C++ Basics

Created	@May 25, 2022
Created by	
Tags	Education
Property	

The difference between continue and break statement

Inside a for or a while loop, if you write "continue" then it will not execute the rest of the loop below the continue statement and it will go directly to the top of the loop and then execute the loop again.

If the "continue" is replaced by the "break" statement then after the break statement it will exit the loop entirely.

Data Structures in C++

1. Built in C style arrays
2. The array class
3. The vector class

In C++ you cannot change the size of the array that once you specified. There is **no appending the array in the C++**

The size of the array has to be known at the compile time. Hence it cannot be entered by the used after the program is compiled. The reason for this is that ,the memory location for all the elements in the array are continuous.

The indexing of the array starts with 0.

the memory allocated to the array elements is contiguous i.e. it is the memory locations one after the another.

Following code creates a simple array and iterates through the elements of the array.

Note: the second type of for loop used in this example is called a “range based” for loop.

```
#include<iostream>

using namespace std;

int main()
{
  cons int AS = 5; // Using a constant int to define the size of the array
  int myArray[AS]; // Defining the array

  string name[4]{"Shubham", "Amita", "Abhay", "Deshpande"}; /*This is one way to initializ
                                                            the array size and the element
                                                            in a single line.*/

  myArray[0] = 1; // Definig the individual elements in the array
  myArray[1] = 2;
  myArray[2] = 3;
  myArray[3] = 4;
  myArray[4] = 5;

  for (int i =0; i<=AS; i++) // For loop for interating through the array
  {
      cout<<myArray[i]<<endl;
  }

  /*for (int i =0; i<=4; i++) // For loop for interating through the array
  {
      cout<<name[i]<<endl;
  }*/

/* The other way of iterating through the array is by using the advanced for loop which
focuses on the elements of the array rather than the indexes.*/
```

```
  for(string name : names) // ':' means that for every 'names' inside 'name'
  {
    cout<<names<<endl;
  }
  return 0;
}
```

Run the program and find the fault in the program

array class

it is a sequence container

the following program shows hoe to use the array class and read the following link to get more information regarding the array class and the Standard Template Library in C++

Array class in C++ - GeeksforGeeks

The introduction of array class from C++11 has offered a better alternative for C-style arrays. The advantages of array class over C-style array are :- Array classes knows its own size, whereas C-style arrays lack this property. So when passing to functions, we don't need to pass size of Array as a separate parameter.

https://www.geeksforgeeks.org/array-class-c/

```
#include <iostream>
#include<array>

using namespace std;


int main()
{
    const int AS = 9;

    /*This progra uses a concept called class arrays in c++. This tyoe of an array is similar to the built in type of an array. But has som
    functionalities to maje it more robust. This type of arrays are covered under the Standard Template Library STL of C++. For more inform
    documentation in the link
    https://www.geeksforgeeks.org/array-class-c/ */


    /* THE TASK OF THE EXERCISE IS TO CREATE AN ARRAY CLASS THAT CONTAINS THE ELMENTS SUCH THAT ELMENT = ELMENT*2 */

    // defining the array class


    array<int, 10> ar = {};

    for (int i = 0; i <= ar.size(); i++) // This is how an array class is defined.
    {
        ar.at(i) = i * 2;  /*This is an advantage of using STL and array class. It has various functions. such as .at(i) fetches the elemen
                           this is simply not possible in built in array. */

        cout << ar.at(i) << endl;

    }

    return 0;


}
```

Vector classes in C++

This is another example of the storage type that is stored in the standard template library (STL) in the C++.

Vectors can be thought of as mutable arrays. Which means you can add or remove the elements at the end, beginning or in the middle of the vector.

For more information regarding the operations that are available in the STL for the vectors, read the following link.

