ECE 220: Computer Systems & Programming

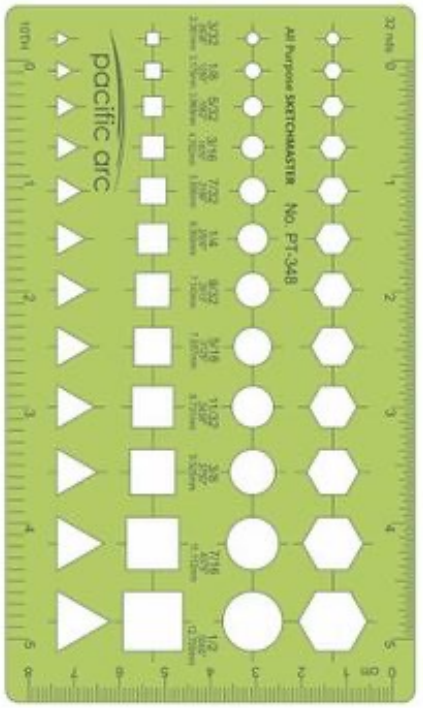
Lecture 22: Intro to C++: Iterators and Templates

Thomas Moon

April 11 , 2024

Templates

- A template is a blueprint for creating a generic function or a class.



- Template ruler:
  Shapes are pre-defined, but colors are not.

- Template in cpp:
  Functionalities are pre-defined, but types are not.

1. Function templates 2. Class templates

Function Templates

```
int Add(int a, int b){
    return a+b;
}
double Add(double a, double b){
    return a+b;
}
```

```
int main(){
    cout<<Add(1, 3)<<endl;
    cout<<Add(1.2, 2.5)<<endl;
}
```

```
template <class T>
```

Or

```
template <typename T>
T Add(T a, T b){
    return a+b;
}
```

template parameter

Function Templates

- Multiple template parameters are possible.

```
template <typename T1, typename T2>
void Print(T1 a, T2 b){
    cout<<a<<endl;
    cout<<b<<endl;
}

int main(){
    Print(2,'b');
}
```

Function Templates: Example

```
template <typename T>
T Mul(T a, T b){
    return a*b;
}
```

← T is Point class

← (Point object) * (Point object)

Is it defined in Point class?

You need something like this…

```
Point operator*(const Point& p){
    Point temp(x*p.x, y*p.y);
    return temp;
}
```

```
class Point{
    private:
        int x,y;
    public:
        Point(int _x=0, int _y=0){x=_x; y=_y;}
        void ShowPosition(){cout<<x<<", "<<y<<endl;}
};

int main(){
    Point p1(1,2);
    Point p2(3,1);
    Point p3;
    p3 = Mul(p1,p2);
```

Class Templates

- Just like function templates, class templates allow classes to have members that use template parameters as types.

```
class Data{
    int data;
public:
    Data(int d){
        data = d;
    }
    void ShowData(){
        cout<<data<<endl;
    }
};
int main(){
    Data d1(10);
    d1.ShowData();
```

```
template <typename T>
class Data{
    T data;
public:
    Data(T d){
        data = d;
    }
    void ShowData(){
        cout<<data<<endl;
    }
};
int main(){
    Data<int> d1(10);
    d1.ShowData();

    Data<char> d2('a');
    d2.ShowData();
```

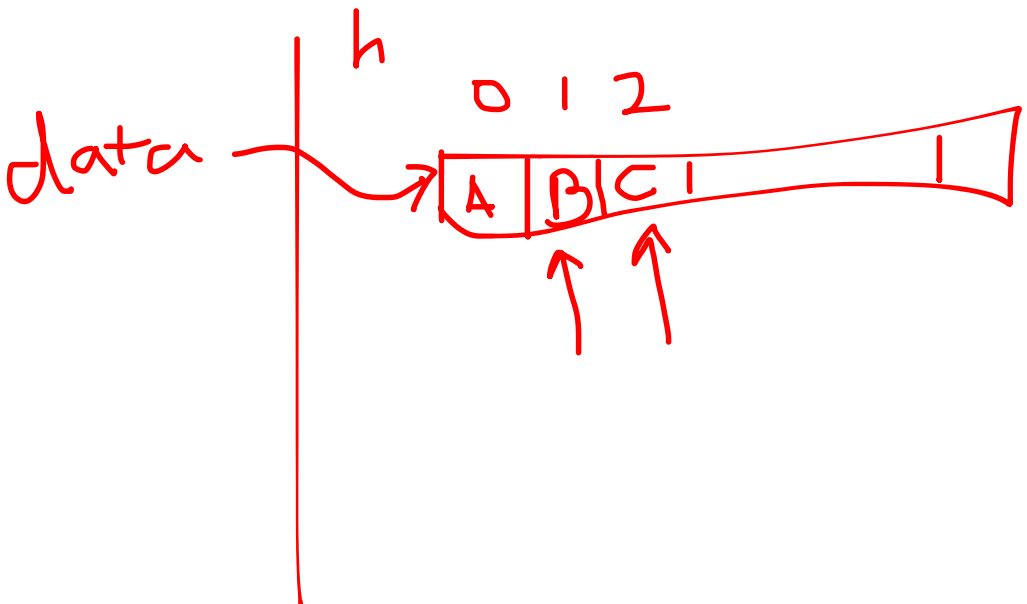
Require an explicit template parameter for class templates.

