Array in C++

In C++, an array is a data structure that is used to store multiple values of similar data types in a contiguous memory location.

Array Declaration in C++

In C++, we can declare an array by simply specifying the data type first and then the name of an array with its size.

data\_type array\_name[Size\_of\_array];

Take Inputs from User and Store Them in an Array

#include <iostream>

using namespace std;

int main()

{

int numbers[5];

cout << "Enter 5 numbers: " << endl;

// store input from user to array

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

cin >> numbers[i];

}

cout << "The numbers are: ";

// print array elements

for (int n = 0; n < 5; ++n) {

cout << numbers[n] << " ";

}

return 0;

}

============================================================================

#include <iostream>

using namespace std;

int main()

{

int numbers[5];

cout << "Enter 5 numbers: " << endl;

// store input from user to array

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

cin >>\*(numbers+i);

}

cout << "The numbers are: ";

// print array elements

for (int n = 0; n < 5; ++n) {

cout << \*(numbers+n) << " ";

}

return 0;

}

================================================================================

The C++ Program to Illustrate How to Find the Size of an Array

C++

#include <iostream>

using namespace std;

int main()

{

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

// Size of one element of an array

cout << "Size of arr[0]: " << sizeof(arr[0]) << endl;

// Size of array 'arr'

cout << "Size of arr: " << sizeof(arr) << endl;

// Length of an array

int n = sizeof(arr) / sizeof(arr[0]);

cout << "Length of an array: " << n << endl;

return 0;

}============================================================================================

// C++ Program to Illustrate that Array Name is a Pointer that Points to First Element of the Array

#include <iostream>

using namespace std;

int main()

{

// Defining an array

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

// Define a pointer

int\* ptr = arr;

// Printing address of the arrary using array name

cout << "Memory address of arr: " << &arr[0] << endl; //100

// Printing address of the array using ptr

cout << "Memory address of arr: " << ptr << endl; //100

return 0;

}

===========================================================================

// C++ Program to Print Array Elements without Indexing

#include <iostream>

using namespace std;

int main()

{

// Define an array

int arr[] = { 11, 22, 33, 44 };

// Print elements of an array

cout << "first element: " << \*arr << endl; //\*(arr+0)

cout << "Second element: " << \*(arr + 1) << endl;

cout << "Third element: " << \*(arr + 2) << endl;

cout << "fourth element: " << \*(arr + 3) << endl;

return 0;

}

==============================================================================================Assignment

1)Find out maximum element in an array

2)calculate average of all elements of an array using pointer

3)copy elements of 1 array into another using pointer

===================================================================

Two Dimensional Array in C++

In C++, a two-dimensional array is a grouping of elements arranged in rows and columns. Each element is accessed using two indices: one for the row and one for the column, which makes it easy to visualize as a table or grid.

2D ARRAY

//name of 2d array itself is a constant pointer pointing to base address of an array

syntax:

datatype arrayname[row][col];

int a[3][3]={1,2,3,4,5,6,7,8,9};

//no of elements=row\*col-->3\*3-->9 elements

printf("%d",sizeof(a));//9\*4-->36 bytes

printf("%d",sizeof(a[0][0]));//4 bytes

ADDRESS

1)SUBSCRIPT NOTATION(&arrayname[row][col]);

&a[0][0]-->100

&a[0][1]-->104

&a[1][1]-->116

&a[2][2]--->132

2)POINTER NOTATION --->\*(arrayname/pointername +row)+col;

\*(a+0)+0-->100

\*(a+0)+1-->104

\*(a+1)+0 -->112

\*(a+1)+1 --->116

\*(a+2)+2-->132

&a[0][0]===\*(a+0)+0 ------------->100

&a[1][2]===\*(a+1)+2 ------------->120

&a[i][j]===\*(a+i)+j-------------->address

VALUE

1)SUBSCRIPT NOTATION(arrayname[row][col]);

a[0][0]-->1

a[0][1]-->2

a[1][1]-->5

a[2][2]--->9

2)POINTER NOTATION --->\*(\*(arrayname/pointername +row)+col);

\*(\*(a+0)+0)-->\*100-->1

\*(\*(a+0)+1)-->\*104-->2

\*(\*(a+2)+2)-->\*132-->9

a[0][0]===\*(\*(a+0)+0) ------------->1

a[i][j]===\*(\*(a+i)+j)-------------->value

// c++ program to illustrate the two dimensional array

#include <iostream>

using namespace std;

int main()

{

// Declaring 2D array

int arr[3][3];

// Initialize 2D array using loop

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

for (int j = 0; j<=2; j++) {

cin>>arr[i][j];

}

}

// Printing the element of 2D array

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

for (int j = 0; j<=2; j++) {

cout<<arr[i][j];

