char ch[]);
int main(){
    char c[50];
    printf("Enter string: ");
    gets(c);
    Display(c);    // Passing string c to function.
    return 0;
}
void Display(char ch[]){
    printf("String Output: ");
    puts(ch);
}
```

Here, string `c` is passed from `main()` function to user-defined function `Display()`. In function declaration, `ch[]` is the formal argument.

String manipulation

Strings are often needed to be manipulated by programmer according to the need of a problem. All string manipulation can be done manually by the programmer but, this makes programming complex and large. To solve this, the C supports a large number of string handling functions.

There are numerous functions defined in "string.h" header file. Few commonly used string handling functions are discussed below:

Function	Work of Function
<u>strlen()</u>	Calculates the length of string
<u>strcpy()</u>	Copies a string to another string
<u>strcat()</u>	Concatenates(joins) two strings
<u>strcmp()</u>	Compares two string
<u>strlwr()</u>	Converts string to lowercase
<u>strupr()</u>	Converts string to uppercase

Strings handling functions are defined under "string.h" header file, i.e, you have to include the code below to run string handling functions.

```
#include <string.h>
```

gets() and puts()

Functions gets() and puts() are two string functions to take string input from user and display string respectively as mentioned in previous chapter.

```
#include<stdio.h>
int main(){
    char name[30];
    printf("Enter name: ");
    gets(name);    //Function to read string from user.
    printf("Name: ");
    puts(name);    //Function to display string.
    return 0;
}
```

Though, gets() and puts() function handle string, both these functions are defined in "stdio.h" header file

Source Code to Find the Frequency of Characters

```
#include <stdio.h>
int main(){
    char c[1000],ch;
    int i,count=0;
    printf("Enter a string: ");
    gets(c);
    printf("Enter a character to find frequency: ");
    scanf("%c",&ch);
    for(i=0;c[i]!='\0';++i)
    {
        if(ch==c[i])
            ++count;
    }
    printf("Frequency of %c = %d", ch, count);
    return 0;
}
```

Source Code to Reverse String

```
include<stdio.h>
#include<string.h>
void Reverse(char str[]);
int main(){
    char str[100];
    printf("Enter a string to reverse: ");
    gets(str);
    Reverse(str);
    printf("Reversed string: ");
    puts(str);
    return 0;
}
void Reverse(char str[]){
    int i,j;
    char temp[100];
    for(i=strlen(str)-1,j=0; i+1!=0; --i,++j)
    {
        temp[j]=str[i];
    }
    temp[j]='\0';
    strcpy(str,temp);
}
```

Structure

Structure is the collection of variables of different types under a single name for better handling. For example: You want to store the information about person about his/her name, citizenship number and salary. You can create these information separately but, better approach will be collection of these information under single name because all these information are related to person.

Structure Definition in C

Keyword `struct` is used for creating a structure.

Syntax of structure

```
struct structure_name  
{  
    data_type member1;  
    data_type member2;  
    .  
    .  
    data_type member;  
};
```

We can create the structure for a person as mentioned above as:

```
struct person  
{  
    char name[50];  
    int cit_no;  
    float salary;  
};
```


This declaration above creates the derived data type **struct person**.

Structure variable declaration

When a structure is defined, it creates a user-defined type but, no storage is allocated. For the above structure of person, variable can be declared as:

```
struct person
```

```
{  
  
    char name[50];  
  
    int cit_no;  
  
    float salary;  
  
};
```

Inside main function:

```
struct person p1, p2, p[20];
```

Another way of creating sturcture variable is:

```
struct person
```

```
{  
  
    char name[50];  
  
    int cit_no;  
  
    float salary;  
  
}p1 ,p2 ,p[20];
```

In both cases, 2 variables *p1*, *p2* and array *p* having 20 elements of type **struct person** are created.

Accessing members of a structure

There are two types of operators used for accessing members of a structure.

Member operator(.)

Structure pointer operator(->) (will be discussed in structure and pointers chapter)

Any member of a structure can be accessed as: `structure_variable_name.member_name`

Suppose, we want to access salary for variable `p2`. Then, it can be accessed as:

`p2.salary`

Example of structure

Write a C program to add two distances entered by user. Measurement of distance should be in inch and feet.(Note: 12 inches = 1 foot)

```
#include <stdio.h>
struct Distance{
    int feet;
    float inch;
} d1,d2,sum;
int main(){
    printf("1st distance\n");
    printf("Enter feet: ");
    scanf("%d",&d1.feet); /* input of feet for structure variable d1 */
    printf("Enter inch: ");
    scanf("%f",&d1.inch); /* input of inch for structure variable d1 */
    printf("2nd distance\n");
    printf("Enter feet: ");
    scanf("%d",&d2.feet); /* input of feet for structure variable d2 */
    printf("Enter inch: ");
    scanf("%f",&d2.inch); /* input of inch for structure variable d2 */
    sum.feet=d1.feet+d2.feet;
    sum.inch=d1.inch+d2.inch;
    if (sum.inch>12){ //If inch is greater than 12, changing it to feet.
        ++sum.feet;
        sum.inch=sum.inch-12;
    }
    printf("Sum of distances=%d\'-%.1f'",sum.feet,sum.inch);
    /* printing sum of distance d1 and d2 */
    return 0;
}
```

Output

1st distance

Enter feet: 12

Enter inch: 7.9

2nd distance Enter feet: 2 Enter inch: 9.8 Sum of distances= 15'-5.7"
--

Keyword typedef while using structure

Programmer generally use typedef while using structure in C language. For example:

