**Syntax for C Program:**

#include <stdio.h>

int main(void)

{

printf("Hello World\n");

return 0;

}

**What is Character Set:**

Uppercase letters A-Z

Lowercase letters a-z

Digits 0-9

Special symbols [!,@,.,#,$,%,^&,\*,(,),<,>,?,/,],[,},{,\](mailto:!,@,.,#,$,%,^&,*,(,),<,>,?,/,],[,},{,/)

**What is Keywords in C?**

Keywords - in-built/predefined reserved words in C.

- It gives special meaning to the compiler.

It will be always in the syntax.

Keywords cannot be used as identifiers.

**What is Identifier?**

Keywords ku pakathula use pana koodiya unique words ah dhan ‘Identifiers’ nu solluvanga.

Identiffier ngradhu is a unique word. If u use duplicate names as identifiers , it will throw an error.

**Where are these identiffiers used?**

It is used for the variables.Duplicate variable names cannot be used in a program.

Identifier names can be used only one time in a program .

It cannot be used more times in a program .

It will throw an error.

Identifiers are used for function names and variable names.

Simply, Identifier - means- unique names.

Data can be stored as variables,constants or literals.

Advantage of variable:

\* We can change the value of the variable anywhere inside the program.

\* Variable name can include alphanumeric characters.

**PROGRAM:**

#include <stdio.h>

int main(void)

{

Int money = 90;

Money =78;

return 0;

}

**What is literals?**

**Where and Why are literals used?**

Ans: Literals are used to represent the fixed values and are not assigned anywhere.

In literals, we have floating point literals such as

1. Numeric literals -postive and negative numbers.
2. Floating point literals - decimal values.(positive and negative values)
3. Character literals - any single thing inside single quotes. (I.e) any character inside ‘ ‘.
4. String literals - anything inside double quotes (“ “).

# What is a constant?

If u don’t want to change the value of a variable at anywhere,then u can declare a variable as constant by using the ‘const’ keyword.

**Advantage of constant:**

Because we cannot change a value/variable which is declared as constant anywhere in the program.

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

const double PI = 3.14;

PI = 4.5; // here u will get an error

}

**DATATYPES in C:**

In printf statement , for integer datatype, the format specifier is “%d” .

The integer can store value upto 4 bytes(32 bits).

If u want to store value beyond 4 bytes, then use ‘long int’.

For ‘long int’, the format specifier is “%ld”.

Long int can store upto 8 bytes(64 bits).

There is also another datatype called ‘short int’.

‘Short int’ can store only upto 2 bytes(16 bits).

char datatype can store only one byte(i.e)only 1 value within single quotes(’ ‘)

The format specifier for character is “%c”.

The format specifier for string is “%s”.

The format specifier for unsigned short integer is “%hu”.

The format specifier for signed short integer is “%hi” .

The format specifier for float is “%f”.

The format specifier for double is “%f” or “%lf”.

Float can store both normal whole number and a decimal number.

Double also can store both normal number and a decimal number.

The format specifier for double is either “%f” or “%lf”.

Float can store upto 4 bytes.

Double can store upto 8 bytes.

Integer -> whole numbers.

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int number =10;

long int r = 345;

short int s = 2;

char letter = ‘w’;

char[10] arr = ‘rambokhati’; //for storing the character array

float f = 1.2;

double d =2.4;

printf(“%lf”,d);

printf(“%f”,f);

printf(“%hi”,s);

printf(“%ld”,r);

printf(“%c”,letter);

printf(“%d”,number);

}

**Input and Output in C:**

To print 2 integer values, put two format specifiers. “%d%d”.

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int a =10;

int b =23;

float f =5.5f;

double d = 6.7;

char c= ‘e’;

printf(“%c”,c)

printf(“%lf”,d);

printf(“%f”,f);

printf(“%d%d”,a,b);

}

## How to get input?

To get input,use scanf() statement.

What type of input u r getting, what type of datatype u r getting as input , u should mention that format specifier in the scanf() statement.After putting that format specifier , type ‘&’ symbol and then enter that variable name .

### Why are we using this ‘&’ symbol?(very important in c)

Each variable has an address value associated with it.

Suppose consider the address of the variable is 3456783.

Now if u give 23 as input to the variable ‘testInteger’ , then the input value gets stored in the address of the variable . It doesn’t get directly stored on the variable name.

It is an indirect access.

The input value first gets stored on the address of the variable and then it gets stored on the particular variable name.

