|  |
| --- |
| **C Notes** |
| Prepared by Rahul Sir |
| Do not copy |
|  |
|  |
| **Square Circle Computer Institute** |

***Syllabus of C Programming***

1. Introduction to C
2. Operators & Expressions
3. Type conversion
4. Conditional Statements/ Decision Control Structure
5. Iteration Statements/ Loops
6. Switch-Case Statements
7. Function
8. Storage Classes
9. Array
10. String
11. Pointer
12. Dynamic Memory Allocation (DMA)
13. Structure
14. Union
15. File handling
16. Projects/ Assignments

**What is C?**

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972 by Dennis Ritchie. Reasons for its popularity: It is Simple, Reliable & Easy to use.

**What are Constants, variables and key-words?**

**Constants:-** Values that do not change(Fixed)

Types :

* Integer Constants(1,2,3,-1,-2,-3, etc).
* Real Constants(1.0,2.0,3.0, 3.14, 9.8 etc).
* Character Constants( 'a' , 'b' , 'c' , '#' ,'&' ,etc).

**Variables:-** Variable is the name of a memory location which stores some data. (It also changes)

**Key-words:-** Reserved words that have special meaning to the compiler. C has 32 key-words.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Auto | const | double | Float | int | short | struct | unsigned |
| Break | continue | else | For | long | signed | switch | void |
| Case | default | enum | Goto | register | sizeof | typeof | volatile |
| Char | do | extern | If | return | static | union | while |

**Rules for Variables:**

* Variables are case sensitive.
* Keywords are not allowed to use as variables.
* No Comma or Blank space is allowed. However we can use underscore(\_).
* Variables should not start with number/Digits(0-9).

**What is operator?**

Operators are the foundation of any programming language. Thus the functionality of C languages in incomplete without the use of operators. Operators allow us to perform different kinds of operations on operands. In C, operators can be categorized based upon their different functionality.

For Example: if we write, a+b.

Here a & b are operands and + sign is an operator.

**Type of Operators:**

* Arithmetic Operators (+ , - , \* , /, %)
* Unary Operators
* Assignment Operators
* Relational Operators
* Logical Operators
* Bitwise Operators

**First C program:**

#include<stdio.h>

void main()

{

printf(“Hello World”);

}

**Program of addition of two numbers:**

#include<stdio.h>

void main()

{

int a,b,c;

a=10;

b=20;

c=a+b;

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

}

Output:- Sum of a & b is=30

**Another C program of finding total & average of three numbers:**

#include<stdio.h>

void main()

{

int a,b,c,tot,avg;

a=10;

b=20;

c=60;

tot=(a+b+c);

avg=tot/3;

printf("Total of numbers is=%d \n",tot);

printf("Average of numbers is=%d \n",avg);

}

Output: Total of numbers is=90

Average of numbers is=30

**Use of printf() function**

**It serves mainly two purposes:**

* To display your message on the screen/ console.

printf(“Hello World”);

* To display any variable containing some value.

printf(“format string”, variable);

**Practice Questions:**

**Write the following programs:-**

1) Write a program to find the Total & Average of three numbers.

2) Write a program to find the Whole Square of a & b.

3) Write a program to find the Area & Circumference of a circle.

4) Write a program to find the Simple Interest when Principal, Rate of Interest & Time are given.

5) Write a program to find the Total Marks & Percentage of five subjects.

6) Write a program to find the profit when Cost Price & Selling Price is given.

7) Write a program to evaluate the following expressions:-

i. x=a3+b3+c3+3abc

ii. y=a2-b2

**How to receive input from user?**

***Use scanf() function:***

***Scanf(“format string”, &variable/s);***

**Example 1:**

#include<stdio.h>

void main()

{

int a,b,c;

printf("Enter first number: \n");

scanf("%d",&a);

printf("Enter second number: \n");

scanf("%d",&b);

c=a+b;

printf("Sum of numbers is=%d \n",c);

}

Output:- As entered by the user.

**Example 2:**

#include<stdio.h>

void main()

{

int a,b,c,tot,avg;

printf("Enter first number:");

scanf("%d",&a);

printf("Enter second number:");

scanf("%d",&b);

printf("Enter third number:");

scanf("%d",&c);

tot=(a+b+c);

avg=tot/3;

printf("Total of numbers is=%d \n",tot);

printf("Average of numbers is=%d \n",avg);

}

Output:- As entered by the user.

**Practice Questions:**

**Write the following programs when input is given by the user:-**

1) Write a program to find the Total & Average of three numbers.

2) Write a program to find the Whole Square of a & b.

3) Write a program to find the Area & Circumference of a circle.

4) Write a program to find the Simple Interest when Principal, Rate of Interest & Time are given.

5) Write a program to find the Total Marks & Percentage of five subjects.

6) Write a program to find the profit when Cost Price & Selling Price is given.

7) Write a program to evaluate the following expressions:-

i. x=a3+b3+c3+3abc

ii. y=a2-b2

**Operators & expressions in C**

**We have following operators:**

1. Assignment Operator
2. Arithmetic Operators(+,-,\*,/,%)
3. Logical Operators
4. Increment/Decrement Operators
5. Shorthand arithmetic assignment Operators
6. Conditional Operators
7. Bitwise Operators
8. Comma Operator

**Some important & commonly used operators:**

* Assignment Operator
* Arithmetic Operators
* Logical Operators
* Increment/Decrement Operators
* Shorthand arithmetic assignment Operators

**1.Assignment Operator:**

The single equal symbol (=) is refered to as the assigment Operator.

It is used to assign the value of an expression to a variable.

**Syntax:**

variable=expression;

The expression can be a constant, variable or any valid C expression.

