C++ Programming
Lesson 7

TINPRO02-5B

HOGESCHOOL ROTTERDAM

PROGRAMMA

Recap

Templates(generics)

Recap Lesson 6

- Stream I/O.
- Operator overloading
- Reference return
- lvalue, rvalue

Templates: generic programming

- “Generics” is the idea to allow types, especially user-defined types to be a parameter to methods, classes and interfaces
- The method of Generic Programming is implemented to increase the efficiency of the code.
- Generic Programming enables the programmer to write a general algorithm which will work with all data types.
- It eliminates the need to create different algorithms if the data type is an integer, string or a character.

Generic programming advantages

Advantages of generic programming are:

1. Code Reusability
2. Avoid Function Overloading
3. Once written it can be used for multiple times and cases.

Templates

- Generics can be implemented in C++ using Templates.
- Template is a simple and yet very powerful tool in C++.
- The simple idea is to pass data type as a parameter so that we don’t need to write the same code for different data types.
- For example, you may need a function sort() for different data types. Rather than writing and maintaining multiple overloaded functions, we can write one sort() and pass data type as a parameter.

Example: template part 1

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

// One function works for all data types.
// This would work even for user defined types
// if operator '>' is overloaded

template <typename T>

T myMax(T a, T b)
{
    T result;
    result = (a>b)? a : b;
    return (result);
}
```

Example: template part 2

```
int main()
{
    // Call myMax for int
    cout << myMax<int>(3, 7) << endl;

    // call myMax for double
    cout << myMax<double>(3.0, 7.0) << endl;

    // call myMax for char
    cout << myMax<char>('g', 'e') << endl;

    return 0;
}
```

Example: template alternative

```
int main()
{
    int i=3, j = 7;
    // Call myMax for int
    cout << myMax(i, j) << endl;

    double a = 3.0, b = 7.0;
    // call myMax for double
    cout << myMax(a, b) << endl;
    char g = 'g', e = 'e';
    // call myMax for char
    cout << myMax(g, e) << endl;

    return 0;
}
```

Template

Because our template function includes only one template parameter (class T) and the function template itself accepts two parameters, both of this T type, we cannot call our function template with two objects of different types as arguments:

```
int i;
long l;
k = myMax (i,l);
```

This would not be correct, since `myMax` function template expects two arguments of the same type, and in this call to it we use objects of two different types.

Template with different argument types

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

// One function works for all data types.
// This would work even for user defined types
// if operator '>' is overloaded

template <typename T, typename U>

T myMax(T a, U b)
{
    T result;
    result = (a>b)? a : b;
    return (result);
}
```

Template declaration

The format for declaring function templates with type parameters is:

```
template <class identifier> function_declaration;
template <typename identifier> function_declaration;
```

The only difference between both prototypes is the use of either the keyword class or the keyword typename. Its use is indistinct, since both expressions have exactly the same meaning and behave exactly the same way.

Generic class using template

- Like function templates, class templates are useful when a class defines something that is independent of data type.
- Can be useful for classes like LinkedList, binary tree, Stack, Queue, Array, etc.

Template class part 1 (.h)

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

template <typename T>
class Array {
private:
    T* ptr;
    int size;

public:
    Array(T arr[], int s);
    void print();
};
```

Template class part 2 (.h)

```
template <typename T>  Array<T>::Array(T arr[], int s) {
    ptr = new T[s];
    size = s;
    for (int i = 0; i < size; i++)
        ptr[i] = arr[i];
}

template <typename T>  void Array<T>::print() {
    for (int i = 0; i < size; i++)
        cout << " " << *(ptr + i);
    cout << endl;
}
```

Template class usage

```
#include "myArray.h"

int main()
{
    int arr[5] = { 1, 2, 3, 4, 5 };
    Array<int> a(arr, 5);
    a.print();
    return 0;
}
```

Templates: .h and .cpp exception!

- Templates are special: compiled on demand, until an instantiation with specific template arguments is required.
- Exception in multi-file projects: the implementation (definition) of a template class or function must be in the same file as its declaration.
- We cannot separate the interface in a .h file: both interface and implementation must be included in any file that uses the templates
- Compilers are prepared to allow the inclusion more than once of the same template file with both declarations and definitions in a project without generating linkage errors.

overtref jezelf