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int testInteger;

scanf(“%d”,&testInteger);

printf(“%d”,testInteger);

}

**Explanation:**

**Address of the variable ‘testInteger’: 3456783**

**Input value: 23**

**//3456783 = 23 (indirect storage of input value to the variable through address)**

**Input value is stored in the address.**

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int testInteger;

float input;

double in;

char letter;

scanf(“%f”,&input);

scanf(“%d”, &testInteger);

scanf(“%lf”,&in);

scanf(“%c”,&letter);

printf(“%d”, testInteger);

printf(“%f”,input);

printf(“%lf”,in);

printf(“%c”,letter);

}

Whenever we are getting the input from user iin scanf(), definitely we need ‘&’ symbol.

If u want to print the ascii value of a character,then see this following program.

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

char letter;

scanf(“%c”,&letter);

printf(“%d”,letter);

}

Within the same line,if u want to get multiple inputs,

For eg, if u want to get inputs from int and float datatype,then see the below program.

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int num1;

float num2;

scanf(“%d%f”,&num1,&num2);

printf(“%d%f”,num1,num2);

}

### OPERATORS IN C:

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int a=5,b=3;

printf(“a+b = %d\n”,a+b);

printf(“a-b = %d\n”,a -b);

printf(“a\*b = %d\n”,a \*b);

printf(“a/b = %d\n”,a /b);

//printf(“a%b = %d”,a %b); //Here I should not give this as a%b because it will consider this as the format specifier.

//So we should print this as

printf(“Remainder = %d\n”,a %b); //’\n’ is used for new line

}

**INCREMENT AND DECREMENT OPERATOR:**

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int a=5,b=3;

printf(“++a = %d\n”, ++a);

printf(“a++ = %d\n”, a++); //After the printf statement is over only, the variable will store and assign that incremented value.

//Already anga irukura 6 ngra value print aagi mudicha aprm dhan , anga 7 ngra value ve update aagum. So, u can see that updated changed value in the next line only.

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

printf(“--b = %d\n”, --b);

printf(“b-- = %d\n”, b--);

printf("%d\n",b); //After the printf statement is over only, the variable will store and assign that decremented value.

//Already anga irukura 2 ngra value print aagi mudicha aprm dhan , anga 1 ngra value ve update aagum. So, u can see that updated changed value in the next line only.

}

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int a = 5,b=3;

a+= b; //a=a+b;

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

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

}

#### RELATIONAL OPERATORS IN C:

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int a=5,b=3;

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);

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

}

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int a=5,b=3;

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

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

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

printf("%d\n",!(a==3));

}

**SIZEOF() OPERATOR:**

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int a;

float b;

double c;

char d ;

printf(“%lu\n” ,sizeof(a)); //whenever u r printing sizeof() operator ,use the unsigned integer format specifier “%lu”

//Because indha sizeof() operator la irundhu enaku vara pora values ellame bytes la dhan vara pogudhu . So, na indha unsigned integer oda format specifier use pandren.

printf(“%lu\n” ,sizeof(b));

printf(“%lu\n” ,sizeof(c));

printf(“%lu\n” ,sizeof(d));

//So, na ipo indha a,b,c,d ella datatype oda size enna nu paaka poren.

}

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int a=5;

if(a<3)

{

printf(“True”);

}

else if(a>2)

{

printf(“High”);

}

else

{

printf(“False”);

}

}

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int a=5;

if(a>3)

{

if(a==5)

{

printf(“True”);

}

else

{

printf(“High”);

}

}

else

{

printf(“False”);

}

}

#### LOOPS IN C:

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int i;

for(i=0;i<10;i++)

{

printf(“%d”,i);

}

}

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int i;

for(i=0;i<10;++i)

{

printf(“%d”,i);

}

}

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int i;

for(i=1;i<11;++i)

{

printf(“%d”,i);

}

}

**EXAMPLE PROGRAM FOR WHILE LOOP:**

#include <stdio.h>

int main(void)

{

int i=0;

while(i<10)

{

printf(“%d”,i);

i++;

}

}

**EXAMPLE PROGRAM FOR DO-WHILE LOOP:**

#include <stdio.h> //After running the program only,it will check the condition itself.

int main(void)

{

int i=2;

do

{

printf(“%d”,i);

i++;

}

while(i<1);

}

**OUTPUT :** 2

**EXAMPLE PROGRAM FOR WHILE LOOP:**

#include <stdio.h> //After running the program only,it will check the condition itself.

int main(void)

{

int i=2;

while(i<1)

{

printf(“%d”,i);

i++;

}

}

**EXAMPLE PROGRAM FOR WHILE LOOP:**

#include <stdio.h> //After running the program only,it will check the condition itself.

int main(void)

{

int i=0;

do

{

printf(“%d”,i);

i++;

}

while(i<10);

}

**Break and Continue in C:**

These 2 statements will work in loops only.(more usefully)

**Example Program for BREAK Statement:**

#include <stdio.h> //After running the program only,it will check the condition itself.

int main(void)

{

for(int i=0;i<10;i++)

{

if(i==5)

{

break; //forcefully stop the loop and come out of the loop.