**Example:**

int a,b,c;

a=10;

b=a;

c=a+b;

**2.Arithmetic Operators(+,-,\*,/,%)**

#include<stdio.h>

void main()

{

int a=15,b=4;

printf("%d\n",a+b);

printf("%d\n",a-b);

printf("%d\n",a\*b);

printf("%d\n",a/b);

printf("%d\n",a%b);

}

**Output:**

19

11

60

3

3

**Task 1: Perform the above expression after taking input from the user.**

**Task 2: Evaluate the following expression:**

a) a+b\*c

b) a+b/c\*d

Here we can follow BODMAS, however,we can override the default order

of evaluation of an expression by using pair of parenthesis.

Result of a+b\*c is different from (a+b)\*c.

**3.Relational Operator:**

Relational operators are used to construct relational expressions

which are used to compare two quantities.

**For Example:**

a>b

a>10

b<a

a==b

a!=b

The value of relational operator is either true(1) or false(0).

**For example:**

#include<stdio.h>

void main()

{

int a=10,b=15;

printf("%d\n",a>b);

printf("%d\n",b>a);

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

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

}

**Output:**

0

1

0

1

**4.Logical Operators:**

The logical operators are used to construct compound conditional

expression. We have three logical operators: AND(&&), OR(||) and NOT(!).

**The working of AND(&&) operator is depicted below:**

Condition 1 Condition 2 Output

true true true

true false false

false true false

false false false

Both/All the conditions must be true to get 'true' as output. If any condition fails then output will be false. Note: We can check more than two conditions.

**The working of OR(||) operator is depicted below:**

Condition 1 Condition 2 Output

true true true

true false true

false true true

false false false

If any of the condition is true then output will be true.

If all the conditions get failed/false, then output will be false.

Note: We can check more than two conditions.

**NOT(!) operator:**

It reverses the output, from true to false and vice versa.

**6.Shorthand Arithmetic Assignment Operators**

These are commonly used Shorthand Arithmetic Assignment Operators:

[+=, -==, \*=, /=, %=]

Suppose 'a' is a variable with the value 10. If we want to add 5 to it.

What will we write?

a=a+5;

This assignment statement can now be written as:

a+=5;

It is same as a=a+5;

Similarly:

a=a-5 can be written as a-=5;

a=a\*5 can be written as a\*=5;

a=a/5 can be written as a/=5;

a=a%5 can be written as a%=5;

**Example of Shorthand Arithmetic Assignment Operators:**

#include<stdio.h>

void main()

{

int a,b,c,d,e;

a=b=c=d=e=10;

a+=5;

b-=5;

c\*=5;

d/=5;

e%=5;

printf("Value of a is=%d\n",a);

printf("Value of b is=%d\n",b);

printf("Value of c is=%d\n",c);

printf("Value of d is=%d\n",d);

printf("Value of e is=%d\n",e);

}

**6.Increment/Decrement Operator[++,--]**

#include<stdio.h>

void main()

{

int a=6,c=10,b;

b=a++;

printf("Value of b is=%d\n",b);

b=++c;

printf("Value of b is=%d\n",b);

b=c--;

printf("Value of b is=%d\n",b);

b=--c;

printf("Value of b is=%d\n",b);

}

Output:

Value of b is=6

Value of b is=11

Value of b is=11

Value of b is=9

**What is type conversion?**

It is a conversion from one data type to another data type.

Conversion takes place at two instances.

**i) At the time of evaluation of an expression:** Whenever an expression has two data items

which are of different types. Lower types converted into higher type. The result of

the expression will be in higher type mode.

The hierarchy of data types in the increasing order is given as follows:

char(The lowest data type)

short

unsigned

long

float

double

long double(The highest data type)

Example:

int a=10;

float f=3.14

a+f=10+3.14

=13.14

**ii) At the time of assignment of the value of an expression or source variable**

to a target variable. The value of the expression on the right hand side of an

assignment statement gets converted into the type of variable collecting it.

**Example:**

int c,a=10;

float f=3.14

c=a+f;

Here c is a variable of int type. The right hand side of the assignment statement

gets evaluated to 13.14. But the variable c on the left hand side collects only 13

since it is a variable of int type. If the collecting variable c were of float type,it would

have collected the exact result.

Under some circumstances, automatic type conversion does not work out for us.

For instance-

Consider the arithmetic expression 5/2. In C, the value of this expression would

be 2.

Since both 5 and 2 are integers, the result of the expression would also be

of int type. This is according to the rules of automatic type conversion. But

the exact result of this expression is 2.5.

How can we get this result?

**Type casting or forcible conversion is the answer to it.**

the general form of casting a value is as follow:

(type-name)expression;

example:

(float)5/2;

**Example:**

#include<stdio.h>

void main()

{

int a=10,a1;

float f=3.14,f1;

a1=f;

printf("Value of expression is=%d\n",a1);

f1=a;

printf("Value of expression is=%f\n",f1);

a1=5/2;

printf("Value of expression is=%d\n",a1);

a1=5.0/2;

printf("Value of expression is=%d\n",a1);

f1=5/2;

printf("Value of expression is=%f\n",f1);

f1=5.0/2;

printf("Value of expression is=%f\n",f1);

f1=(float)5/2;

printf("Value of expression is=%f\n",f1);

}

Output:

Value of expression is=3

Value of expression is=10.000000

Value of expression is=2

Value of expression is=2

Value of expression is=2.000000

Value of expression is=2.500000

Value of expression is=2.500000

**Decision Control Structure**

Sometimes we need to make decision in our real life. Decisions are made based on certain conditions. Like If weather is good, I will go for a long drive. If I complete my Graduation with 60% or above marks, I will get a Mobile Phone from my parents as a gift. Here, actions like long drive and mobile phone as a gift are the actions or outputs arising out of fulfillment of certain conditions. In the same way, in C language rather in any language, we need to execute certain action(s) based on certain conditions. If certain conditions are fulfilled, then a certain set of actions will be taken. For this, we have IF-ELSE statements.