```
typedef struct complex {  
  
    int imag;  
  
    float real;  
  
}comp;
```

Inside main:

```
comp c1,c2;
```

Here, typedef keyword is used in creating a type *comp*(which is of type as **struct complex**). Then, two structure variables *c1* and *c2* are created by this *comp* type.

Structures within structures

Structures can be nested within other structures in C programming.

```
struct complex  
{  
  
    int imag_value;  
  
    float real_value;  
  
};  
  
struct number{  
  
    struct complex c1;
```

```
int real;  
}n1,n2;
```

Suppose you want to access *imag_value* for *n2* structure variable then, structure member *n1.c1.imag_value* is used.

Union

Unions are quite similar to the structures in C. Union is also a derived type as structure. Union can be defined in same manner as structures just the keyword used in defining union is **union** where keyword used in defining structure was **struct**.

```
union car{  
    char name[50];  
    int price;  
};
```

Union variables can be created in similar manner as structure variable.

```
union car{  
    char name[50];  
    int price;  
}c1, c2, *c3;
```

OR;

```
union car{  
    char name[50];  
    int price;  
};
```

-----Inside Function-----

```
union car c1, c2, *c3;
```

In both cases, union variables *c1*, *c2* and union pointer variable *c3* of type **union car** is created.

Accessing members of an union

The member of unions can be accessed in similar manner as that structure. Suppose, we you want to access price for union variable *c1* in above example, it can be accessed as *c1.price*. If you want to access price for union pointer variable *c3*, it can be accessed as *(*c3).price* or as *c3->price*.

Difference between union and structure

Though unions are similar to structure in so many ways, the difference between them is crucial

```
#include <stdio.h>
union job {      //defining a union
    char name[32];
    float salary;
    int worker_no;
}u;
struct job1 {
    char name[32];
    float salary;
    int worker_no;
}s;
int main(){
    printf("size of union = %d",sizeof(u));
    printf("\nsize of structure = %d", sizeof(s));
    return 0;
}
```

Output

```
size of union = 32
size of structure = 40
```

There is difference in memory allocation between union and structure as suggested in above example. The amount of memory required to store a structure variables is the sum of memory size of all members.



Fig: Memory allocation in case of structure

But, the memory required to store a union variable is the memory required for largest element of an union.

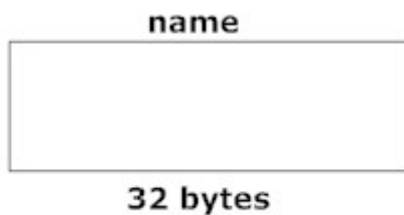


Fig: Memory allocation in case of union

What difference does it make between structure and union?

As you know, all members of structure can be accessed at any time. But, only one member of union can be accessed at a time in case of union and other members will contain garbage value.

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

```
union job {
```

```
    char name[32];
```

```
    float salary;
```

```
    int worker_no;
```

```
}u;
```

```
int main(){
```

```
    printf("Enter name:\n");
```

```
    scanf("%s",&u.name);
```

```
    printf("Enter salary: \n");
```

```
    scanf("%f",&u.salary);
```

```
    printf("Displaying\nName :%s\n",u.name);
```

```
printf("Salary: %.1f",u.salary);  
  
return 0;  
  
}
```

Output

Enter name

Hillary

Enter salary

1234.23

Displaying

Name: f%Bary

Salary: 1234.2

Note: You may get different garbage value of name.

Why this output?

Initially, *Hillary* will be stored in `u.name` and other members of union will contain garbage value. But when user enters value of salary, 1234.23 will be stored in `u.salary` and other members will contain garbage value. Thus in output, salary is printed accurately but, name displays some random string.

Examples of structure

C Program to Store Information of Single Variable

```
#include <stdio.h>  
struct student{  
    char name[50];  
    int roll;  
    float marks;  
};  
int main(){  
    struct student s;  
    printf("Enter information of students:\n\n");  
    printf("Enter name: ");  
    scanf("%s",s.name);
```

```

printf("Enter roll number: ");
scanf("%d",&s.roll);
printf("Enter marks: ");
scanf("%f",&s.marks);
printf("\nDisplaying Information\n");
printf("Name: %s\n",s.name);
printf("Roll: %d\n",s.roll);
printf("Marks: %.2f\n",s.marks);
return 0;
}

```

Source code to add two distance using structure