}

printf(“%d”,i);

}

}

**What is CONTINUE statement?**

CONTINUE statement vandhu … oru set of code ah apdiye skip pannirum.

Continue statement will skip a particular set of code.

**Example Program for CONTINUE Statement:**

#include <stdio.h> //After running the program only,it will check the condition itself.

int main(void)

{

for(int i=0;i<10;i++)

{

if(i==5)

{

continue; //forcefully stop the loop and come out of the loop.

}

printf(“%d”,i);

}

}

**OUTPUT:** 0 1 2 3 4 6 7 8 9

**EXPLANATION:**

In the above code, when it reaches (i==5), it will not proceed with the next statement. Again it will go back to the top of the loop.

i == 5 ah irukumbodhu , adhu adhuku aprm adutha line ke varaama thiruppi top of the loop ke poirum .

**SWITCH Case in C:**

Switch statement is similar to that of like a if statement ,but it works on the basis of cases.

Case ah base panni dhan adhu work aagum.

Switch kulla,cases ah vandhu mention pannuvom.

We will mention the cases in the switch statement.

In each and every case, definitely there must be a break statement.

Because , after executing the case statements, definitely it must forcefully come out of the switch statement.

If we doesn’t give break statement in the (switch)case, then all the cases will start running.

Each and every case what we have mentioned will start running.

Case 1 means it is checking whether the value of a is 1.

Case 2 means it is checking whether the value of a is 2.

Case 3 means it is checking whether the value of a is 3.

It goes and so on.

Since our value of a is 3, it is running the case 3.

**Example Program for SWITCH CASE Statement:**

#include <stdio.h> //After running the program only,it will check the condition itself.

int main(void)

{

int a=3;

switch(a)

{

case 1:

printf("Hello");

break;

case 2:

printf("Hello World");

break;

case 3:

printf("High");

break;

}

}

Suppose if we didn’t give break in any of the switch cases,then the output will be

**EXAMPLE PROGRAM:**

#include <stdio.h> //After running the program only,it will check the condition itself.

int main(void)

{

int a=1;

switch(a)

{

case 1:

printf("Hello");

//break;

case 2:

printf("Hello World");

//break;

case 3:

printf("High");

//break;

}

}

**OUTPUT:**  Hello Hello World High

**EXPLANATION:**

That is , endha statement true aagudho, adhuku keela irundhu (including that true statement)ella statements um adhu run pannum.

**EXAMPLE PROGRAM 1:**

#include <stdio.h> //After running the program only,it will check the condition itself.

int main(void)

{

int a=2;

switch(a)

{

case 1:

printf("Hello");

//break;

case 2:

printf("Hello World");

//break;

case 3:

printf("High");

//break;

}

}

**OUTPUT:**  Hello World High

**EXAMPLE PROGRAM 2:**

#include <stdio.h> //After running the program only,it will check the condition itself.

int main(void)

{

int a=3;

switch(a)

{

case 1:

printf("Hello");

break;

case 2:

printf("Hello World");

break;

case 3:

printf("High");

break;

}

}

**OUTPUT:**  High

So only, the break statement is very important in switch case.

Whenever u give break statement in a case, it comes out of the switch(entire) immediately.

In ‘;if’ condition , how we have another facility like ‘else’ , samewise , in switch also , we have another facility like ‘default’.

What is the use of **default** means, suppose if all the switch cases has been failed means, then this default case will run.

**EXAMPLE PROGRAM for Default:**

#include <stdio.h> //After running the program only,it will check the condition itself.

int main(void)

{

int a=3;

switch(a)

{

case 1:

printf("Hello");

break;

case 2:

printf("Hello World");

break;

case 3:

printf("High");

break;

default:

printf("Default");

}

}

Suppose if any of the switch cases has been runned , then also this default will run along with that passed switch case.

**OUTPUT:** High Default

Default is more similarly like else condition . But this default will run along with the passed switch case.