Vector in C++ STL - GeeksforGeeks

Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. In vectors, data is inserted at the end.

https://www.geeksforgeeks.org/vector-in-cpp-stl/

sample code for the vectors

```
#include <iostream>
#include<vector>
#include<string>

using namespace std;

int main()
{
    // define a vector of integers without any size
    vector<int> someVec;

    // define a vector of strings with a fixed size
    vector<string> anotherVec(3);

    // push_back() is used in the STL to append the currevt vector by the elements specified in the brackets.
    someVec.push_back(1);
    someVec.push_back(2);
    someVec.push_back(3);

    //printing the size of the someVec after the addition of the elements
    cout << "Size of the someVec " << someVec.size() << endl;

    //adding the strings in the second vector in the same way as adding the elements in an array
    anotherVec[0] = "shubham";
    anotherVec[1] = "abhay";
    anotherVec[2] = "amita";

    //using push_back on anotherVec
    anotherVec.push_back("deshpande");

    //printing the both vectors to see the results
    for (int val : someVec)
    {
        cout << val << endl;

    }

    cout << endl;

    for (string str : anotherVec)
    {
        cout << str << endl;

    }

    cout << endl;

    // return the elements at the front and the back of the vector by using .front() and .back() methods
    cout <<"element in the front: " << anotherVec.front() << endl;
    cout << endl;
    cout << "element in the back: " << anotherVec.back() << endl;
    cout << endl;



    /*the std::list::pop_back() method is used to remove the element at the end of the vector. (Should not be used with the cout or endl)

    std::list::begin() is used to index or point to the element that is at the brginning of the vector.

    std::list::insert() is used to insert an element in the vector. This method takes two arguments, 1st is the index at which the element
    and the second is the element that is to be inserted in the list. */

    anotherVec.pop_back(); //remove the last element of the vector
    anotherVec.insert(anotherVec.begin(), "baba");//insert a new element in the front
```

```
    cout << "now the fist element is: " << anotherVec.front() << endl;
    cout << endl;
    cout << "the last element is: " << anotherVec.back() << endl;


    return 0;
}
```

Multidimensional Arrays

having rows and columns

refer the following code for the example.

```
#include <iostream>
using namespace std;


int main()
{
    // Create a 2D array, having 2 rows and 3 columns
    int numarr[2][3]{
        {1,2,3},
        {4,5,6}
    };

    //printout the element in the first row and the third column
    cout << numarr[0][2] << endl;

    //change the value of the second row and the third element. Print the results
    numarr[2][3] = 14;
    cout << numarr[2][3] << endl;

    //print the elements of the array in reverse using nested for loop
    for (int i =1 ; i >= 0; i-- )
    {
        for (int j=2; j >= 0; j-- )
        {

            cout << numarr[i][j]<<" ";

        }
        cout << endl;
    }

    return 0;
}
```

multidimensional arrays can also be defined using STL. Read the following link for better explanation.

Multidimensional std::array

In C++, how do I create a multidimensional std::array? I've tried this: std::array<std::array<int, 3>, 3> arr = {{5, 8, 2}, {8, 3, 1}, {5, 3, 9}}; But it doesn't work. What am I doing...

https://stackoverflow.com/questions/17759757/multidimensional-stdarray

Code for the last project of the vectors and arrays

Accept 5 numbers from the user, store them in an array or a vector and print the twice the input numbers

```
// ArrayData.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include <iostream>
#include<vector>

using namespace std;


int main()
{
```

```
    // Define an integer which will be held to temporarily hold the value given by the user.

    int num;

    // Define an array in order to store the 5 numbers
    vector<int> MyArray;

    //For loop for accepting the inputs from the user and storing them in array
    for (int i=0; i <= 4; i++)
    {
        cout << "Enter the " << i << "th number in an array." << endl;
        cin >> num;

        MyArray.push_back(num);

    }


    // Print twice the inputed number as an output
    for (int j = 0; j <= MyArray.size(); j++)
    {
        cout << "Twice the " << j << "th number is: " << MyArray.at(j) * 2<< endl;
        cout <<endl ;

    }


    return 0;
}
```

