C++ Basics

DSA - C++ Basics (By Uday verma)

C++ Basics

C++ is a powerful programming language that allows developers to create complex and efficient computer programs. If you are new to programming, learning C++ is a great way to start. In this article, we will go over the basics of C++ programming.

Variables

Variables are used to store data in a C++ program. In order to use a variable, you must first declare it. For example, to declare an integer variable named 'num', you would write:

```
int num;
```

You can then assign a value to the variable using the assignment operator, like so:

```
num = 42;
```

You can also declare and assign a value to a variable at the same time, like this:

```
int num = 42;
```

C++ supports a variety of data types, including integers, floating point numbers, characters, and boolean values. Here are some examples of how to declare and initialize variables of different data types in C++:

```
int num = 42;
float pi = 3.14159;
char letter = 'a';
bool isTrue = true;
```

In the above code, an integer variable `num` is initialized to the value `42`, a floating point variable `pi` is initialized to the value `3.14159`, a character variable `letter` is initialized to the value `'a'`, and a boolean variable `isTrue` is initialized to the value `true`.

There are also other data types in C++ such as arrays, pointers, and structures, which allow you to store and manipulate more complex data.

Functions

Functions are used to perform specific tasks in a C++ program. They can be written by the programmer or used from pre-existing libraries. Here is an example of a simple function that takes two integer parameters and returns their sum:

```
int add(int a, int b) {
    return a + b;
}
```

You can then call this function from your program like this:

```
int result = add(5, 7);
```

Control structures

Control structures are used to control the flow of a program. The most commonly used control structures are 'if' statements and loops. Here is an example of an 'if' statement:

```
if (num > 0) {
    cout << "The number is positive." << endl;
} else {
    cout << "The number is negative." << endl;
}
```

This statement will print "The number is positive." if the value of 'num' is greater than 0, and "The number is negative." otherwise.

Here is an example of a 'for' loop:

```
for (int i = 0; i < 10; i++) {
    cout << i << endl;
}
```

This loop will print the numbers 0 through 9.

In addition to the basics of C++ programming covered above, it's also important to learn about the Standard Template Library (STL) and other important concepts for problem solving.

The STL is a collection of data structures and algorithms that are commonly used in C++ programming. It includes containers like vectors, lists, and maps, as well as algorithms like sorting and searching. Learning how to use the STL effectively can greatly simplify your programming and make your code more efficient.

Here's an example of using the STL vector container:

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

using namespace std;

int main() {
    vector<int> nums; // create an empty vector of integers

    // add some numbers to the vector
    nums.push_back(3);
    nums.push_back(1);
    nums.push_back(4);
    nums.push_back(1);
    nums.push_back(5);

    // print out the contents of the vector
    for (int i = 0; i < nums.size(); i++) {
        cout << nums[i] << " ";
    }
    cout << endl;

    // sort the vector
    sort(nums.begin(), nums.end());

    // print out the sorted vector
    for (int i = 0; i < nums.size(); i++) {
        cout << nums[i] << " ";
    }
```

```
    cout << endl;

    return 0;
}
```

Other important concepts for problem solving in C++ include object-oriented programming, pointers, dynamic memory allocation, and recursion. Object-oriented programming allows you to create reusable code by defining classes and objects. Pointers and dynamic memory allocation are used to manage memory in your program, while recursion is a technique for solving problems by breaking them down into smaller sub-problems.

By mastering these concepts and continuing to practice your programming skills, you can become a proficient C++ programmer and tackle even the most complex programming challenges with confidence. Good luck!

Conclusion

These are just the basics of C++ programming, but they should give you a good foundation to build on. C++ is a complex language with many advanced features, so don't be discouraged if it takes you some time to master it. Happy coding!