**EXAMPLE PROGRAM for Default:**

#include <stdio.h> //After running the program only,it will check the condition itself.

int main(void)

{

int a=10;

switch(a)

{

case 1:

printf("Hello");

break;

case 2:

printf("Hello World");

break;

case 3:

printf("High");

break;

default:

printf("Default");

}

}

**OUTPUT:** Default

**goto statement in C:**

#include <stdio.h>

int main(void)

{

int a=10;

if(a>5)

{

goto pandiyanstores;

}

pandiyanstores:

printf("Hello World");

}

It is better not to use goto statement in C programming because without goto statement itself, we can do everything in C programming.

**Why we should not use goto?**

Goto statement can do everything .That’s why we are telling not to use goto statement in C. Because, if we use goto, there is a lot of chances of getting bug/bugs in the program.

There is a chance to get lot of confusion in the code.

So only, we need to know about the goto statement but it is better to avoid not using the goto statement in C.

Label can be given anywhere inside the main function.

**Syntax of goto statement:**

#include <stdio.h>

int main(void)

{

if(condition)

{

goto label;

}

label:

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

{

printf(“%d”,i);

}

//Inside label we can do printf (or) loop (or) function ..anything inside the label we can perform.

}

**EXAMPLE PROGRAM for goto:**

#include <stdio.h>

int main(void)

{

int a=3;

if(a>2)

{

goto label;

}

label:

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

{

printf(“%d”,i);

}

//Inside label we can do printf (or) loop (or) function ..anything inside the label we can perform.

}

EXAMPLE PROGRAM:

#include <stdio.h>

int main(void)

{

int a=3;

for(int i=0;i<=a;i++)

{

goto label;

}

label:

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

{

printf(“%d”,i);

}

//Inside label we can do printf (or) loop (or) function ..anything inside the label we can perform.

}

**OUTPUT:** 01234

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int a=3;

for(int i=0;i<=a;i++)

{

goto label;

}

label:

printf("Work hard and receive everything");

//Inside label we can do printf (or) loop (or) function ..anything inside the label we can perform.

}

**OUTPUT:** Work hard and receive everything

**Note:** If we put goto statement inside a for loop also , it will happen only once.

See the above example.

1. **How many times will the following loop execute?**

**for(j = 1; j <= 10; j = j-1)**

**Ans:** Forever

**Explanation:** Because , j value is always less than 10.

1. **Which statement is used to run a set of another code?**

**Ans:** goto statement

**FUNCTIONS IN C PROGRAMMING:**

There are 2 types of functions in C programming.

1. Standard Library Functions
2. User-defined Functions.

**Standard Library Functions:** Functions which are available inside C programming itself.

1. Namma pana vendiya work ah andha library functions namakaaga pannum.
2. The library functions will do the work which we need to do .
3. These standard library functions will optimize the performance of code for us and saves the development time.
4. For eg., If the user wants to find the square root of a number , then there is a library function available for that. So, to use that library function , we need to import a header file.
5. which is #include <math.h>
6. When we are getting the square root of a number,the format specifier we should use is “%f”, because the square root of a number will definitely be in a decimal point.
7. Only a double datatype value can be passed in a sqrt() function.

**EXAMPLE PROGRAM:**

#include<stdio.h>

#include<math.h>

int main(void)

{

double a = 2.5;

double b = 4;

printf("%f\n",sqrt(a));

printf("%f\n",sqrt(b));

}

**User-defined Functions:** Functions created by the user.

**EXAMPLE PROGRAM 1:**

#include<stdio.h>

#include<math.h>

void addNumber();

int main(void)

{

addNumber();

}

void addNumber()

{

int a=5,b=3;

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

}

**OUTPUT:** 8

**EXAMPLE PROGRAM 2:**

#include<stdio.h>

#include<math.h>

//void addNumber();

int main(void)

{

addNumber();

}

void addNumber()

{

int a=5,b=3;

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

}

**OUTPUT:** Error

**Explanation:** If u are creating a function below the main method, then u must declare that function above the main method itself.Then only it will not show any error.It will run.

Otherwise , it will not run and show the error.

Or else, u can create the user-defined method before the main method itself.

**EXAMPLE PROGRAM 3:**

#include<stdio.h>

#include<math.h>

void addNumber()

{

int a=5,b=3;

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

}

int main(void)

{

addNumber();

}

**OUTPUT:** 8

**Types of User-Defined Functions:**

There are 4 types of user-defined functions in C.