Accept an arbitrary number of input from the user. Store them in an vector. Print twice the numbers after the user inputs a negative number.

```
#include <iostream>
#include<vector>

using namespace std;


int main()
{
    //Specify an inter that will be used to enter the elements in the vector.
    int num;

    //create a vector of unspecified length
    vector<int>MyVec;

    /* Using do-while loop to check the condition if the entered number is positive or negative. If it is positive add it in the vector. If
    exit the loop and print the twice the all the elements in the vector. */
    do
    {
        //Accept the input from the user.
        cout << "Enter the number to be added in the vector. If the number is negative, then program will exit the loop !!!!" << endl;
        cin >> num;

        // Upend the vector with the input.
        MyVec.push_back(num);
    }
    while (num >= 0);

    // After exiting the loop, print twice of all the elements in the vector.
    cout << "The double of all the elements of the vector are: " << endl;

    for (int i =0; i<MyVec.size(); i++)
    {
        cout << MyVec.at(i) * 2 << endl;
    }

    return 0;
}
```

We can collect two values simultaneously, store them in two parallel arrays and use these arrays.

We have demonstrated this with by collecting the names of the people and their corresponding weights

```
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<vector>

using namespace std;

int main()
{
    // Defining the number of prople
    const int num_people = 5;

    //Defining the vectors to store the weights and the names
    vector<int>weights;
    vector<string> names;


    // Defining the temporary weights and name

    int tempWeight;
    string tempName;

    // Defining the for loop for accepting the weights of the people
    for(int i = 0; i<= num_people; i++)
    {
        cout << "Enter the person's full name"<< endl;
        getline(cin, tempName);

        cout << "Please enter"<<tempName<<"'s weight"<<endl;
        cin>>tempWeight;

        cin.get(); //consume a newline character

        names.push_back(tempName);
        weights.push_back(tempWeight);

    }

    cout<<endl;

    for(int j = 0 ; j<num_people; j++)
    {
        cout<<names[j]<<" Weighs "<<weights[j]<<" lbs."<<endl;
    }

    return 0;

}
```

Functions

Re-useable named blocks of code

We can put entire function above the main() or we can put the prototype above the main() and put the main function below it.

```
#include<iostream>

using namespace std;

/* For the very introduction we will write a function and above we will define a function which will later be called in main.
In Cpp it is considered a good practice to let main() be the first function in the program. But, by law, we also need to define
something before we actually use it. So we will need to use function prototype for this issue. */

// Defining the prototype for the function. Which is basically function header. It is like variable. We need to define it before we use it.

void printMyname();

int main()
{
    printMyname();

    return 0;
}


void printMyname()
```

```
{
    cout<<"Hello, My Name is Shubham Abhay Deshpande"<<endl<<"This line was printed by the function!!!!"<<endl;

}
```

The functions with multiple return types and the arguments are demonstrated as below

Difference between arguments and parameters:

Arguments **are any data type that we use** ***when we actually call a function***

Parameters **are the data types that we use** ***when we define the function.***

Parameters are placeholders that will receive the values in form of arguments

we can pass one function as an argument to another function. It is as shown in the example below.

```
# include<iostream>

using namespace std;

/* A function is defined by it's parameter type and if the function returns any value.
The functions prototypes which are defined below take different parameters and may or
maynot return a value

The functions with 'void' are sometimes called vlue returing. And functions with return type
other than void are called paramerterized. */

void printHello();
void printNumber(int a);
int giveMe10();
int addThese(int num1, int num2);


int main()
{
    printNumber(100);
    printHello();

    int c = 129;
    int d = 430;

    int someVal = addThese(c, d);
    cout<<someVal<<endl;

    int var = giveMe10();
    cout<<var<<endl;

    //we can pass one function as a parameter to another function.

    int x = 500;
    int y = 200;

    printNumber(addThese(x,y));

    return 0;
}

void printHello()
{
    cout<<"hello there"<<endl;
}

void printNumber(int a)
{
    cout<<"The number is: "<<a<<endl;
}

int giveMe10()
{
    return 10;
}

int addThese(int num1, int num2)
{
```

```
    int result = num1 + num2;

    return result;
}
```

Pass-by-value and Pass-by reference

There are two ways in which we can pass a parameter to the function. One is pass by value which is the default setting for the function and other is pass by reference.