```

#include <stdio.h>
struct Distance{
    int feet;
    float inch;
}d1,d2,sum;
int main(){
    printf("Enter information for 1st distance\n");
    printf("Enter feet: ");
    scanf("%d",&d1.feet);
    printf("Enter inch: ");
    scanf("%f",&d1.inch);
    printf("\nEnter infomation for 2nd distance\n");
    printf("Enter feet: ");
    scanf("%d",&d2.feet);
    printf("Enter inch: ");
    scanf("%f",&d2.inch);
    sum.feet=d1.feet+d2.feet;
    sum.inch=d1.inch+d2.inch;

    /* If inch is greater than 12, changing it to feet. */
    if (sum.inch>12.0)
    {
        sum.inch=sum.inch-12.0;
        ++sum.feet;
    }
    printf("\nSum of distances=%d\'-%.1f'",sum.feet,sum.inch);
    return 0;
}

```


Source Code to Add Two Complex Number

```
#include <stdio.h>
typedef struct complex {
    float real;
    float imag;
} complex;
complex add(complex n1, complex n2);
int main() {
    complex n1, n2, temp;
    printf("For 1st complex number \n");
    printf("Enter real and imaginary respectively: \n");
    scanf("%f%f", &n1.real, &n1.imag);
    printf("\nFor 2nd complex number \n");
    printf("Enter real and imaginary respectively: \n");
    scanf("%f%f", &n2.real, &n2.imag);
    temp = add(n1, n2);
    printf("Sum = %.1f + %.1fi", temp.real, temp.imag);
    return 0;
}
complex add(complex n1, complex n2) {
    complex temp;
    temp.real = n1.real + n2.real;
    temp.imag = n1.imag + n2.imag;
    return(temp);
}
```

Source Code to Store Information of 10 students Using Structure

```
#include <stdio.h>
struct student {
    char name[50];
    int roll;
    float marks;
};
int main() {
    struct student s[10];
    int i;
    printf("Enter information of students: \n");
    for(i=0; i<10; ++i)
    {
        s[i].roll = i+1;
        printf("\nFor roll number %d\n", s[i].roll);
        printf("Enter name: ");
        scanf("%s", s[i].name);
        printf("Enter marks: ");
```

```

        scanf("%f",&s[i].marks);
        printf("\n");
    }
    printf("Displaying information of students:\n\n");
    for(i=0;i<10;++i)
    {
        printf("\nInformation for roll number %d:\n",i+1);
        printf("Name: ");
        puts(s[i].name);
        printf("Marks: %.1f",s[i].marks);
    }
    return 0;
}

```

Source Code Demonstrate the Dynamic Memory Allocation for Structure

```

#include <stdio.h>
#include<stdlib.h>
struct name {
    int a;
    char c[30];
};
int main(){
    struct name *ptr;
    int i,n;
    printf("Enter n: ");
    scanf("%d",&n);

    /* Allocates the memory for n structures with pointer ptr pointing to the base address. */
    ptr=(struct name*)malloc(n*sizeof(struct name));
    for(i=0;i<n;++i){
        printf("Enter string and integer respectively:\n");
        scanf("%s%d",&(ptr+i)->c, &(ptr+i)->a);
    }
    printf("Displaying Infomation:\n");
    for(i=0;i<n;++i)
        printf("%s\t%d\t\n",(ptr+i)->c,(ptr+i)->a);
    return 0;
}

```

Files

In C programming, file is a place on disk where a group of related data is stored.

Why files are needed?

When the program is terminated, the entire data is lost in C programming. If you want to keep large volume of data, it is time consuming to enter the entire data. But, if file is created, these information can be accessed using few commands.

There are large numbers of functions to handle file I/O in C language. In this tutorial, you will learn to handle standard I/O(High level file I/O functions) in C.

High level file I/O functions can be categorized as:

- Text file
- Binary file

File Operations

- Creating a new file
- Opening an existing file
- Reading from and writing information to a file
- Closing a file

Working with file

While working with file, you need to declare a pointer of type file. This declaration is needed for communication between file and program.

```
FILE *ptr;
```

Opening a file

Opening a file is performed using library function `fopen()`. The syntax for opening a file in standard I/O is:

```
ptr=fopen("filename", "mode")
```

For Example:

```
fopen("E:\\cprogram\\program.txt","w");
```

```
/* ----- */
```

E:\\cprogram\\program.txt is the location to create file.

"w" represents the mode for writing.