}

}

return 0;

}

// c++ program to illustrate the two dimensional array

#include <iostream>

using namespace std;

int main()

{

// Declaring 2D array

int arr[3][3];

// Initialize 2D array using loop

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

for (int j = 0; j<=2; j++) {

cin>>\*(\*(arr+i)+j);

}

}

// Printing the element of 2D array

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

for (int j = 0; j<=2; j++) {

cout<<\*(\*(arr+i)+j);

}

}

return 0;

}

===============================================================================

Assignment

1)addition of 2 matrix

2)transpose of a matrix

3)square of all elements of 2d array

=============================================================

Default Arguments

==>A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the calling function doesn’t provide a value for the argument..

Following are the rules of declaring default arguments -

==>The values passed in the default arguments are not constant. These values can be overwritten if the value is passed to the function. If not, the previously declared value retains.

==>All the values that will be given as a default value will be on the right.

#include <iostream>

using namespace std;

void add(int=1,int=2,int=3,int=4);

int main()

{

add();//10

add(10);//19

add(10,20);//37

add(10,20,30); //64

add(10,20,30,40);//100

return 0;

}

void add(int a,int b,int c,int d)

{

cout<<a+b+c+d<<endl;

}

#include <iostream>

using namespace std;

void display(char = '\*', int = 39);

int main() {

int count = 50;

cout << "No argument passed: ";

display();

cout << "First argument passed: ";

display('#');

cout << "Both arguments passed: ";

display('$', count);

}

void display(char c, int count) {

for(int i = 1; i <= count; ++i)

{

cout << c;

}

cout << endl;

}

==========================================================

#include <iostream>

using namespace std;

void display(char c='\*',int count=39)

{

for(int i = 1; i <= count; ++i)

{

cout << c;

}

cout << endl;

}

int main() {

int count = 50;

cout << "No argument passed: ";

display();

cout << "First argument passed: ";

display('#');

cout << "Both arguments passed: ";

display('$', count);

}

============================================

#include<iostream>

using namespace std;

void func(int a, bool flag = true)

{

if (flag == true )

{

cout<< "Flag is true. a = " << a;

}

else

{

cout<< "Flag is false. a = " << a;

}

}

int main()

{

func(200, false);

}

Keypoints:Default arguments are different from constant arguments as constant arguments can’t be changed whereas default arguments can be overwritten if required.

==============================================

If we are defining the default arguments in the function definition instead of the function declaration, then the function must be defined before the function call.

g

#include <iostream>

using namespace std;

int sum(int x, int y, int z = 5, int w = 10)

{

return (x + y + z + w);

}

int main()

{

cout << sum(10, 15) << endl;

cout << sum(10, 15, 25) << endl;

cout << sum(10, 15, 25, 30) << endl;

}

=======================================================

points to remember

1)Once we provide a default value for a parameter, all subsequent parameters must also have default values. For example,

// Invalid

void add(int a, int b = 3, int c, int d);

// Invalid

void add(int a, int b = 3, int c, int d = 4);

// Valid

void add(int a, int c, int b = 3, int d = 4);

2)Default values shld be given from trailing ends.i.e from right to left

void add(int a,int b=3,int c=3,int d=4);//valid compulsary we have to pass 1 argument from function call

void add(int a=4,int b=3,int c=3,int d);//invalid

void add(int a=1,int b,int c=3,int d=4);//invalid

===========================================================================================

====================================================================================================================

Array of Integer pointers

#include <iostream>

using namespace std;

int main()

{

int ptr1[5]; // integer array declaration

int \*ptr2[5]; // integer array of pointer declaration

cout << "Enter five numbers :" <<endl;

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

{

cin >> ptr1[i];

}

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

{

ptr2[i]=&ptr1[i];

}

cout << "The values are" << endl;

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

{

cout << \*ptr2[i] << endl;

}

cout<<"sizeof"<<sizeof(ptr1);

cout<<"sizeof"<<sizeof(ptr2);

}

===================================================================

#include <iostream>

using namespace std;

int main()

{

int ptr1[5]; // integer array declaration

int \*ptr2[5]; // integer array of pointer declaration

cout << "Enter five numbers :" <<endl;

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

{

cin >> ptr1[i];

}

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

{

ptr2[i]=&ptr1[i];

}

cout << "The values are" << endl;

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

{

cout << \*ptr2[i] << endl;

}

\*ptr2[4]=50;

cout<<\*ptr2[4];

cout << "The values are" << endl;

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

{

cout << \*ptr2[i] << endl;

}

int x=ptr1[4]==\*ptr2[4];

cout<<x;

}

============================================================

================================================================================================================

Passing and Returning Objects in C++

In C++ we can pass class’s objects as arguments and also return them from a function the same way we pass and return other variables.