**Type 1: No argument passed, no return value.**

**EXAMPLE PROGRAM 1:**

#include<stdio.h>

#include<math.h>

void addNumber();

int main(void)

{

addNumber();

}

void addNumber()

{

int a=5,b=3;

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

}

**OUTPUT:** 8

**Type 2: No argument passed, but return value.**

**EXAMPLE PROGRAM 2:**

#include<stdio.h>

#include<math.h>

int addNumber();

int main(void)

{

int a = addNumber();

printf("%d",a);

}

int addNumber()

{

int a=5,b=3,c;

c = a+b;

return c;

}

**OUTPUT:** 8

**EXAMPLE PROGRAM 2:**

#include<stdio.h>

#include<math.h>

int addNumber();

int main(void)

{

printf("%d",addNumber());

}

int addNumber()

{

int a=5,b=10;

return a+b;

}

**Type 3: Argument passed, but no return value.**

**EXAMPLE PROGRAM:**

#include<stdio.h>

#include<math.h>

int addNumber();

int main(void)

{

printf("%d",addNumber(3,4));

}

int addNumber(pandiyanstores , baakiyalakshmi)

{

printf("%d", pandiyanstores + baakiyalakshmi);

}

**OUTPUT:** 71

**Explanation:** Why it is giving 71 instead of 7 means, when u r declaring the user-defined function above the main method,there also, we have to mention that there are some arguments are passed inside the function.

So, we should mention the datatypes of those arguments.

We have to mention the datatypes we have passed along with some random parameter name.

**EXAMPLE PROGRAM:**

#include<stdio.h>

#include<math.h>

int addNumber(int x,int y);

int main(void)

{

printf("%d",addNumber(3,4));

}

int addNumber(pandiyanstores , baakiyalakshmi)

{

printf("%d", pandiyanstores + baakiyalakshmi);

}

**OUTPUT:** 71

**Explanation:** Now also, we are getting the wrong output , because since we didn’t return any value of that user-defined function, we cannot call the printf() statement for this function.(user-defined function).

Because it will not return any value,it’s just a function call.

So , how we have to call this function is like:

**EXAMPLE PROGRAM:**

#include<stdio.h>

#include<math.h>

int addNumber(int x,int y);

int main(void)

{

addNumber(3,4);

}

int addNumber(pandiyanstores , baakiyalakshmi)

{

printf("%d", pandiyanstores + baakiyalakshmi);

}

**OUTPUT:** 7

Now we are getting the output correctly.

Explanation: But , now also u r getting some warning because in the user-defined function , u have given int method , so definitely it must return any int value.It can either return the value 0 also. For example, like this..

**EXAMPLE PROGRAM:**

#include<stdio.h>

#include<math.h>

int addNumber(int x,int y);

int main(void)

{

addNumber(3,4);

}

int addNumber(pandiyanstores , baakiyalakshmi)

{

printf("%d", pandiyanstores + baakiyalakshmi);

return 0;

}

**OUTPUT:** 7

Better if u don’t return any value or don’t want to return any value, then simply give void method instead.That is better.

**EXAMPLE PROGRAM:**

#include<stdio.h>

#include<math.h>

void addNumber(int x,int y);

int main(void)

{

addNumber(3,4);

}

void addNumber(pandiyanstores , baakiyalakshmi)

{

printf("%d", pandiyanstores + baakiyalakshmi);

}

**OUTPUT:** 7

**Type 4: argument passed and return value.**

**EXAMPLE PROGRAM:**

#include<stdio.h>

#include<math.h>

int addNumber(int x,int y);

int main(void)

{

printf("%d",addNumber(3,4));

}

int addNumber(pandiyanstores , baakiyalakshmi)

{

return pandiyanstores + baakiyalakshmi;

}

**OUTPUT:** 7

**RECURSION IN C:**

Recursion - A function calling itself.

**EXAMPLE PROGRAM for RECURSION:**

#include<stdio.h>

#include<math.h>

int sum(int x);

int main(void)

{

int a=5;

int result = sum(a);

printf("%d",result);

}

int sum(n)

{

if(n!=0)

{

return n+sum(n-1);

}

else

{

return n;

}

}

**OUTPUT:** 15

**EXAMPLE PROGRAM for RECURSION:**

#include<stdio.h>

#include<math.h>

int sum(int x);

int main(void)

{

int a=5;

int result = sum(a);

printf("%d",result);

}

int sum(n)

{

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

if(n!=0)

{

return n+sum(n-1);

}

else

{

return n;

}

}

**OUTPUT:** 5 4 3 2 1 0 15

**Drawback of Recursion:**

Recursion makes the program performance **slow**.