```
/* ----- */
```

Here, the program.txt file is opened for writing mode.

Opening Modes in Standard I/O		
File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exist, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading	If the file exists, its contents are

Opening Modes in Standard I/O

File Mode	Meaning of Mode	During Inexistence of file
	and writing.	overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.

Closing a File

The file should be closed after reading/writing of a file. Closing a file is performed using library function `fclose()`.

`fclose(ptr);` //ptr is the file pointer associated with file to be closed.

The Functions `fprintf()` and `fscanf()` functions.

The functions `fprintf()` and `fscanf()` are the file version of `printf()` and `scanf()`. The only difference while using `fprintf()` and `fscanf()` is that, the first argument is a pointer to the structure `FILE`

Writing to a file

```
#include <stdio.h>
int main()
{
    int n;
    FILE *fptr;
    fptr=fopen("C:\\program.txt","w");
    if(fptr==NULL){
        printf("Error!");
        exit(1);
    }
    printf("Enter n: ");
    scanf("%d",&n);
```

```
fprintf(fptr,"%d",n);
fclose(fptr);
return 0;
}
```

This program takes the number from user and stores in file. After you compile and run this program, you can see a text file program.txt created in C drive of your computer. When you open that file, you can see the integer you entered.

Similarly, `fscanf()` can be used to read data from file.

Reading from file

```
#include <stdio.h>
int main()
{
    int n;
    FILE *fptr;
    if ((fptr=fopen("C:\\program.txt","r"))==NULL){
        printf("Error! opening file");
        exit(1);    /* Program exits if file pointer returns NULL. */
    }
    fscanf(fptr,"%d",&n);
    printf("Value of n=%d",n);
    fclose(fptr);
    return 0;
}
```

If you have run program above to write in file successfully, you can get the integer back entered in that program using this program.

Other functions like `fgetchar()`, `fputc()` etc. can be used in similar way.

Binary Files

Depending upon the way file is opened for processing, a file is classified into text file and binary file.

If a large amount of numerical data is to be stored, text mode will be insufficient. In such case binary file is used.

Working of binary files is similar to text files with few differences in opening modes, reading from file and writing to file.

Opening modes of binary files

Opening modes of binary files are rb, rb+, wb, wb+,ab and ab+. The only difference between opening modes of text and binary files is that, b is appended to indicate that, it is binary file.

Reading and writing of a binary file.

Functions fread() and fwrite() are used for reading from and writing to a file on the disk respectively in case of binary files.

Function fwrite() takes four arguments, address of data to be written in disk, size of data to be written in disk, number of such type of data and pointer to the file where you want to write.

```
fwrite(address_data,size_data,numbers_data,pointer_to_file);
```

Function fread() also takes 4 arguments similar to fwrite() function as above.

Examples of file

Write a C program to read name and marks of n number of students from user and store them in a file.

```
#include <stdio.h>
int main(){
    char name[50];
    int marks,i,n;
    printf("Enter number of students: ");
    scanf("%d",&n);
    FILE *fptr;
    fptr=(fopen("C:\\student.txt","w"));
    if(fptr==NULL){
```

```

    printf("Error!");
    exit(1);
}
for(i=0;i<n;++i)
{
    printf("For student%d\nEnter name: ",i+1);
    scanf("%s",name);
    printf("Enter marks: ");
    scanf("%d",&marks);
    fprintf(fptr,"\nName: %s \nMarks=%d \n",name,marks);
}
fclose(fptr);
return 0;
}

```

Write a C program to read name and marks of n number of students from user and store them in a file. If the file previously exists, add the information of n students.

```

#include <stdio.h>
int main() {
    char name[50];
    int marks,i,n;
    printf("Enter number of students: ");
    scanf("%d",&n);
    FILE *fptr;
    fptr=(fopen("C:\\student.txt","a"));
    if(fptr==NULL){
        printf("Error!");
        exit(1);
    }
    for(i=0;i<n;++i)
    {
        printf("For student%d\nEnter name: ",i+1);
        scanf("%s",name);
        printf("Enter marks: ");
        scanf("%d",&marks);
        fprintf(fptr,"\nName: %s \nMarks=%d \n",name,marks);
    }
    fclose(fptr);
    return 0;
}

```


Write a C program to write all the members of an array of structures to a file using fwrite(). Read the array from the file and display on the screen.

```
#include <stdio.h>
struct s
{
char name[50];
int height;
};
int main(){
    struct s a[5],b[5];
    FILE *fptr;
    int i;
    fptr=fopen("file.txt","wb");
    for(i=0;i<5;++i)
    {
        fflush(stdin);
        printf("Enter name: ");
        gets(a[i].name);
        printf("Enter height: ");
        scanf("%d",&a[i].height);
    }
    fwrite(a,sizeof(a),1,fptr);
    fclose(fptr);
    fptr=fopen("file.txt","rb");
    fread(b,sizeof(b),1,fptr);
    for(i=0;i<5;++i)
    {
        printf("Name: %s\nHeight: %d",b[i].name,b[i].height);
    }
    fclose(fptr);
}
```

Example of C programs

```
//A Program to convert a length from kilometers to meters.
#include <stdio.h>
#include <conio.h>

void main(){
    float km, m;
    clrscr();

    printf("Enter the length in kilometer:");
    scanf("%f", &km);

    m = km*1000; //calculate equivalent of kilometer.

    printf("\nThe equivalent length of %f kilometer is %f meters", km, m); // Printing output
    getch();
}