1. Pass-by-value

Following is the code and output for passing a argument to the function as a value.

```
#include<iostream>

using namespace std;

void valueChanged1(int someNum);

int main()
{
    int mynum =12 ;
    cout<<"Before calling function valueChanged 1 called, mynum is : "<<mynum<< endl;

    valueChanged1(mynum);

    cout<<"After calling function valueChanged1, mynum is : "<<mynum<<endl;

    return 0;


}

void valueChanged1(int someNum)
{

    someNum = 100;

    cout<<"someNum in function valueChanged1 is : "<<someNum<<endl;

}
```

output for pass by value:

```
Before calling function valueChanged 1 called, mynum is : 12
someNum in function valueChanged1 is : 100
After calling function valueChanged1, mynum is : 12
```

Important thing to note here:

someNum is passed as a parameter to the function valueChange1.

1. before calling valueChange1, the variable mynum points to the memory location containing number 12.
2. when mynum is passed as a argument to the function changeValue1, the value 12 is copied in someNum.
3. But since someNum's value is changed inside the function valueChange1, it prints out 100.
4. Since mynum and someNum are two different variables (technically someNum is a parameter that acts like a variable inside function valueChange1) the value of munum is never changed. And that is why it still prints out 12 in the last line of the output.

Now, the question is, how to change the value of the variable, inside the function?

output for pass by reference:

```
#include<iostream>

using namespace std;

void valueChanged1(int someNum);

void valueChanged2(int& someNum);

int main()
{
    int mynum =12 ;
    cout<<"Before calling function valueChanged 1 called, mynum is : "<<mynum<< endl;

    valueChanged1(mynum);

    cout<<"After calling function valueChanged1, mynum is : "<<mynum<<endl;

    valueChanged2(mynum);

    cout<<"After calling function valueChanged2 , mynum is : "<<mynum<<endl;

    return 0;

}

void valueChanged1(int someNum)
{

    someNum = 100;

    cout<<"someNum in function valueChanged1 is : "<<someNum<<endl;

}

void valueChanged2(int& someNum)
{
    someNum = 100;

    cout<<"someNum in the function valeuChanged2 is: "<<someNum<<endl;
}
```

Output for pass by reference:

```
Before calling function valueChanged 1 called, mynum is : 12
someNum in function valueChanged1 is : 100
After calling function valueChanged1, mynum is : 12
someNum in the function valeuChanged2 is: 100
After calling function valueChanged2 , mynum is : 100
```

Here the interesting thing to note here is that after passing mynum as a argument to the function, it’s value changes to the value of someNum. This means that mynum actually points to the location of somenum. Hence it can be said that either mynum or somenum are acting like alias in this case.

In the following function, we have 2 parameters. First is the input number. Second parameter is passed by reference. And second parameter is equal to 3 times the first parameter. The result is printed in main().

```
#include<iostream>

using namespace std;

void threeTimesN(int num1, int&num2);

int main()
{
    int x = 0;
    int y = 0;

    cout<<"Enter any integer: "<<endl;
    cin>>x;
    cout<<endl;
```

```
    cout<<"initial value of second variable is: "<<y<<endl;

    cout<<endl;

    // Calling function
    threeTimesN(x,y);

    cout<<"The value of y after calling function threeTimesN should be equal to 3 times tha value of x. Which is equal to: "<<y<<endl;


    return 0;
}
void threeTimesN(int num1, int& num2)
{
    num2 = num1 * 3 ;

}
```

Variable scopes and lifetimes

Scope:

In which part of the program is the variable acceptable? where it is defined and where can we use it?

There are two types of variables defined in the C++ code. One is local variable which is defined inside a function (which can be main or some other function) and other are the global variable which are defined in the program at the top before any other function is defined. See the following program for more clarification.

```
#include<iostream>

using namespace std;

void someFunction(int param);

double myGlobalDouble = 50 ; // This is a global variable can be acced by any other function in the program.

int main()
{
    int localTomain = 20;

    cout << "local to main variabe is: "<<localTomain<<endl; //Accessing the loacal variable inside the main
    cout<<"Value of global double in the main is: "<<myGlobalDouble<<endl; //Accessing the global variable

    someFunction(25);
    return 0;

}

void someFunction(int param)
{
    int myLocalvar = 30;

    cout<<"Glocal variable in the someFunction is: "<<myGlobalDouble<<endl; //Accessing the global variable
    cout<<"The parameter is: "<<param<<endl;
    cout<<"the local number is: "<<myLocalvar<<endl; //Accessing the local variable inside the someFunction.

}
```

output:

```
local to main variabe is: 20
Value of global double in the main is: 50
Glocal variable in the someFunction is: 50
The parameter is: 25
the local number is: 30
```

Lifetime:

For how long does the variable live?