**Question 1:**

#include <stdio.h>

int main()

{

void foo();

printf("1 ");

foo();

}

void foo()

{

printf("2 ");

}

**OUTPUT:** 1 2

**Explanation:** Because after printing 1 it calls the foo() and prints 2 after it.

**EXAMPLE PROGRAM:**

#include <stdio.h>

void main()

{

m();

}

void m()

{

printf("hi");

m();

}

**OUTPUT:** Infinite hi

**Explanation:** Because m() is called inside the same m(). So it runs infinitely.

**Arrays in C:**

If u want to store multiple datas or multiple values of thhe same type , then array is used.

For eg., if u want to store 10 integers within the same variable , then we can use array.

**EXAMPLE PROGRAM FOR ARRAY:**

#include <stdio.h>

int main()

{

int a[5]={1,24,34,54,23};

a[2] = 56;

printf("%d",a[2]);

}

**OUTPUT:** 56

**EXAMPLE PROGRAM FOR ARRAY:**

#include <stdio.h>

int main()

{

int a[5]={1,24,34,54,23};

a[2] = 56;

printf("%d",a);

}

**OUTPUT:** Error

**Explanation:** Now if we give like this,it will show error because “%d” can print only a single integer.It cannot print a group of integers.

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main()

{

int a[5]={1,24,34,54,23};

a[2] =56;

a[3] =45;

printf("%d\t%d",a[2],a[3]);

}

**OUTPUT:** 56 45

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main()

{

int a[5];

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

{

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

}

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

{

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

}

}

**OUTPUT:** 34 45 56 78 889

34 45 56 78 889

**Multi-Dimensional Arrays in C:**

**EXAMPLE PROGRAM for 2-D Array:**

#include <stdio.h>

int main()

{

int a[5]a[3]; //By typing like this,I can store 15 elements in an array.

}

**EXAMPLE PROGRAM for 3-D Array:**

#include <stdio.h>

int main()

{

int a[5][3]; //By typing like this,in this array, I can store upto 30 elements .

}

a[3][4] can be considered as 3 rows amd 4 columns.

Like,

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

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

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

The data will be in the form of a matrix format.

**How to initialize an 2-D Array?**

#include <stdio.h>

int main()

{

int a[2][3] = {{1,2,3},{-4,4,2}};

int c[2][3][4]; //2 rows.3 columns, and each one has 4 values.

}

int c[2][3][4] can be represented as :

2 rows,3 columns and each data containing 4 values.

{{1,2,3,4},{5,6,7,8},{9,10,11,12}}

{{13,14,15,16},{17,18,19,20},{21,22,23,24}}

**How to get input in 2-D array and print it?**

Since there are 2 dimensions in 2-D array I’m going to use 2 for loops.

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main()

{

int a[2][3];

for(int i=0;i<2;i++)

{

for(int j=0;j<3;j++)

{

printf("a[%d][%d]:",i,j);

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

}

}

for(int i=0;i<2;i++)

{

for(int j=0;j<3;j++)

{

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

}

}

}

**OUTPUT:**

**//scanf**

a[0][0]:14

a[0][1]:45

a[0][2]:74

a[1][0]:75

a[1][1]:54

a[1][2]:20

**//printf**

a[0][0]:14

a[0][1]:45

a[0][2]:74

a[1][0]:75

a[1][1]:54

a[1][2]:20

**How to get input in 3-D array and print it?**

Since there are 2 dimensions in 3-D array I’m going to use 3 for loops.

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main()

{

int a[2][3][2];

for(int i=0;i<2;i++)

{

for(int j=0;j<3;j++)

{

for(int k=0;k<2;k++)

{

printf("a[%d][%d][%d]:",i,j,k);

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

}

}

}

for(int i=0;i<2;i++)

{

for(int j=0;j<3;j++)

{

for(int k=0;k<2;k++)

{

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

}

}

}

}

**OUTPUT:**

**//scanf**

a[0][0][0]:10

a[0][0][1]:14

a[0][1][0]:17

a[0][1][1]:85

a[0][2][0]:96

a[0][2][1]:74

a[1][0][0]:54

a[1][0][1]:24

a[1][1][0]:85

a[1][1][1]:96

a[1][2][0]:54

a[1][2][1]:10

**//printf**

a[0][0][0]:10

a[0][0][1]:14

a[0][1][0]:17

a[0][1][1]:85

a[0][2][0]:96

a[0][2][1]:74

a[1][0][0]:54

a[1][0][1]:24

a[1][1][0]:85

a[1][1][1]:96

a[1][2][0]:54

a[1][2][1]:10

**What are Pointers?**

A pointer is a variable whose value is the address of another variable,

1. e., direct address of the memory location.
2. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is −

**type \*var-name;**

**EXAMPLE FOR POINTERS:**

#include <stdio.h>

int main()

{

int var = 20; /\* actual variable declaration \*/

int \*ip; /\* pointer variable declaration \*/

ip = &var; /\* store address of var in pointer variable\*/

printf("Address of var variable: %x\n", &var );

/\* address stored in pointer variable \*/

printf("Address stored in ip variable: %x\n", ip );

/\* access the value using the pointer \*/

printf("Value of \*ip variable: %d\n", \*ip );

return 0;

}

**OUTPUT:**

Address of var variable: 9bfd930c

Address stored in ip variable: 9bfd930c

Value of \*ip variable: 20

**How to Use Pointers?**

There are a few important operations, which we will do with the help of pointers very frequently.