**Simple example of IF Statement**

#include<stdio.h>

void main()

{

int per;

printf("Enter your percentage: \n");

scanf("%d",&per);

if(per>=60)

printf("Mobile phone is gifted \n");

}

Here if given condition holds true then statement after if gets executed , No statement will be run if condition gets failed.

**Simple example of IF-ELSE Statement**

#include<stdio.h>

void main()

{

int per;

printf("Enter your percentage: \n");

scanf("%d",&per);

if(per>=60)

printf("Mobile phone is gifted \n");

else

printf("You need to work hard!");

}

Here if given condition holds true then statement after if gets executed ,otherwise statement after else will be executed.

**Practice Sets:**

1. Write a program to find the whether the given age is eligible to vote or not.

2. Write a program to find whether the given number is Even or Odd.

3. Write a program to find whether the given year is Leap year or not.

4. Write a program to find the greater number out of two numbers.

6. Write a program to calculate the gross salary:

DA is 80%, TA is 40% and HRA is 20% if basic salary is above or equals to 10000.

Otherwise DA is 40%, TA is 20% and HRA is 10% if basic salary is below 10000.

(gross salary= basic Salary+DA+TA+HRA).

**Logical Operators:**

C allows usage of three logical operators:-

AND operator(&&)

OR operator(||)

NOT operator(!)

Here we will discuss first two operators, AND & OR.

These two operators allow two or more conditions to be combined in If statement.

**Use of AND Operator:**

AND operator checks all the given conditions. If All conditions are true then statements under IF get executed. If any condition out of given conditions get failed, it is assumed to be failed totally.

**Ex1).**

#include<stdio.h>

void main()

{

int age,salary;

printf("Enter your age: \n");

scanf("%d",&age);

printf("Enter your Salary: \n");

scanf("%d",&salary);

if(age>=25 && salary>=50000)

printf("You are mature enough for marriage:\n");

else

printf("You have to wait for marriage:\n");

}

**Use of OR Operator:**

OR operator checks all the given conditions. It needs only one condition to be true. If any of the given condition gets true, it works. It fails when all the conditions get failed.

**Ex2).**

#include<stdio.h>

void main()

{

int age,salary;

printf("Enter your age: \n");

scanf("%d",&age);

printf("Enter your Salary: \n");

scanf("%d",&salary);

if(age>=25 || salary>=50000)

printf("You are mature enough for marriage:\n");

else

printf("You have to wait for marriage:\n");

}

**Practice Set:**

1. The marks obtained by a student in 5 different subjects are input through the key-board. The student gets a division as per the following the rules:

Percentage above or equal to 60- First Division

Percentage between 50 and 59- Second Division

Percentage between 40 and 49- Third Division

Percentage less than 40- Fail

1. Write a program to find the greatest number out of three numbers.
2. Write a program of Login ID and Password. ID will be open if both ID and password get

Matched.

**Nested if-else**

Whenever we define if-else block inside another if-else block called

nested if-else.

Syntax:

if (Condition 1)

{

if (Condition 2)

{

Code;

}

else

{

Code;

}

}

else

{

if (Condition 3)

{

Code;

}

else

{

Code;

}

}

**Example:**

#include<stdio.h>

void main()

{

int a=20,b=10,c=45;

if(a>b)

{

if(a>c)

{

printf("Highest number is:%d \n",a);

}

else

{

printf("Highest number is:%d \n",c);

}

}

else

{

if(b>c)

{

printf("Highest number is:%d \n",b);

}

else

{

printf("Highest number is:%d \n",c);

}

}

}

**Else-if Statements**

It is used when we have only one if block, multiple else-if blocks and at the last else block.

Syntax:

if (Condition 1)

{

Statement 1;

}

else if (Condition 2)

{

Statement 2;

}

else

{

Statement 3;

}

**Example:**

#include<stdio.h>

void main()

{

int num;

printf("Enter any number:\n");

scanf("%d",&num);

if(num>0)

printf("Given number is positive:\n");

else if(num<0)

printf("Given number is negative:\n");

else

printf("Given number is zero:\n");

}

**Another example of else-if:**

#include<stdio.h>

void main()

{

int num;

printf("Enter any number(0-6):\n");

scanf("%d",&num);

if(num==0)

printf("It is Sunday:\n");

else if(num==1)

printf("It is Monday:\n");

else if(num==2)

printf("It is Tuesday:\n");

else if(num==3)

printf("It is Wednesday:\n");

else if(num==4)

printf("It is Thursday:\n");

else if(num==5)

printf("It is Friday:\n");

else

printf("It is Saturday:\n");

}

**What is Ternary Operator?**

The conditional operators ? and : are sometimes called ternary operators because

they take three arguments. In fact, they form a kind of foreshortened If-then-Else.

**Their general form is:**

**expression 1? expression 2: expression 3**

If expression 1 is true then value returned will be expression 2 otherwise

expression 3 will be returned.

**Ex 1).**

#include<stdio.h>

void main()

{

int a,b,c;

printf("Enter the value of A: \n");

scanf("%d",&a);

printf("Enter the value of B: \n");

scanf("%d",&b);

c=(a>b?printf("A is greater \n"):printf("B is greater \n"));

}

**Ex 2).**

#include<stdio.h>

void main()

{

int age,x;

printf("Enter the age: \n");

scanf("%d",&age);

x=(age>=18?printf("You are eligible to vote \n"):printf("You are Minor \n"));

}

**Loops**

***The loop is used when we want to perform a task repeatedly many times. There are three methods by which we can repeat a part of a program. They are:***

1. Using a while statement
2. Using a For Statement
3. Using a do-while statement

We will discuss each of these statements one by one.