Passing an Object as argument

To pass an object as an argument we write the object name as the argument while calling the function the same way we do it for other variables.

Syntax:

function\_name(object\_name);

Returning Object as argument

Syntax:

return object\_name;

#include<iostream>

using namespace std;

class temp

{

int a;

public:

void get();

temp add(temp & b1) //This function will take object as arguments and return object

{

temp aobj3;

aobj3.a= this->a + b1.a;

return aobj3;

}

void show()

{

cout<<"value is "<<a;

}

};

void temp:: get()

{

cin>>this->a;

}

int main()

{

temp aobj1,aobj2;

aobj1.get();

aobj2.get();

temp aobj4=aobj1.add(aobj2);

aobj4.show();

}

Exercise 1:

Which of the following is true about this pointer?

(A) It is passed as a hidden argument to all function calls

(B) It is passed as a hidden argument to all non-static function calls

(C) It is passed as a hidden argument to all static functions

(D) None of the above

Answer: (B)

===========================================================================================

Array of Objects

https://photos.app.goo.gl/CjCcfL8fMHLxqNvj9

==>In C++, an array of objects is a collection of objects of the same class type that are stored in contiguous memory locations.

==>Since each item in the array is an instance of the class, each one's member variables can have unique value. This makes it possible to manage and handles numerous objects by storing them in a single data structure and giving them similar properties and behaviours.

syntax:

classname objectname[arraysize];

Example:

Storing more than one Employee data. Let’s assume there is an array of objects for storing 30 employee's data.

#include<iostream>

using namespace std;

class Employee

{

int id;

float sal;

public:

void getdata();

void putdata();

};

void Employee::getdata()

{

cout << "Enter Id : ";

cin >> id;

cout << "Enter sal : ";

cin >> sal;

}

void Employee::putdata()

{

cout << id << " ";

cout << sal << " ";

cout << endl;

}

int main()

{

// This is an array of objects having maximum limit of 3 Employees

Employee emp[3];

int i;

// Accessing the function

for(i = 0; i < 3; i++)

emp[i].getdata();

cout << "Employee Data - " << endl;

// Accessing the function

for(i = 0; i < 3; i++)

emp[i].putdata();

}

Advantages of Array of Objects:

==>The array of objects represent storing multiple objects in a single name.

==>In an array of objects, the data can be accessed randomly by using the index number.

==>Reduce the time and memory by storing the data in a single variable.

#include<iostream>

using namespace std;

class item

{

char name[30];

int price;

public:

void getItem();

void printItem();

};

void item::getitem()

{

cout << "Item Name = ";

cin >> name;

cout << "Price = ";

cin >> price;

}

void item ::printitem()

{

cout << "Name : " << name <<

"\n";

cout << "Price : " << price <<

"\n";

}

const int size = 3;

int main()

{

item t[size];

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

{

cout << "Item : " << (i + 1) << "\n";

t[i].getitem();

}

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

{

cout << "Item Details : " << (i + 1) << "\n";

t[i].printitem();

}

}

=====================

Assignment

store details of 10 books in an array

==============================================================================================

Constant Objects

==>To create constant object use const keyword.

==>When we create an object using the const keyword, the value of data members can never change till the life of the object in a program. The const objects are also known as the read-only objects.

==>constant objects invokes only constant member function. But a non constant object can invoke both i.e constant as well as non constant function.

==>The data members of a constant object cannot accessed through any non const member function.

Syntax

const class\_name obj\_name;

ex: const Complex c1(1,2);

Example: Let's create a program to use the constant objects in the C++ programming language.

#include <iostream>

using namespace std;

class ABC

{

public:

int A;

ABC ()

{

A = 10;

}

};

int main ()

{

const ABC obj;

cout << " The value of A: " << obj.A << endl;

// obj.A = 20; // It returns a compile time error

return 0;

}

Output

The value of A: 10

In the above program, we assign the value of A is 10, the compiler prints "The value of A: 10", and when we assign the value of A to 20, the object of the class returns the compile time error in the program.

=================================================================================================

Constant Member function

==>To make a member function constant,use const keyword.

==>We can create a constant member function of a class by adding the const keyword after the name of the member function.

Syntax

return\_type mem\_fun() const

{

}

In the above syntax, mem\_fun() is a member function of a class, and the const keyword is used after the name of the member function to make it constant.

==>Those functions that do not alter the values of the data members, or the functions that are 'read only' can be

made constant.

==>If a function is declared as const and the implementation of that function tries to change the object by changing the value of any of its members,the compiler flags an error.