1. We define a pointer variable,
2. assign the address of a variable to a pointer and
3. finally access the value at the address available in the pointer variable.
4. This is done by using unary operator \* that returns the value of the variable located at the address specified by its operand.

The following example makes use of these operations −

**EXAMPLE FOR NULL POINTER:**

#include <stdio.h>

int main ()

{

int \*ptr = NULL;

printf("The value of ptr is : %x\n", ptr );

return 0;

}

**OUTPUT:**

The value of ptr is : 0

To check for a null pointer, you can use an 'if' statement as follows −

if(ptr) // succeeds if p is not null

if(!ptr) // succeeds if p is null

**EXAMPLE PROGRAM FOR POINTERS:**

#include <stdio.h>

int main()

{

int var = 5;

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

// Notice the use of & before var

printf("address of var: %p", &var);

return 0;

}

**OUTPUT:**

var: 5

address of var: 0x7ffdf90aa284

**Format Specifiers:**

The format specifier **“%p”** is used to represent an address or a pointer.

|  |  |
| --- | --- |
| **Specifier** | **Used For** |
| %p | an address (or pointer) |
| %f | a floating point number for floats |
| %u | int unsigned decimal |
| %e | a floating point number in scientific notation |

## C Pointers

Pointers (pointer variables) are special variables that are used to store addresses rather than values.

You can also declare pointers in these ways.

1. int\* p;
2. int \*p1;
3. int \* p2;
4. int\* p1,p2;

**EXAMPLE PROGRAM:**

#include<stdio.h>

int main()

{

int \*pc;

int \*c;

c = 5;

pc = &c; //0x7fff60f2f238

printf("%d", \*pc); //if u want to print the value , put “%d” format specifier.

}

**Explanation:** Here, the address of c is assigned to the **pc** pointer. To get the value stored in that address, we used **\*pc.**

**Note:** In the above example, **pc** is a pointer, not **\*pc**. You cannot and should not do something like **\*pc = &c;**

By the way,**\*** is called the **dereference operator** (when working with pointers). It operates on a pointer and gives the value stored in that pointer.

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int\* pc,c;

//int c = 5;

c = 5;

pc = &c;

printf("%p\n", pc);

c = 1;

printf("%d\n", c); // Output: 1

printf("%p\n", pc);

printf("%d", \*pc);

}

**OUTPUT:**

0x7ffef80bea3c

1

0x7ffef80bea3c

1

**Explanation:**

**pc** will store the address of the variable .

See, understand one thing… the variable name will not change.

The address of the variable will not change. It is immutable

Only the value it holds can be changed.

Only the value that the variable holds can be changed.

We have assigned the address of c to the **pc** pointer.

Then, we changed the value of **c** to **1**. Since **pc** and the address of c is the same, **\*pc** gives us 1.

For eg., PDKV ngra college name um change pana mudiyadhu.

PDKV college irukura location(address) ah yum maatha mudiyadhu.

But anga vandhu pora students ah .nama maathikalam.. namaku yetha maari epo venalum maathikalam.

**EXAMPLE PROGRAM:**

#include <stdio.h>

int main(void)

{

int\* pc, c;

c = 5;

pc = &c;

\*pc = 1; //c=1 (Since \*pc = c)

printf("%d\n", \*pc); // Ouptut: 1

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

}

**OUTPUT:**

1

1

**Explanation:** We have assigned the address of c to the pc pointer.

Then, we changed \*pc to 1 using \*pc = 1;. Since pc and the address of c is the same, c will be equal to 1.

**EXAMPLE PROGRAM:**

#include<stdio.h>

int main()

{

int\* pc, c, d; //int c; int d;

c = 5;

d = -15;

pc = &c;

printf("%d\n", \*pc); // Output: 5

pc = &d;

printf("%d", \*pc); // Ouptut: -15

}

**OUTPUT:**

5

-15

**Explanation:**

Initially, the address of c is assigned to the pc pointer using pc = &c;. Since c is 5, \*pc gives us 5.