***The While loop:***

It is often the case in programming that you want to do something a fixed number of times. Suppose we want to calculate the Simple Interest of 15 different customers/ Persons.

For this purpose while loop is best suited. We can avoid 15 separate executions by using this loop.

The operation of the while loop is illustrated in the following diagram.

**Start**

**Initialize**

**False**

**Test**

**Body of loop**

**Increment**

**Write the following programs using *while* loop:-**

1. Write a program to Calculate Simple Interest for 3 persons.
2. Write a program to print numbers from 1 to 10.
3. Write a program to print numbers from 10 to 1.
4. Write a program to Calculate the Area & Circumference of 4 circles of different radius.
5. Write a program to Calculate the profits of five articles when purchased & sold at different prices.

***The for loop:-***

Perhaps one reason why few programmers use while loop is that they are too busy using the for loop, which is probably the most popular looping instruction. The for allows us to specify three things about a loop in a single line:

1. Setting a loop counter to an initial value.
2. Testing of loop counter to determine whether its value has reached the number of repetitions desired.
3. Increasing the value of loop counter each time the program segment within the loop has been executed.

***The general form of for statement is as under:***

*for( Initialize Counter; Test Counter; Increment Counter)*

{

do this;

and this;

and this;

}

**Write the following programs using *for* loop:-**

1. Write a program to Calculate Simple Interest for 3 persons.
2. Write a program to print numbers from 1 to 10.
3. Write a program to print numbers from 10 to 1.
4. Write a program to Calculate the Area & Circumference of 4 circles of different radius.
5. Write a program to Calculate the profits of five articles when purchased & sold at different prices.

***The do-while loop:***

The do-while loop looks like this:

do

{

this;

and this;

and this;

and this;

} while(this condition is true)

There is a minor difference between the working of while and do-while loops. The difference is the place where the condition is tested. The while tests the condition before executing any of the statements within the while loop. As against this, the do-while tests the condition after having executed the statements within the loop. This means that do-while would execute its statements at least once, even if the condition fails for the first time. The while, on the other hand will not execute its statements if the condition fails for the first time.

**Jumps in Loops**

break & continue statements

**The break Statement**

We often come across situations where we want to jump out of a loop instantly, without waiting to get back to the condition. The keyword 'break' allows us to do this. While break is encountered inside a while,for or do-while loop, control automaticaly passes to the first statement after the loop. A break is

usually associated with an if.

**Let us understand the use of break statement.**

#include<stdio.h>

void main()

{

int n,i,sum=0,number;

printf("Enter number of elements:\n");

scanf("%d",&n);

for(i=1;i<=n;i++)

{

scanf("%d",&number);

if(number<0)

break;

sum+=number;

}

printf("Sum of number is=%d\n",sum);

}

**The continue statement**

In some programming situations, we want to take the control to the begining

of the loop, bypassing the statements inside the loop, which have not yet been

executed.The keyword 'continue' allows us to do this.

When continue is encountered inside a loop, control jumps to the begining

of the loop for performing next iteration. A 'continue' is usually associated

with an if.

**Let us understand the use of break statement.**

#include<stdio.h>

void main()

{

int n,i,sum=0,number;

printf("Enter number of elements:\n");

scanf("%d",&n);

for(i=1;i<=n;i++)

{

scanf("%d",&number);

if(number<0)

continue;

sum+=number;

}

printf("Sum of number is=%d\n",sum);

}

**Switch case statements:**

When we need to select one block of statements out of two alternatives, we use if-else structure. When we need to select one block of statements for execution out of n blocks of statements and condition is associated with each block of statements (All conditions being mutually exclusive), we tend to use else-if ladder. Since the conditions are mutually exclusive, only one condition would evaluate to true at any point of time resulting in the selection of the corresponding block of statements. Even though the programming situation is implementable using else-if ladder, the code becomes more complex when the number of conditions increase since the degree of readability decreases. Fortunately C provides switch structure, an alternative to else-if ladder, which simplifies the code and enhances readability when we need to implement problems which involve selection of one out of many alternatives.

***Syntax of switch structure***

**switch ( expression)**

{

case 1:

Statements;

case 2:

Statements;

case 3:

Statements;

default:

Statements;

}

***We can use break statements after every case if we need to execute particular case.***

**switch ( expression)**

{

case 1:

Statements;

break;

case 2:

Statements;

break;

case 3:

Statements;

break;

default:

Statements;

}

**Practice Questions:**

1. Write a program using switch case statements to perform calculations (Addition, Subtraction, Multiplication, and Division).
2. Write a program using switch case statements to calculate area of Circle, Rectangle, Triangle and Square.

**What is function?**

A function is defined to be a self-contained program which is written for the purpose of accomplishing some task.

**Classification of functions:-**

**We can categorize functions into two types:-**

* Built-in-function.
* User-defined Function.

**Built-in-function:-** these are the function which are already made available as a part of C library. They can be used by any programmer . For Example:- Scanf(), Printf(), Strlen(), Strcpy() Etc.

**User-defined Function:-** these are the function which is written by a user to solve a problem.

**SYNTAX:-**

Return-type Function-name (arguments)

{

Local variables;

Statements;

Return(expression);

}

**Functions are further classified into three types:-**

* Functions with no arguments and no return value.
* Functions with arguments and no return value.
* Functions with arguments and return value.