Example: Program to use the const keyword with the member function of class

class ABC

{

public:

int A;

void fun () const

{

A = 0; // it shows compile time error

}

};

int main ()

{

ABC obj;

obj.fun();

return 0;

}

Output

The above code throws a compilation error because the fun() function is a const member function of class ABC, and we are trying to assign a value to its data member 'x' that returns an error.

// Constant member function defined outside the class

#include <iostream>

using namespace std;

class Demo {

int x;

public:

void set\_data(int);

int get\_data() const;

};

void Demo::set\_data(int a)

{

x = a;

}

int Demo::get\_data() const

{

return x;

}

main()

{

Demo d;

d.set\_data(10);

cout << d.get\_data();

return 0;

}

Output

10

// C++ program to demonstrate that const functions can be called by non const objects

#include <iostream>

using namespace std;

class Test {

int value;

public:

Test(int v = 0) { value = v; }

int getValue() const { return value; }

};

int main()

{

Test t(20);

cout << t.getValue();

return 0;

}

Output

20

// C++ program that demonstrate that non-const functions can not be called by const objects

#include <iostream>

using namespace std;

class Test {

int value;

public:

Test(int v = 0) { value = v; }

// non const member function

int getValue() { return value; }

};

int main()

{

// const object

const Test t;

cout << t.getValue();

return 0;

}

//ERROR

===============================================================================

Constant function Arguments

We can declare the function arguments as the constant argument using the const keyword. And if the value of function arguments is declared const, it does not allow changing its value.

Syntax

return\_type fun\_name (const int x)

{

}

The fun\_name() function contains a const argument whose value can never be changed once it defines in the program.

Example: Let's consider an example to use the const keyword with function arguments in the C++ programming language.

#include <iostream>

using namespace std;

int Test (const int num)

{

// if we change the value of the const argument, it throws an error.

//num = num + 10;

cout << " The value of num: " << num << endl;

}

int main ()

{

Test(5);

}

Output

The value of num: 5

In the above program, the num is a constant argument, and we cannot update the num value. If we update the value of the num variable, it returns the compile-time error.

FAQ on constant object:

Q1. Can const objects of a class call non-const member functions?

Answer:

No,an object declared as const cannot be modified and hence, can invoke only const member functions as these functions ensure not to modify the object.

Q2. Can non-const objects of a class call const member function?

Answer: yes

When a function is declared as const, it can be called on any type of object.

Note: A constant member function can be overloaded with a function of same name.

ex:

void func() const;

void func();

class temp

{

public:

void show();

void show() const;

};

void temp::show()

{

cout<<"in non constant show";

}

void temp::show()const

{

cout<<"in constant show";

}

int main()

{

temp t;

t.show();

}

In this case, constness of the object will determine which functions should be invoked.

Note:Constructors and Destructors cannot be made constants.

===================================================================================

Dynamic memory allocation

==>mDMA allows us to allocate memory dynamically(runtiemory allocated at compile time leads to either memory wastage or memory required.

==>for proper memory management go for DMA.

me) on heap section

for allocating memory dynamically two functions should be used

1)malloc()

2)calloc()

declared in stdlib.h

void pointer-->void\*

void pointer is a generic pointer which can store or can return the address of any type of variable.

#include<stdio.h>

int main()

{

float a=10;

void\* b=&a;

}

printf("%d",sizeof(void\*));//4 bytes on 32 bit and 8 bytes on 64 bits

#include<stdio.h>

int main()

{

show();

}

void\* show()//show() can return the address of any type of variable

{

int a=10;

return &a;

}

malloc()

malloc() is used to allocate memory dynamically on heap section..

syntax..

datatype \* pointername=void\* malloc(n \* sizeof(datatype));

n-->no of elements to be accepted at runtime

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

int\* ptr =(int\*) malloc(20);

=return base address of an array-->100-->int\*

#include<stdio.h>

#include<stdlib.h>

int main()

{

int\* ptr;

int n,i;

printf("enter no of elements\n");

scanf("%d",&n);

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

printf("accept elements\n");

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

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

printf("display elements\n");

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

printf("%d\t",ptr[i]);//\*(ptr+i)

free(ptr);//to deallocate the memory on heap section which is allocated by malloc/calloc()

}

#include<stdio.h>

#include<stdlib.h>

int main()

{

char\* ptr;

int n,i;

printf("enter no of characters in ur name\n");

scanf("%d",&n);

ptr=(char\*) malloc((n\*sizeof(char))+1);

printf("accept name\n");

scanf("%s",&ptr[0]);//(ptr+0),(0+ptr)

printf("display name\n");

printf("%s",&ptr[0]);//(ptr+0),(0+ptr)

free(ptr);//to deallocate the memory

}

========================================================================