But after that, the address of d is assigned to the pc pointer using pc = &d;. Since d is -15, \*pc gives us -15.

**Example: Working of Pointers**

#include <stdio.h>

int main()

{

int\* pc, c;

c = 22;

printf("Address of c: %p\n", &c);

printf("Value of c: %d\n\n", c); // 22

pc = &c;

printf("Address of pointer pc: %p\n", pc);

printf("Content of pointer pc: %d\n\n", \*pc); // 22

c = 11;

printf("Address of pointer pc: %p\n", pc);

printf("Content of pointer pc: %d\n\n", \*pc); // 11

\*pc = 2; //\*pc =c (c=2)

printf("Address of c: %p\n", &c);

printf("Value of c: %d\n\n", c); // 2

return 0;

}

**OUTPUT:**

Address of c: 0x7ffec84a794c

Value of c: 22

Address of pointer pc: 0x7ffec84a794c

Content of pointer pc: 22

Address of pointer pc: 0x7ffec84a794c

Content of pointer pc: 11

Address of c: 0x7ffec84a794c

Value of c: 2

**Common mistakes when working with pointers:**

int c, \*pc;

// pc is address but c is not

pc = c; // Error

// &c is address but \*pc is not

\*pc = &c; // Error

// both &c and pc are addresses

pc = &c; // Not an error

// both c and \*pc are values

\*pc = c; // Not an error

**Example of pointer syntax often finds confusing:**

#include <stdio.h>

int main()

{

//int \*p,c;

int c = 5;

int \*p = &c; //p=&c

printf("%d", \*p); // 5 (i.e) \*p=c;

return 0;

}

**OUTPUT:**

5

**Explanation:**

**Why didn't we get an error when using**  **int \*p = &c;**?

It's because int \*p = &c; is equivalent to

int \*p:

p = &c;

In both cases, we are creating a pointer p (not \*p) and assigning &c to it.

To avoid this confusion, we can use the statement like this:

**int\* p = &c;**

**RELATIONSHIP BETWEEN ARRAYS AND POINTERS:**

**EXAMPLE PROGRAM FOR ARRAYS AND POINTERS:**

#include <stdio.h>

int main()

{

int x[4];

int i;

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

{

printf("&x[%d] = %p\n", i, &x[i]);

}

printf("Address of array x: %p", x);

return 0;

}

**OUTPUT:**

&x[0] = 0x7ffdf260f6a0

&x[1] = 0x7ffdf260f6a4

&x[2] = 0x7ffdf260f6a8

&x[3] = 0x7ffdf260f6ac

Address of array x: 0x7ffdf260f6a0

**Explanation:**

There is a difference of 4 bytes between two consecutive elements of array x. It is because the size of int is 4 bytes (on our compiler).

Notice that, the address of &x[0] and x is the same. It's because the variable name x points to the first element of the array.

From the above example, it is clear that

* &x[0] is equivalent to x. And, x[0] is equivalent to \*x.

Similarly,

* &x[1] is equivalent to x+1 and x[1] is equivalent to \*(x+1).
* &x[2] is equivalent to x+2 and x[2] is equivalent to \*(x+2).

Basically,

* &x[i] is equivalent to x+i and x[i] is equivalent to \*(x+i).

**EXAMPLE PROGRAM FOR POINTERS AND ARRAYS:**

#include <stdio.h>

int main()

{

int i, x[6], sum = 0;

printf("Enter 6 numbers: ");

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

{

// Equivalent to scanf("%d", &x[i]);

scanf("%d", x+i);

// Equivalent to sum += x[i] bcoz x[0]=\*x and \*x = \*(x+0)

sum += \*(x+i); //sum=sum+ \*(x+i)

}

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

return 0;

}

**OUTPUT:**

Enter 6 numbers: 1 2 3 4 5 6

Sum = 21

**EXAMPLE PROGRAM 2: Arrays and Pointers**

#include <stdio.h>

int main()

{

int x[5] = {1, 2, 3, 4, 5};

int\* ptr;

// ptr is assigned the address of the third element

ptr = &x[2];

printf("%p\n",ptr);

printf("\*ptr = %d \n", \*ptr); // 3

printf("\*(ptr+1) = %d \n", \*(ptr+1)); // 4

printf("\*(ptr-1) = %d", \*(ptr-1)); // 2

return 0;

}

**OUTPUT:**

0x7ffc121a8038

\*ptr = 3

\*(ptr+1) = 4

\*(ptr-1) = 2