//A program to calculate discount on items
#include <stdio.h>
#include <conio.h>

void main(){
    float item_1_price, item_2_price, item_3_price, item_1_discount = 0, item_2_discount =
0, item_3_discount = 0, total_purchase = 0;
    float item_1_net_price = 0, item_2_net_price = 0, item_3_net_price = 0, special_discount
= 0, net_cost = 0;
    printf(" \n*****Given there are three products*****");
    clrscr();
    printf("\nEnter the price of First Item:");
    scanf("%f",&item_1_price);

    printf("\nEnter the price of Second Item:");
    scanf("%f",&item_2_price);

    printf("\nEnter the price of ThirdItem:");
    scanf("%f",&item_3_price);

    item_1_discount = item_1_price * 0.15; //15% discount on first item.
    item_1_net_price = item_1_price - item_1_discount;

    item_2_discount = item_2_price * 0.10; //10% discount on second item.
    item_2_net_price = item_2_price - item_2_discount;
```

```

    item_3_discount = item_3_price * 0.05; //5% discount on third item.
    item_3_net_price = item_3_price - item_3_discount;

    total_purchase = item_1_net_price + item_2_net_price + item_3_net_price;

    if(total_purchase > 100000){
        printf("The total purchase exceeds 100000");
        special_discount = total_purchase * 0.01; // special discount of 1% if total
purchase is greater than 100000
        net_cost = total_purchase - special_discount;
    }else{
        net_cost = total_purchase;
    }

    printf("\n The net cost of all item after deduction is %f", net_cost);

    getch();
}

// A program to calculate commission amount
#include <stdio.h>
#include <conio.h>

void main(){
    int sales_amount;
    float commission_amount = 0;
    clrscr();
    printf("\n*****A program to find the commission amount onf the basis of sales
amount.*****");

    printf("\nEnter the sales amount:");
    scanf("%d", &sales_amount);

    if(sales_amount >= 0 && sales_amount <= 1000){
        commission_amount = sales_amount * 0.05;
    }else if(sales_amount >= 1001 && sales_amount <= 2000){
        commission_amount = sales_amount * 0.10;
    }else if(sales_amount > 2000){
        commission_amount = sales_amount * 0.12;
    }

    printf("\n The commission amount is %f", commission_amount);
    getch();
}

```

```

// A program to calculate weekly wages of worker
#include <stdio.h>
#include <conio.h>

void main(){

    int total_hours_worked;
    float weekly_wages = 0;
    clrscr();
    printf("\n *****A program to print weekly wages on the basis of number of hours
worked.*****");
    printf("\n\n Enter the total number of hours worked in a week:");
    scanf("%d", &total_hours_worked);

    if(total_hours_worked <= 288 && total_hours_worked >= 1 ){
        if(total_hours_worked <= 30){
            weekly_wages = total_hours_worked * 50;
        }else if(total_hours_worked <= 55){
            weekly_wages = 30 * 50 + (total_hours_worked - 30) * 50 * 1.5;
        }else if(total_hours_worked > 55){
            weekly_wages = 30 * 50 + 25 * 50 * 1.5 + (total_hours_worked - 55) * 50
* 2;
        }

        printf("\n\n The weekly wages of man who worked %d hours in a week is: %f",
total_hours_worked, weekly_wages);
    }else{
        printf("Wrong Input!!! A week cannot have more than 288 hours.");
    }

    getch();
}

// 5. A program to calculate marks, percentage of student.

# include <stdio.h>
# include <conio.h>
# define N 7

void main(){
    char subject[N];
    float marks[N];
    float total = 0.0, percentage = 0.0;
    int counter;
    int pass = 1;
    clrscr();

```

```

// Asking user to input marks of student.
printf("\nEnter the marks of different subject\n");
for(counter = 1; counter<=N; counter++){
    printf("\nEnter the marks of subject %d:", counter);
    scanf("%f", &marks[counter]);
}

//calculating total marks and percentage
for(counter = 1; counter<=7; counter++){
    if(marks[counter] < 35){
        pass = 0;
    }
    total += marks[counter];
}

if(pass == 1){
    percentage = total/N;
    if(percentage >= 60){
        printf("Congratulations!!! You have passed in first division, with %f
percent", percentage);
    }else if(percentage < 60 && percentage >= 45){
        printf("Congratulations!!! You have passed in second division, with %f
percent", percentage);
    }else if(percentage < 45 && percentage >= 35){
        printf("Congratulations!!! You have passed in third division, with %f
percent", percentage);
    }else{
        printf("Sorry!!! You are failed. \n Best of luck for next time");
    }
}
else{
    printf("Sorry!!! You are failed. \n Best of luck for next time");
}
getch();
}

// A program to calculate monthly interest of customer
#include <stdio.h>
#include <conio.h>

void main(){
    int principle_amount, period;
    float interest, monthly_interest;
    clrscr();
    printf("\n ***A program to calculate monthly interest of customer.***");
    printf("\n Enter the value of principal amount:");
    scanf("%d", &principle_amount);

```