**Let us see each of them one by one:-**

**Functions with no arguments and no return value:-**

**#include<stdio.h>**

**void sum()**

**{**

**int a,b,c;**

**printf("enter the values of a and b");**

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

**c=a+b;**

**printf("sum of numbers is =%d \n",c);**

**}**

**void main()**

**{**

**sum();**

**}**

**Functions with arguments and no return value:-**

#include<stdio.h>

void sum(int a,int b)

{

int c;

c=a+b;

printf("sum of numbers is=%d\n",c);

}

void main()

{

int x,y;

printf("enter the values of x and y\n");

scanf("%d%d",&x,&y);

sum(x,y);

}

**Functions with arguments and return value:-**

#include<stdio.h>

int sum(int a,int b)

{

int c;

c=a+b;

return(c);

}

void main()

{

int x,y,add;

printf("enter the values of x and y");

scanf("%d%d",&x,&y);

add=sum(x,y);

printf("sum of numbers is= %d\n",add);

}

**What is Recursive Function?**

The phenomenon of a function calling itself is called recursion. The function involved in the process is referred to as a recursive function.

**Here are two programs involving the use of recursive function:-**

**To find the factorial of a number using recursion:-**

#include<stdio.h>

int fact(int num)

{

int f;

if(num==0)

return (1);

else

f=num\*fact(num-1);

return (f);

}

void main()

{

int n,fv;

printf("enter a number");

scanf("%d",&n);

fv=fact(n);

printf("factorial value is = %d\n",fv);

}

**To find x to power of y using recursion:-**

#include<stdio.h>

float power(int x,int y)

{

float p;

if(y==0)

return (1);

else if(y>0)

p=x\*power(x,y-1);

else

p=1/power(x,-y);

return (p);

}

void main()

{

int a,b;

float pow;

printf("Enter the values of a and b");

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

pow=power(a,b);

printf("%f\n",pow);

}

**Storage classes**

When we declare a variable of some type, the variable name and its data types are associated with the variable. It is identified by the given variable name and it is capable of storing value of the specified type.

**In addition to these, two important factors are associated with the variable:-**

1)Scope of the variable

2)Life time of the variable

Scope of a variable is defined to be the area of its existence in a program.

Lifetime of a variable is defined to be the duration of time for which the

variable exists.

**Depending on where variables are declared, they are of two types:**

1)Internal variables(Local Variables)

2)External variables(Global Variables)

**Internal variables**: These are those varaibles which are declared within the

body of a function.Internal variables are known only within the body of their

enclosing functions.

**External variables**:These are those variables which are declared outside the

functions.They are accessible from the point of declaration till the end of

the program.

**Example:**

int a;

void main()

{

int b;

}

void myfunction()

{

int c;

}

Here 'a' is a global variable, which is available and accessible for everyone. But 'b' and 'c' are local variables, they are available and accessible only for their respective enclosing functions.

**What is Array?**

An array is defined to be a group of logically related data items of similar types, stored in contiguous memory locations, sharing a common name but distinguished by subscript(s) value.

**Types of Array:**

Depending upon the number of subscripts, arrays are classified into one-dimensional, two-dimensional and so on. But at this level, we will restrict our discussion to one-dimensional array.

**Declaration of one-dimensional array:**

Data type variable-name[size]

Data type refers to any data type supported by C.

Variable name refers to array name and it should be a valid C identifier.

Size indicates number of data items of type data-type grouped together.

Each element in the array is identifiable by the array name (variable-name), followed by a pair of square brackets enclosing a subscript value. The subscript value ranges from 0 to size-1. When the subscript value is 0, first element in the array is selected, when the subscript value is 1, second element is selected and so on.

Ex. int a[5];

Here, ‘a’ is declared to be an array of int type and size five.

Five contiguous memory locations get allocated to store five integer values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 6 | 8 | 9 |

a[0] a[1] a[2] a[3] a[4]

Each data item in the array ‘a’ is identified by the array name ‘a’ followed by a pair of square brackets enclosing a subscripts value. The subscript value ranges from 0 to 4, i.e. a[0] denotes first data item, a[1] denotes second data item and a[4] denotes the fifth or last data item.

**Initialization of one-dimensional array:**

Just as we have initialized ordinary variables, we can initialize one-dimensional array also, i.e., locations of the array can be given values while they are declared. The syntax of initializing an array of one-dimensional is as follow:

Data type variable\_name [size]= {Value 0, Value 1, Value 2………..Value(size-1)}

Here,

1. **Data-type refers to any data type of the array elements.**
2. **Variable\_name refers to the name of the array.**
3. **{Value 0, Value 1, Value 2………..Value(size-1)} are the constant values of type data-type.**

In short, the syntax of initializing 1-d array can be written as:

Data-type Variable\_name [size]= {Initializer List};

As a result of this, memory locations of ‘a’ get filled up as follow:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 6 | 8 | 9 |

a[0] a[1] a[2] a[3] a[4]

**To illustrate declaration , initialization of a 1-d array and display its result.**

#include<stdio.h>

void main()

{

int i,a[5]={2,3,5,9,4};

printf("Elements of the array is:\n");

for(i=0;i<=4;i++)

printf("%d \n",a[i]);

}

**Output:-**

2 3 5 9 4

**Practice:**

* Write a program to display salary of 10 employees through array.
* Write a program to display the weight of 10 body builders through array.
* Write a program to display the letters of your name through array.