This includes the

1. Global variable
2. Local variable
3. Static local variable

Check out the following program, output and the conclusion:

```
#include<iostream>

using namespace std;

void someFunction(int param);

double myGlobalDouble = 50 ; // This is a global variable can be acced by any other function in the program.

int main()
{
    int localTomain = 20;

    cout << "local to main variabe is: "<<localTomain<<endl; //Accessing the loacal variable inside the main
    cout<<"Value of global double in the main is: "<<myGlobalDouble<<endl; //Accessing the global variable

    someFunction(25);
    someFunction(65);
    someFunction(90);
    return 0;

}

void someFunction(int param)
{
    int myLocalvar = 30;

    myLocalvar++;
    myGlobalDouble++;

    cout<<"Global variable in the someFunction is: "<<myGlobalDouble<<endl; //Accessing the global variable
    cout<<"The parameter is: "<<param<<endl;
    cout<<"the local number is: "<<myLocalvar<<endl; //Accessing the local variable inside the someFunction.

    cout<<endl;
    cout<<endl;

}
```

Output:

```
local to main variabe is: 20
Value of global double in the main is: 50
Glocal variable in the someFunction is: 51
The parameter is: 25
the local number is: 31


Glocal variable in the someFunction is: 52
The parameter is: 65
the local number is: 31


Glocal variable in the someFunction is: 53
The parameter is: 90
the local number is: 31
```

Conclusion:

we have called the someFunction three times in the above program. Each time we call it, we have increased the value of the global variable by 1 before printing it. And we have also increased the value of the local variable in the function.

But notice the output, every time we print he global variable using someFunction, its value is increased by 1. i.e. 51, 52, 53 etc. But the value of the local number is always 31 in the someFunction.

This is because every time someFunction is called, the value of local variable is reset to original. This is not the case with the global variable. It's value is kept as long as the program is running. It will not be reset just because the function it is used in is recalled.

This means that the lifetime of the global variable is equal to the lifetime of the program. And the scope of the global variable is unbounded in the program.

If we want a variable whose scope is limited to a specific function and whose lifetime is equal to the runtime of the program then we define a **static local variable**.

```
#include<iostream>

using namespace std;

void someFunction(int param);

double myGlobalDouble = 50 ; // This is a global variable can be acced by any other function in the program.

int main()
{
    int localTomain = 20;

    cout << "local to main variabe is: "<<localTomain<<endl; //Accessing the loacal variable inside the main
    cout<<"Value of global double in the main is: "<<myGlobalDouble<<endl; //Accessing the global variable

    someFunction(25);
    someFunction(65);
    someFunction(90);
    return 0;

}

void someFunction(int param)
{
    int myLocalvar = 30;
    static int someFunctionVar = 500;

    myLocalvar++;
    myGlobalDouble++;
    someFunctionVar++; //Defining the static local variable.

    cout<<"Global variable in the someFunction is: "<<myGlobalDouble<<endl; //Accessing the global variable
    cout<<"The parameter is: "<<param<<endl;
    cout<<"the local number is: "<<myLocalvar<<endl; //Accessing the local variable inside the someFunction.
    cout<<"The value of the someFunctionVar is: "<<someFunctionVar<<endl;

    cout<<endl;
    cout<<endl;

}
```

Output:

```
local to main variabe is: 20
Value of global double in the main is: 50
Global variable in the someFunction is: 51
The parameter is: 25
the local number is: 31
The value of the someFunctionVar is: 501


Global variable in the someFunction is: 52
The parameter is: 65
the local number is: 31
The value of the someFunctionVar is: 502


Global variable in the someFunction is: 53
The parameter is: 90
the local number is: 31
The value of the someFunctionVar is: 503
```

Here the static local variable someFunctionVar has scope limited to the someFunction (like the local variable) but the value of the variable is not reset every time the function is called (like a global variable).