```

printf("\n Enter the value of time period in month:");
scanf("%d", &period);

if(period < 6){
    interest = principle_amount * period * 5/100;
}else if(period >= 6 && period <= 12){
    interest = principle_amount*period*6/100;
}else{
    interest = principle_amount * period * 10/100;
}

monthly_interest = interest/12;
printf("The monthly interest of customer is %f", monthly_interest);

getch();
}

// A program to perform arithmetic operation
#include <stdio.h>
#include <conio.h>

void main(){
    float first_number, second_number, result;
    char choice;

    printf("\n Enter the first number:");
    scanf("%f", &first_number);

    printf("\n Enter the second number:");
    scanf("%f", &second_number);

    printf("\n Enter your choice a = add, s = subtract, m = multiply and d = divide\n");
    choice = getchar();
    if(choice == 'a'){
        result = first_number + second_number;
        printf(" You press a\n The addition of two number is %f", result);
    }else if(choice == 's'){
        result = first_number - second_number;
        printf(" \nYou press s\n The subtraction of two number is %f", result);
    }else if(choice == 'm'){
        result = first_number * second_number;
        printf(" \nYou press m\n The multiplication of two number is %f", result);
    }else if(choice == 'd'){
        result = first_number / second_number;
        printf(" \nYou press a\n The addition of two number is %f", result);
    }
}

```

```

    }else{
        printf("\n Wrong choice!!! Please input correct choice.");
    }

    getch();
}

// A program to calculate the square of natural numbers.
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define N 10

void main(){
    int counter1, counter2 = 1, counter3 = 1, number[N];

    printf("\n A program to calculate the square of natural numbers:");
    printf("Square of natural numbers using for loop");
    for(counter1 = 1; counter1 <= N; counter1++){
        printf("\n The square of %d is %d", counter1, pow(counter1, 2));
    }

    printf("Square of natural numbers using while loop");
    while(counter2 <= N){
        printf("\n The square of %d is %d", counter2, pow(counter2, 2));
    }

    printf("Square of natural numbers using do while loop");
    do{
        printf("\n The square of %d is %d", counter3, pow(counter3, 2));
    }while(counter3 <= N)

    getch();
}

// A program to calculate the sum of first natural numbers
#include <stdio.h>
#include <conio.h>
#define N 50

void main(){
    int counter, total_sum = 0;
    printf("\n A program to calculate the sum of first %d numbers", N);

    for(counter = 1; counter <= N; counter++){

```

```

        total_sum += counter;
    }

    printf("\n The sum of number is %d", total_sum);
    getch();
}

// A program to display sum of even number
#include <stdio.h>
#include <conio.h>

void main(){
    int total_number, number[N], even_sum = 0;

    printf("\n A program to display sum of even number");
    printf("\n Enter the total numbers");
    scanf("%d", &total_number);

    for(counter = 1; counter <= total_number; counter++){
        if(counter%2 == 0){
            even_sum += counter;
        }
    }
    printf("\n The sum of even number is %d", even_sum);
}

/*write a program to find the sum of the cube of the first 10 numbers.*/
// 11. Q.N. 29 of page 221
#include <stdio.h>
#include <conio.h>

void main(){
    int i, sum = 0;
    printf("\n A program to display sum of cube of 10 numbers");
    for(i=1; i<=10; i++){
        sum += i*i*i;
    }
    printf("\n The sum of cube of first 10 number is %d", sum);
    getch();
}

// 12. Q.N. 31 of page 221
#include <stdio.h>
#include <conio.h>
#define N 20

```



```

void main(){

    printf("Enter the number to display its multiplication table");
    scanf("%d", &number);

    for(i=1; i<=N; i++){
        printf("%d * %d = %d", number, i, number*i);
    }
    getch();
}

```

// A program to write a sentence to file
 # include <stdio.h>
 # include <conio.h>

```

void main(){
    FILE *fp;
    char *str;
    clrscr();

    fp = fopen("d:\\madhu.txt","a");
    clrscr();

    printf("\n Enter the name of student:");
    scanf("%[^\\n]", str);

    fprintf(fp, "%s\\n",str);

}

```

/* Write a multiplication table of 1 to 10*/
 # include <stdio.h>
 # include <conio.h>
 # define N 10

```

void main(){
    int i,k;
    clrscr();
    printf("The multiplication table of 1 to 10 is:");

    for(k=1; k<=N; k++){
        printf("\\n");
        for(i=1; i<=N; i++){
            printf("\\n%d * %d = %d", k, i, k*i);
        }
    }
}

```

```

        getch();
    }

/* Write a program to read a positive integer less than 20 and display its multiplication table.*/
# include <stdio.h>
# include <conio.h>
# define N 20

void main(){
    int number, i;
    clrscr();
    printf("Enter the number to display its multiplication table:");
    scanf("%d", &number);
    if(number<20){
        for(i=1; i<=N; i++){
            printf("\n%d * %d = %d", number, i, number*i);