**To accept a list of numbers into 1-d array:**

#include<stdio.h>

void main()

{

int i, a[5];

printf("Enter five numbers:\n");

for(i=0;i<=4;i++)

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

printf("\n Entered numbers are as follow:\n");

for(i=0;i<=4;i++)

printf("%d\n",a[i]);

}

**Practice:**

* Write a program to accept & display salary of 10 employees through array.
* Write a program to accept & display the weight of 10 body builders through array.
* Write a program to accept & display the letters of your name through array.

**To accept a list of numbers into 1-d array, find their total, average and display them.**

#include<stdio.h>

void main()

{

int i,n,sum=0,avg, a[10];

printf("Enter number of elements:\n");

scanf("%d",&n);

printf("Enter numbers:\n");

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

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

printf("The entered numbers are as follow:\n");

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

printf("%d\n",a[i]);

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

sum=sum+a[i];

avg=sum/n;

printf("Total of numbers is=%d \n",sum);

printf("Average of numbers is=%d \n",avg);

}

**Practice:**

* Write a program to accept & display salary of 10 employees also find their total and average through array.
* Write a program to accept & display the weight of 10 body builders also find their total and average through array.

**Array & Functions:**

**One dimensional array as arguments to functions:**

To make a function take an array of one dimension as its input to be provided by a calling program, we need to define the function with two formal parameters:array name with data type specified, and the number of elements of the array being passed. The function is defined as:

function(array, size)

{

//Your code;

}

or,to be more precise

return-type function\_name(data-type array\_name[], int size)

{

//Your Code;

}

**Example:**

#include<stdio.h>

void display(int a[], int n)

{

int i;

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

printf("%d\n",a[i]);

}

void main()

{

int i,a[20],n;

printf("Enter number of values:\n");

scanf("%d",&n);

printf("Enter %d values \n",n);

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

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

printf("The elements are \n");

display(a,n);

}

**Exercise:**

1.WAP to display the elements of 1-D array.

2.WAP to find the sum of elements in a 1-D array.

**Strings in C**

Consider the statement printf("Welcome"); the printf() statement display the message "Welcome" on the screen. Let us now understant how the message "Welcome" is stored in memory and how the printf() works. The message "welcome" is stored in memory as:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| W | e | l | c | o | m | e | \0 |

Note that the last location in the memory has the special character '\0' is automatically appended to the end of the message by the compiler.

In the statement printf("Welcome"); The printf() is provided with the starting address of the memory block. That is, the address of the class character "W". printf() will then continue to pick each character from the block and displays it till the '\0' is reached. This is how the message id displayed. Here, the message "Welcome" is a string. To be precise, it is a string constant.

A string in C is defined to be sequence of characters terminated by the special characters '\0'. The special characters '\0’ is called null character and it is to indicate the end of a string.

An array of char is also regarded as a string variable, since it can accommodate a string and it permits us to change its contents. In contrast, a sequence of characters enclosed within a pair of double quotes is called a string constant.

For Example: “Welcome” is a string constant.

Initialization of Array of char Types

The syntax of initializing a string variable has two variations:

Variation 1

Char str1[5]={‘a’ , ‘b’, ‘c’, ‘d’, ‘\0’} ;

Here, str1 is declared to be a string variable with size five. It can accommodate the maximum five characters. The initializer-list consists of comma separated characters constants. Note that the null character ‘\0’ is explicitly listed. This is required in this variation.

Variation 2

Char str2[5]={“abcd”};

Here, str2 is also declared to be a string variable with size five. It can accommodate maximum five characters including null character. The initializer-list consists of a string constant. In this variation, null character ‘/0’ will be automatically appended to the end of string by the compiler.

In either of these variations, the size of the character array can be skipped, in which case, the size and the number of characters in the initializer-list would be automatically supplied by the compiler.

**Example:**

Char s1[]={“abcde”};

The size of s1 would be five, four characters plus one for the null character ‘/0’.

Char s2[]={‘a’ , ‘b’, ‘c’, ‘d’, ‘\0’} ;

The size of s2 would be five, four characters plus one for the null character ‘/0’.

**String I/O**

**Using scanf() and printf()**

Syntax of using scanf() to accept a string into a string variable s is as follow:

scanf(“%s”,s);

In case of string, in scanf() we do not use ‘&’ sign as string itself points to an address.

Above expression accepts a string into s upto a white space character.

Here white space means ( Blank Space, New Line character, Tab space).

The null character ‘/0’ will be automatically appended to the string. The size of s is thus expected to be larger enough to collect even the null character. %s is the format specifier for strings. Note that s is not preceded by symbol & (Address of operator). This is because, the string variable name s itself gives the address of the string.

**How to take input from the user?**

#include<stdio.h>

void main()

{

char name[30];

char post[25];

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

scanf("%s",name);

printf("Enter your post :\n");

scanf("%s",post);

printf("Name and post entered by you are as follow:\n");

printf("Your name is:%s \n",name);

printf("Your post is:%s \n",post);

}

**gets() & puts()**

The purpose of gets() is to accept a string upto a new line character into a string variable. It automatically appends the null character ‘\0’ to the end of the string.

The purpose of puts() is to display a string contained is a string variable. It also adds new-line character ‘\0’ to the string automatically, as a result of which the cursor is moved down by one line after the string is displayed.

