Generics

Introduction

A data structure is a format for organizing, managing, and storing data for efficient access and modification that is capable of working with any data type that fulfills its constraints. Thus, it is ideal to construct them as *generic container classes* since it allows a single class definition that makes the collection private, prohibiting direct access, and uses only special member functions and other methods to initialize, destroy, copy, analyze, and modify the collection.

Generics

A generic function or class uses placeholder types that actual data types replace when the function is invoked, or the class is instantiated. To declare a generic function or class, use the '`template`' keyword, followed by a template parameter list as follows:

`template <`*`template-parameter-list`*`>`

where each template parameter is in the form

`typename` *`identifier`* or `class` *`identifier`*

The keywords '`typename`' and '`class`' are interchangeable when defining templates.

Within the function or class body, the template identifier replaces actual data types. Below are examples of generic functions and generic classes:

Example:

```
template <typename T>
void Swap(T& a, T& b)
{
  T t = a;
  a = b;
  b = t;
}
```

```
template <class T>
class Item
{
  public:
  T value;
  int count;
};
```

```
template <class K, class V>
class Entity
{
  public:
  K key;
  V values[100];
  int size;
};
```

```
template <typename T>
T Max(T data[],int n)
{
  T m = data[0];
  int i = 0;

  while(i < n)
  {
    if(m < data[i])
    {
      m = data[i];
    }
    i += 1;
  }
  return m;
}
```

Key Observations:

- Not all members of a generic class need to be generic (see *Item* class).
- Multiple template parameters can be used (see *Entity* class).

Template Invocation & Instantiation

To invoke a generic function or instantiate a generic class object, use the following syntaxes:

- **Function Invocation Syntax**:
  `function-name``[<`*`data-type`*`>](`*`argument-list`*`)`
- **Class Instantiation Syntax**:
  `class-name``<`*`data-type`*`>` *`identifier`*`[(`*`argument-list`*`)];`

Note: Specifying the data type is typically optional for functions, as the compiler can deduce it from its arguments. However, explicitly proving the type is mandatory for class instantiation.

Example: (continuation)

```
int main()
{
  std::string words[] = {"apple", "orange", "banana", "peach", "grape"};
  Item<std::string> counter;
  Entity<std::string,int> group;
  Swap(words[1],words[3]); //after word[1] = peach, word[3] = orange
  cout << Max(words,5); // orange
  return 0;
}
```

When invoking a generic function or instantiating a generic class, not all data types can be substituted for template parameters. A valid substitution must support all operations performed on the generic type within the function or class.

Example:

```
template <typename T>
T diff(const T& lhs, const T& rhs)
{;
  return lhs - rhs;
}

template <typename T>
T same(const T& lhs, const T& rhs)
{;
  return lhs + rhs;
}

int main()
{
  std::string a = "first", b = "second";
  std::string c = diff(a,b); //error:  - not defined for std::string
  std::string d = same(a,b); //valid
  return 0;
}
```