        }
    }
    else{
        printf("\n The condition doesnot match");
    }
    getch();
}

/* Write a program to display fibonacci series upto 10 terms*/
# include <stdio.h>
# include <conio.h>
# define N 10

void main(){
    int x,y,z,i;
    clrscr();
    printf("\n Th Fibonacci series is:\n");
    x = 0;
    y = 1;

    printf("%d\t%d\t", x,y);
    for(i=0; i<N; i++){
        z = x+y;
        printf("%d\t", z);
        x=y;
        y=z;
    }
    getch();
}

```

```

/* Write a simple program using strcuture*/
// 14.
#include <stdio.h>
#include <conio.h>

void main()
{
    struct employee
    {
        char name[40];
        int age;
        char company[40];
        char designation[30];
        float salary;
    };
    clrscr();
    struct employee e1 = {"Priyanka Sahani", 18, "Nabil Bank", "Manager", 35000.00};
//Initialize employee e1
    struct employee e2, e3, e4;
    flushall();

    //Asking user to input information of employee e2
    printf("Enter the name of employee e2:");
    scanf("%s", e2.name);

    printf("Enter the age of employee e2:");
    scanf("%d", &e2.age);

    printf("Enter the name of company of employee e2:");
    scanf("%s", e2.company);

    printf("Enter the designation of employee e2:");
    scanf("%s", e2.designation);

    printf("Enter the salary of employee e2:");
    scanf("%f", &e2.salary);

    //Copying value of employee e1 to e3 and employee e2 to e4

    e3 = e1;
    e4 = e2;

    //printing informtion of all employee.

    printf("\nEmployee e1:\t\n");
    printf("Name = %s \n Age = %d\n Name of Company = %s\nDesignation = %s \n Salary

```

```

= %f\n", e1.name, e1.age, e1.company, e1.designation, e1.salary);

    printf("\nEmployee e3:\t\n");
    printf("Name = %s \n Age = %d\n Name of Company = %s\nDesignation = %s \n Salary
= %f\n", e3.name, e3.age, e3.company, e3.designation, e3.salary);

    printf("\nEmployee e2:\t\n");
    printf("Name = %s \n Age = %d\n Name of Company = %s\nDesignation = %s \n Salary
= %f\n", e2.name, e2.age, e2.company, e2.designation, e2.salary);

    printf("\nEmployee e4:\t\n");
    printf("Name = %s \n Age = %d\n Name of Company = %s\nDesignation = %s \n Salary
= %f\n", e4.name, e4.age, e4.company, e4.designation, e4.salary);

    getch();
}

/* Write a C program to print 10 terms of any series and display their sum using for loop.*/

# include <stdio.h>
# include <conio.h>
# include <math.h>

void main(){
    clrscr();

    float i, sum = 0.0, term;
    int N, x;

    printf("\n Enter the number of term in series:");
    scanf("%d", &N);

    printf("\n Enter the value of X:");
    scanf("%d", &x);

    printf("\n The required series is:\n");
    for(i=1; i<=N; i++){
        term=0.0;
        term = pow(x,i)/i;

        if(i<N){
            printf("%f+",term);
        }
        else{
            printf("%f",term);
            sum = sum + term;

```

```

    }

    }

    printf("\n The sum of series %f:", sum);
    getch();
}

/* A program to arrange the student records in ascending order.*/
// 15.
#include <stdio.h>
#include <conio.h>
#define N 5

void main(){
    struct Student{
        char name[30];
        int roll_no;
        char address[40];
        float marks;
    };

    Student s[N];

    //Entering the information of student.
    for(i=1; i<=N; i++){
        printf("\n Enter name of student %d:", i);
        scanf("%s", s[i].name);

        printf("\n Enter roll number of student %d:", i);
        scanf("%d", &s[i].roll_no);

        printf("\n Enter address of student %d:", i);
        scanf("%s", s[i].address);

        printf("\n Enter marks of student %d:", i);
        scanf("%f", &s[i].marks);
    }

    //sorting data into ascending order of marks
    for(i=1; i<=N; i++){
        for(j=1; j<=N-1; j++){
            if(s[i].marks > s[j+1].marks){
                temp = s[j];
                s[j] = s[j+1];
                s[j+1] = temp;
            }
        }
    }
}

```

```

    }
}

printf("\n The Student details in ascending order of marks are:\n");
for(i=1; i<=N; i++){
    printf("\n The Details of student %d:", i);

    printf("\n Name = %s:", s[i].name);
    printf("\n Roll No. = %d:", s[i].roll_no);
    printf("\n Address = %s:", s[i].address);
    printf("\n Marks = %f:", s[i].marks);
}
}

// A program to sort the number in ascending and descending order using.
#include <stdio.h>
#include <conio.h>
#define N 15

void main(){
    clrscr();
    int number[N], i, size;

    printf("\n Enter the number of elements in the array:");
    scanf("%d", &size);

    printf("Enter the elements of array:");
    for(i=0; i<=N ;i++){
        scanf("%d", &number[i]);
    }