**How to take multi-word input from the user?**

#include<stdio.h>

void main()

{

char name[30];

char post[25];

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

gets(name);

printf("Enter your post :\n");

gets(post);

printf("Name and post entered by you are as follow:\n");

puts(name);

puts(post);

}

**String manipulations:**

**The most commonly performed operations over strings are:**

1. Finding the length of string
2. Copy one string to another string
3. Comparing two strings
4. Concatenation of two strings

**How to find the length of a string?**

#include<stdio.h>

#include<string.h>

void main()

{

int n;

char name[30];

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

gets(name);

n=strlen(name);

printf("Length of your name is:%d \n",n);

}

**How to copy one string into other string?**

#include<stdio.h>

#include<string.h>

void main()

{

char name[30],surname[20];

printf("Enter your surname:\n");

gets(surname);

strcpy(name,surname);

printf("Your surname in now contained in 'name' is:%s",name);

}

**How to compare two strings?**

#include<stdio.h>

#include<string.h>

void main()

{

int diff;

char name1[30],name2[30];

printf("Enter two names:\n");

scanf("%s%s",name1,name2);

diff=strcmp(name1,name2);

printf("Difference between strings is:%d \n",diff);

}

**How to concatenate/ combine two strings?**

#include<stdio.h>

#include<string.h>

void main()

{

char name[50],surname[30];

printf("Enter your name and surname:\n");

scanf("%s%s",name,surname);

strcat(name,surname);

printf("Concatenated string is=%s\n",name);

}

**What is pointer?**

A pointer is a variable that can store the address of another variable.

The pointers enable us to:

1. Write efficient and concise programs
2. Establish inter-program data communication
3. Dynamically allocate and de-allocate memory
4. Optimizing memory space usage
5. Deal with hardware components
6. Deal with hardware components
7. Pass variable number of arguments to functions

**Pointer Operators:**

C provides two special operators known as pointer operators. They are ‘&’ and ‘\*’. & stands for “Address of” and it is used to retrieve the address of a variable. \* Stands for “Value at address” and it is used to access the value at a location by means of its address.

Since a pointer is also a variable, it should be declared before it is used. The syntax of declaring a pointer variable is written as:

**Data-type \*variable-name;**

Data-type is any valid data type supported by C or any user-defined type and variable-name is the name of the pointer variable. The presence of \* before variable-name indicates that it is a pointer variable.

**Example:**

Int \*ip;

ip is declared to be a pointer variable of int type.

Float \*fp;

fp is declared to be a pointer variable of float type.

**Simple program of pointer:**

#include<stdio.h>

void main()

{

int a=10,\*b;

b=&a;

printf("Value of a is:%d \n",a);

printf("Address of a is:%u \n",b);

printf("Value of b is:%d\n",b);

}

Value of a is:10

Address of a is:6487572

Value of b is:6487572

**Dynamic Memory Allocation**

It is a way to allocate memory to a data structure during runtime. We need some functions to allocate & free memory dynamically.

**Main functions for Dynamic Memory Allocation(DMA):**

* malloc(): For Memory Allocation
* calloc(): For Continuous Allocation
* free(): For Freeing of Memory
* realloc(): For Reallocation

**malloc():**

It takes number of bytes to be allocated & returns a pointer of type void.

How to allocate memory:

ptr=(int\*)malloc(n\*sizeof(data-types);

**How to check size of different data types:**

#include<stdio.h>

void main()

{

printf("%d\n",sizeof(int));

printf("%d\n",sizeof(float));

printf("%d\n",sizeof(char));

}

**How to allocate memory:**

#include<stdio.h>

#include<stdlib.h>

void main()

{

int i, \*ptr;

ptr=(int\*)malloc(5\*sizeof(int));

ptr[0]=10;

ptr[1]=20;

ptr[2]=30;

ptr[3]=40;

ptr[4]=50;

for(i=0;i<5;i++)

{

printf("%d \n",ptr[i]);

}

}

**Practice:**

WAP to allocate memory to store 5 prices.

**calloc():**

It is a continuous allocation which initializes with 0.

ptr=(int\*)calloc(n,sizeof(data-type));

#include<stdio.h>

#include<stdlib.h>

void main()

{

float \*ptr;

int i;

ptr=(float\*)calloc(5,sizeof(float));

for(i=0;i<5;i++)

{

printf("%f\n",ptr[i]);

}

}

**WAP to allocate memory of size n, where n is entered by the user.**

#include<stdio.h>

#include<stdlib.h>

void main()

{

int \*ptr,i,n;

printf("Enter the value of n:\n");

scanf("%d",&n);

ptr=(int\*)calloc(n,sizeof(int));

for(i=0;i<5;i++)

{

printf("%d\n",ptr[i]);

}

}

**free():**

We use it to free memory that is allocated using malloc & calloc.

free(ptr);

**Example:**

#include<stdio.h>

#include<stdlib.h>

void main()

{

int \*ptr;

int n,i;

printf("Enter the value of n: \n");

scanf("%d",&n);

ptr=(int\*)calloc(n,sizeof(int));

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

{

printf("%d\n",ptr[i]);

}

free(ptr);

printf("Now memory is free:\n");

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

{

printf("%d\n",ptr[i]);

}

}

**realloc():**

realloc(increase or decrease) memory using the same pointer & size.

ptr=realloc(ptr,newsize);

**Example:**

Allocate memory for 5 numbers, then dynamically increase it to 8 numbers.

#include<stdio.h>

#include<stdlib.h>

void main()

{

int \*ptr,i;

ptr=(int\*)calloc(5,sizeof(int));

printf("Enter number(5):\n");

for(i=0;i<5;i++)

{

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

}

ptr=realloc(ptr,8);

printf("Enter number(8):\n");

for(i=0;i<8;i++)

{

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

}

printf("Entered 8 values are as follow:\n");

for(i=0;i<8;i++)

{

printf("%d\n",ptr[i]);

}

}

**What is structure?**

A structure can be defined to be a group of logically related data items, which may be of different types, stored in contiguous memory locations, sharing a common name, but distinguished by its members.

**Definition of Structure template:**

Structure template definition helps in creating a format or prototype of the user-defined data type. The format of prototype of the user-defined data type allows us to logically relate a group of data items which may be of different types.

