Template - Motivation

- What is the difference between an integer stack and a string stack, as far as their specification and implementation are concerned?
- What is the difference between finding the largest items in a list of integers, a list of doubles, or a list of strings? As far as the nature of the algorithm is concerned?
- We would like to reuse the code by writing generic functions and classes.
- How do we define a generic functions and classes in C++?

Template – a C++ feature

- A template is a general pattern for a class or a function in C++
- Everything is filled in, except for one or more types
- Examples:
  - □ A stack template class, with all the definitions complete, methods implemented, etc, but the type of the data item left open as a parameter
  - □ A sort template function: type of the item being sorted is left open

Function template

```
#include <iostream>
#include <string>
using namespace std;
int larger(int x, int y)
{
        if (x > y) return  x;    else return y;
}
int main()
{
        string a("good");
        string b("morning");
        cout << larger(6, 5) << endl;
        cout << larger(6.5, 5.5) <<endl;       //this won’t compile
        cout << larger(a, b) <<endl; //this won’t compile
        return 0;
}
```

Function template

- How to modify the above program so that it will compile?
- Two solutions
  - Create two more versions of larger() functions
  - Use function template

Function template

```
#include <iostream>
#include <string>
using namespace std;
template <class T>
T larger(T x, T y)
{       if (x > y) return  x;     else return y; }
int main()
{
        string a("good");
        string b("morning");
        cout << larger(6, 5) << endl;
        cout << larger(6.5, 5.5) <<endl;
        cout << larger(a, b) <<endl;
        return 0;
}
```

Class Template Example Using Stack (LIFO – last in first out)

A stack of books

A pile of plates

A stack of dvd/cd

Class template

```
//IntStack.h
class IntStack {
  enum { ssize = 100 };
  int stack[ssize];
  int top;
public:
  IntStack() { top = 0;}   <- top is the index beyond the toppest element
  void push(int i);
  int pop();
};
```

Class template

```
//IntStack.cc
#include "IntStack.h"

void IntStack::push(int i) {
    stack[top++] = i;  //post-increment;  same as stack[top] = i; top = top + 1;
}

int IntStack::pop() {
    return stack[--top];    // top = top -1; return stack[top];
}
```

Class template

```
//TestIntStack.cc
#include <iostream>
#include "IntStack.h"
using namespace std;
int main() {
  IntStack is;
  for(int i = 0; i < 20; i++)
    is.push(i);

  for(int k = 0; k < 20; k++)
    cout << is.pop() << endl;

  return 0;
}
```

Class template

- What if I need a stack of double (or strings)?

Class template

```
template <class T>
class TStack {
  enum { ssize = 100 };
  T stack[ssize];
  int top;
public:
  TStack() : top(0) {}  //initialization; similar as assignment but preferred/mandated in some cases such
       as const, see effective C++.
  void push(T i);
  T pop();
};
template <class T>
void TStack<T>::push(T i) {     stack[top++] = i; }

template <class T>
T TStack<T>::pop() {   return stack[--top]; }
```

Class template

```
int main() {
  TStack<int> is;
  TStack<double> ds;

  cout << sizeof(is) << endl << endl;
  for(int i = 0; i < 20; i++)
      is.push(i);
  for(int k = 0; k < 20; k++)
    cout << is.pop() << endl;
  for(int i = 0; i < 10; i++)
     ds.push(i+2.5);
  for(int k = 0; k < 10; k++)
    cout << ds.pop() << endl;
  return 0;
}
```

Strategy to develop function and class templates

- Develop specific class/function first
- Test them
- Convert the specific class/function to templates
- Test them again

vector in the standard template library (STL)

- Standard Template Library
  - Extension to C++
  - Object-oriented
  - Generic entities: container, iterator, algorithm
    - Container: data structure that hold objects, vector, list, stack, queue …
    - Iterator: A generalization of a pointer, used to reference an element in a container
    - Algorithm: generic functions.

Vector container in STL

- Simplest container in STL
- Example functions:
  - □ void push_back(const T& el) – insert el at the end of the vector.
  - □ void clear()
  - □ iterator begin() //return an iterator that references the 1st element of the vector
  - □ at(); pop_back(); insert() …

Vector container in STL

- Initialization:
- …
- #include <vector>

- int main()
- {
- *std::vector<int> v1; //empty vector*
- *for (int i=0; i<5; i++)*
- *v1.push_back(i); //v1 = (0 1 2 3 4)*
- *….*
- *}*

More examples:
https://cplusplus.com/reference/vector/vector/push_back/