    //Sorting in ascending order
    for(i=0; i<=N; i++){
        for(j=0; j<=N-1; j++){
            if(number[j] > number[j+1]){
                tmp = number[j];
                number[j] = number[j+1];
                number[j+1] = tmp;
            }
        }
    }

    // printing array in ascending order
    printf("\n Array in ascending order is:\n");
    for(i=0; i<=N; i++){
        printf("%d\n", number[i]);
    }
}

```

```

//Sorting in descending order
for(i=0; i<=N; i++){
    for(j=0; j<=N-1; j++){
        if(number[j] < number[j+1]){
            tmp = number[j];
            number[j] = number[j+1];
            number[j+1] = tmp;
        }
    }
}
// printing array in descending order
printf("\n Array in descending order is:\n");
for(i=0; i<=N; i++){
    printf("%d\n", number[i]);
}
getch();
}

```

```

//A program to generate Mersenne number
// A Mersenne number is the number in the form of  $2^n - 1$ 
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define N 10

void main(){
    clrscr();
    int i, size;
    printf("\n Enter the number of term in the series:");
    scanf("%d", &size);

    printf("\n The required Mersenne Number series:\n");
    for(i=1; i<=size; i++){
        printf("%d\t", pow(2,i)-1)
    }
    getch();
}

```

```

// A program to input character and change their case.
#include <stdio.h>
#include <conio.h>

void main(){

```

```

clrscr();
char ch;

printf("\n Enter any character; 0 to terminate:\t");
ch = getchar();

while(ch != 0){
    if(ch >= 'a' && ch <= 'z'){
        ch = ch-32;
    }else if(ch >= 'A' && ch <= 'Z'){
        ch = ch+32;
    }

    printf("\n The converted character is:%c\t", ch);
    printf("\n Enter any character:\t");
    ch = getchar();
}
getch();
}

// A program to read value of one-dimensional array, display the elements in reverse order and
print sum of elements.
# include <stdio.h>
# include <conio.h>
# define N 15

void main(){
    clrscr();
    int number[N], i, sum = 0;
    printf("Enter the elements of array:");
    for(i=0; i<=N ;i ++){
        scanf("%d", &number[i]);
    }

    printf("\n The array in reverse order is:\n");
    for(i=N; i>=0; i-- ){
        printf("%d\t", number[i]);
        sum = sum + number[i];
    }

    printf("\n The sum of elements of the given array is:\t%d", sum);
    getch();
}

```



```
// A program to check each element of one-dimensional array for evenness/oddness.
#include <stdio.h>
#include <conio.h>
```

```
void main(){
    clrscr();
    int number[N], i, j, size;

    printf("\n Enter the number of elements in the array:");
    scanf("%d", &size);

    printf("Enter the elements of array:");
    for(i=0; i<=N ;i++){
        scanf("%d", &number[i]);
    }
    printf("\nEven - Odd status:\n");
    for(j=0; j<size; j++){
        if(number[j]%2 == 0){
            printf("%d is even\n", number[j]);
        }else{
            printf("%d is odd\n", number[j]);
        }
    }
    getch();
}
```

```
// A program to check each element of one-dimensional array for primeness.
#include <stdio.h>
#include <conio.h>
#define N 50
```

```
void main(){
    clrscr();
    int number[N], i, j, flag, size, end_value;

    printf("\n Enter the number of elements in the array:");
    scanf("%d", &size);

    printf("Enter the elements of array:");
    for(i=0; i<=size ;i++){
        printf("\n");
        scanf("%d", &number[i]);
    }

    printf("\n Primeness of elements:\n");
    for(i=0; i<size; i++){
```

```

        flag = 0;
        end_value = number[i]/2;
        for(j=2; j<=end_value ;j++){
            if(number[i]%j == 0){
                flag = 1;
                break;
            }
        }
        if(flag == 1){
            printf("%d is not prime.\n", number[i]);
        }else{
            printf("%d is prime.\n", number[i]);
        }
    }
}

// A program to check each element of one-dimensional array is armstrong or not.
#include <stdio.h>
#include <conio.h>
#define N 50

void main(){
    clrscr();
    int number[N], i, j, size, p;

    printf("\n Enter the number of elements in the array:");
    scanf("%d", &size);

    printf("Enter the elements of array:");
    for(i=0; i<=size ;i++){
        printf("\n");
        scanf("%d", &number[i]);
    }

    printf("\n Primeness of elements:\n");
    for(i=0; i<size; i++){
        sum = 0;
    }
}

```

Conclusion

This is a simple project developed in C programming language.

References

1. www.google.com
2. <http://www.codeproject.com>
3. <http://www.programiz.com>
4. <http://www.codeincodeblock.com/>
5. Programming in ANSI C by E. Balagurusamy
6. Let us C by Yashwant Kantetkar