between C and C#

time.h localtime() function in C with Examples

asctime() and asctime_s() functions in C with Examples

return statement in C/C++ with Examples

size of char datatype and char array in C

Arrow operator -> in C/C++ with Examples

Logical Not ! operator in C with Examples

Template Specialization in C++

Template in C++is a feature. We write code once and use it for any data type including user defined data types. For example, sort() can be written and used to sort any data type items. A class stack can be created that can be used as a stack of any data type.
What if we want a different code for a particular data type? Consider a big project that needs a function sort() for arrays of many different data types. Let Quick Sort be used for all datatypes except char. In case of char, total possible values are 256 and counting sort may be a better option. Is it possible to use different code only when sort() is called for char data type?
It is possible in C++ to get a special behavior for a particular data type. This is called template specialization.

```
// A generic sort function
template <class T>
void sort(T arr[], int size)
{
    // code to implement Quick Sort
}

// Template Specialization: A function
// specialized for char data type
template <>
void sort<char>(char arr[], int size)
{
    // code to implement counting sort
}
```

Another example could be a class *Set* that represents a set of elements and supports operations like union, intersection, etc. When the type of elements is char, we may want to use a simple boolean array of size 256 to make a set. For other data types, we have to use some other complex technique.

An Example Program for function template specialization

For example, consider the following simple code where we have general template fun() for all data types except int. For int, there is a specialized version of fun().

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

template <class T>
void fun(T a)
{
   cout << "The main template fun(): "
        << a << endl;
}

template<>
void fun(int a)
{
    cout << "Specialized Template for int type: "
         << a << endl;
}

int main()
{
    fun<char>('a');
    fun<int>(10);
    fun<float>(10.14);
}
```

Output:

```
The main template fun(): a
Specialized Template for int type: 10
The main template fun(): 10.14
```

An Example Program for class template specialization

In the following program, a specialized version of class Test is written for int data type.

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

template <class T>
class Test
{
  // Data memnbers of test
public:
   Test()
   {
       // Initialization of data members
       cout << "General template object \n";
   }
   // Other methods of Test
};

template <>
class Test <int>
{
public:
   Test()
   {
       // Initialization of data members
       cout << "Specialized template object\n";
   }
};

int main()
{
    Test<int> a;
    Test<char> b;
    Test<float> c;
    return 0;
}
```

Output:

```
Specialized template object
General template object
General template object
```

How does template specialization work?

When we write any template based function or class, compiler creates a copy of that function/class whenever compiler sees that being used for a new data type or new set of data types(in case of multiple template arguments).

If a specialized version is present, compiler first checks with the specialized version and then the main template. Compiler first checks with the most specialized version by matching the passed parameter with the data type(s) specified in a specialized version.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Recommended Posts:

- is_fundamental Template in C++
- is_reference Template in C++
- is_empty template in C++
- std::is_integral template in C++
- std::is_enum Template in C++
- is_class template in C++
- is_standard_layout template in C++
- is_arithmetic Template in C++
- std is_object Template in C++
- is_pointer Template in C++
- std is_floating_point Template in C++
- std is_union() template in C++
- Template Metaprogramming in C++
- is_signed template in C++
- is_polymorphic template in C++

Article Tags : C C++ cpp-template

Practice Tags : C CPP

24

2.3

☐ To-do ☐ Done

Based on **45** vote(s)

Feedback/ Suggest Improvement Add Notes Improve Article