The syntax of defining a structure template is as follow:

**struct tag\_name/Structure\_name**

**{**

**Data-type member 1;**

**Data-type member 2;**

**Data-type member 3;**

**Data-type member 4;**

**Data-type member n;**

**}**

**Declaration of structure variable:**

Syntax of declaring a variable of structure type:

**Struct tag\_name variable-name;**

**Example of simple Structure:**

#include<stdio.h>

struct emp

{

char name[25];

int age;

int salary;

char post[25];

};

void main()

{

struct emp x;

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

scanf("%s",x.name);

printf("Enter your age:\n");

scanf("%d",&x.age);

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

scanf("%d",&x.salary);

printf("Enter your post:\n");

scanf("%s",x.post);

printf("Your name is:%s \n",x.name);

printf("Your age is:%d \n",x.age);

printf("Your salary is:%d \n",x.salary);

printf("Your post is:%s \n",x.post);

}

**We can insert more than one structure in a program:**

#include<stdio.h>

struct emp

{

char name[25];

int age;

int salary;

char post[25];

};

struct stu

{

char name[25];

int age;

float per;

char grade[10];

};

void main()

{

struct emp x;

struct stu y;

printf("Details of an employee:\n");

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

scanf("%s",x.name);

printf("Enter your age:\n");

scanf("%d",&x.age);

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

scanf("%d",&x.salary);

printf("Enter your post:\n");

scanf("%s",x.post);

printf("Your name is:%s \n",x.name);

printf("Your age is:%d \n",x.age);

printf("Your salary is:%d \n",x.salary);

printf("Your post is:%s \n",x.post);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Details of a student:\n");

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

scanf("%s",y.name);

printf("Enter your age:\n");

scanf("%d",&y.age);

printf("Enter your percentage:\n");

scanf("%f",&y.per);

printf("Enter your Grade:\n");

scanf("%s",y.grade);

printf("Your name is:%s \n",y.name);

printf("Your age is:%d \n",y.age);

printf("Your Percentage is:%f \n",y.per);

printf("Your Grade is:%s \n",y.grade);

}

**Exercise:**

Create a structure for the customer by taking their name, items, rate, qty, total amount and net amount by giving them some discount. Use this structure in your program.

**What is Union?**

The concept of Union is derived from the concept of structure. The common thing shared by both structure and union is that both enable us to identify a group of data items which may be of different types of a common name. But the difference lies in their storage allocation scheme.

In case of structure, the number of locations allocated would be equal to the number of members in the structures. Whereas in the case of union, only one loation which is large enough to collect the largest data type member in the union gets allocated. This single location can accomodate values of different type at different types one at a time.

**Syntax of union:**

union union\_name

{

data-type member 1;

data-type member 2;

data-type member n;

};

As a result of this, a memory allocation gets allocated, the size of which is equal to that of the largest of the numbers. Accessing the member of a union is similar to accessing the members of a structure. Dot operator is used to access each individual members.

**Example:**

union Alpha

{

int a;

float b;

char c;

};

*union Alpha x;*

A variable 'x' is declared to be of type union Alpha. As a result of this, only one memory location gets allocated. It can be referred to by any one individual member at any point of time. Note that the size of memory location is 4-bytes, which happens to be the size of the largest sized data type float in the member list.

**Example:**

#include<stdio.h>

struct Alpha

{

int a;

float b;

char c;

};

union Beta

{

int a;

float b;

char c;

};

void main()

{

struct Alpha x;

union Beta y;

printf("Size of structure Alpha is=%d\n",sizeof(x));

printf("Size of union Beta is=%d\n",sizeof(y));

}

**Output:**

Size of structure Alpha is=12

Size of union Beta is=4

**What is File handling?**

Storing data in a file and accessing data from a file is called file handling.

**What is file pointer?**

FILE is a (hidden) structute that needs to be created for opening a file. A FILE pointer that points to this structure and is used to access the file.

**Example:**

FILE fp;

**Different modes of File.**

r : For Reading

w : For Writing

a : For Appending

rb : Open to read in binary

wb : Open to write in binary

**Some important file handling operations.**

* Create
* Read
* Write
* Delete
* Copy

**How to create a file?**

#include<stdio.h>

void main()

{

FILE \*fp;

fp=fopen("Alpha.txt","w");

printf("File is created successfully:\n");

fclose(fp);

}

**How to write in the file?**

#include<stdio.h>

void main()

{

FILE \*fp;

int num;

fp=fopen("Alpha.txt","w");

if(fp==NULL)

{

printf("File is not created!\n");

}

printf("File created successfully:\n");

printf("Enter a number:\n");

scanf("%d",&num);

printf("Data has been written successfully:\n");

fprintf(fp,"%d",num);

fclose(fp);

}

**How to read from the file?**

#include<stdio.h>

void main()

{

FILE \*fp;

int num;

if((fp=fopen("Alpha.txt","r"))==NULL)

{

printf("Error Occurred:\n");

exit(1);

}

fscanf(fp,"%d",&num);

printf("%d\n",num);

fclose(fp);

}

**Additional Program for writing and reading a file:**

**How to write in the file?**

void main()

{

FILE \*fp;

char c;

fp=fopen("Beta.txt","w");

printf("Please write in the file:\n");

while((c=getchar())!=EOF)

putc(c,fp);

fclose(fp);

}

**How to read a file?**

#include<stdio.h>

void main()

{

FILE \*fp;

char c;

fp=fopen("Beta.txt","r");

if(fp==NULL)

{

printf("File not found!\n");

}

while((c=getc(fp))!=EOF)

printf("%c",c);

fclose(fp);

}