Creative Software Programming Practice (week-3-2)

Every assignment will be announced on **Thursday** and should be sumitted by next **Tuesday**.

We have practice classes on Wednesdays and Thursdays.
The contents of the practice class are different from the assignments and aim to be completed on the same day.

Assignments will be publish on Thursday and must be submitted by the next Tuesday.

In this week **Handed out will be Sep 17, 2020, Due Sep 22, 2020**

Topics

1. Template
2. STL
3. Pratice

1. Template

Function templates are special functions that can operate with generic types. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.

In C++ this can be achieved using template parameters. A template parameter is a special kind of parameter that can be used to pass a type as argument: just like regular function parameters can be used to pass values to a function, template parameters allow to pass also types to a function. These function templates can use these parameters as if they were any other regular type.

The format for declaring function templates with type parameters is:

```
template <typename T>
void function (T a) {
}
template <class M>
M function (int a) {
    return (type of M);
}
template <typename A, typename B>
void function (A a, B b) {
}
```

When the compiler encounters this call to a template function, it uses the template to automatically generate a function replacing each appearance of Type variable by the type passed as the actual template parameter (int in this case) and then calls it. This process is automatically performed by the compiler and is invisible to the programmer.

```
// file: template.cc
#include <iostream>

template <typename T>
```

```
void function(T a) {
    std::cout << a << std::endl;
}

int main() {

    function<double>(97.3);

    // 97.3 is casted as 97, because function<int> accept a as int
    function<int>(97.3);

    // 97.3 convert to 97 and 97 is 'a' in ASCII code
    function<char>(97.3);

    return 0;
}
```

2. STL

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized. A working knowledge of template classes is a prerequisite for working with STL.

STL has four components

- Algorithms
- Containers
- Functions
- Iterators

You will probably use Containers and Iterators in class.

- String
- Vector
- List
- Stack
- Queue

  ...

You need to reference the header before using stl.

```
#include <string> // std::string
#include <vecotr> // std::vector
...
```

A container is a holder object that stores a collection of other objects (its elements). They are implemented as class templates, which allows great flexibility in the types supported as elements.

The container manages the storage space for its elements and provides member functions to access them, either directly or through iterators (reference objects with similar properties to pointers).

Containers replicate structures very commonly used in programming: dynamic arrays (vector), queues (queue), stacks (stack), heaps (priority_queue), linked lists (list), trees (set), associative arrays (map)...

Various Containers are included in STL, and you can create your own container if you wish.

```
// file: stl.cc
#include <iostream>
#include <string>
#include <list>
#include <vector>

int main() {
    // string is a container which contains characters
    std::string str = "123";

    // list is a container which each node is connected to other
    // The STL container receives as a template parameter what to contain.
    // In this time, we create list contains int type.
    std::list<int> list;

    list.push_back(3);
    list.push_back(2);
    list.push_back(1);
    // now list has 3,2,1

    // you can iter over container using for-each loop
    for (int e : list) {
        std::cout << e << ", ";
    }
    std::cout << std::endl;

    // vector is a container which like a dynamic array
    std::vector<int> vec;
    vec.push_back(1);
    vec.push_back(2);
    vec.push_back(3);

    // Since vectors are like arrays, they can be accessed at random.
    for (int i = 0; i < 3; i++) {
        std::cout << vec[i] << ", ";
    }

    std::cout << std::endl;
}
```

3. Pratice

Learn more about stl on the cpp reference, especially the differences and features of *lists*, *vectors*, *stacks*, and *queues* that will be used frequently in the future.

```
```