Example: Implement Stack using Class Template

```
class MyStack{
    private:
      int TOS; // index for top of the stack
      int size; // size of the stack
      char* data; // pointer for dynamic array
    public:
        MyStack(int _size = 5){
            TOS = -1; // initialize TOS
            size = _size;
            data = new char[size];
        }
        ~ MyStack(){ delete []data;}
        void push(const char &value){
            TOS++;
            data[TOS] = value;
        }
        char top(){ return data[TOS];}
        void pop(){ TOS--;}
        bool empty(){
            if(TOS == -1) return true;
            else return false;
        }
};
```

```
int main(){
    MyStack s(10);
    s.push('A');
    s.push('B');
    s.push('C');

    while(!s.empty()){
        cout<<s.top()<<endl;
        s.pop();
    }
```



```
template <class T>
class MyStack{
    private:
        int TOS;
        int size; // size of the stack
        T* data;
    public:
        MyStack(int _size = 5){
            TOS = -1;
            size = _size;
            data = new T[size];
        }
        ~ MyStack(){ delete []data;}
        void push(const T &value){
            TOS++;
            data[TOS] = value;
        }
        T top(){ return data[TOS];}
        void pop(){ TOS--;}
        bool empty(){
            if(TOS==-1) return true;
            else return false;
```

```
int main(){
    MyStack<char> s1(10);
    s1.push('A');
    s1.push('B');
    s1.push('C');
    while(!s1.empty()){
        cout<<s1.top()<<endl;
        s1.pop();
    }

    MyStack<int> s2(10);
    s2.push(1);
    s2.push(2);
    s2.push(3);
    while(!s2.empty()){
        cout<<s2.top()<<endl;
        s2.pop();
    }
```

Standard Template Library (STL)

- STL is a set of C++ template classes to provide common data structures and functions.
  - Algorithms
  - Functions
  - **Containers – stores objects and data**
    - **vector** → D. A
    - **list** → L. L
    - **stack**
    - **queue**
  - **Iterators – object that points to an element inside the container**

Container example - stack

```
#include <iostream>
#include <stack>

int main(){
    stack<char> s;
    s.push('a');
    s.push('b');
    s.push('c');

    while(!s.empty()){
        cout<<s.top()<<endl;
        s.pop();
    }
}
```

Member functions

(constructor)	Construct stack (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
top	Access next element (public member function)
push	Insert element (public member function)
emplace C++11	Construct and insert element (public member function)
pop	Remove top element (public member function)
swap C++11	Swap contents (public member function)

Container example : Vector (dynamic array)

```
#include <vector>
using namespace std; //for vector and cout

    vector<int> a;          ← default constructor (size = 0)
    vector<int> b(4, 3);    ← parameterized constructor (size, initial values)

    cout<<"size of a: "<< a.size()<<endl;
    for(int i=0; i < a.size(); i++)
        cout<<a[i]<<" ";
    cout<<endl;

    cout<<"size of b: "<< b.size()<<endl;
    for(int i=0; i < b.size(); i++)
        cout<<b[i]<<" ";
    cout<<endl;

    for(int i=0; i < 2; i++)
        a.push_back(i);

    cout<<"size of a: "<< a.size()<<endl;
    for(int i=0; i < a.size(); i++)
        cout<<a[i]<<" ";
    cout<<endl;
```

operator[]

vector::size	Return size (public member function)
vector::push_back	Add element at the end (public member function)

Result:

```
size of a: 0

size of b: 4
3 3 3 3
size of a: 2
0 1
```

Iterator : Why do you need this?

```
vector<int> data;
data.push_back(1);
data.push_back(2);
data.push_back(3);

for(int i=0;i<data.size();i++)
    cout<<data[i]<<endl;
```

```
vector<int> data;
data.push_back(1);
data.push_back(2);
data.push_back(3);

vector<int>::iterator it;
for(it = data.begin(); it != data.end(); it++)
    cout<< *it << endl;
```

```
list<int> data;
data.push_back(1);
data.push_back(2);
data.push_back(3);

for(int i=0;i<data.size();i++)
    cout<<data[i]<<endl;
```

compile error (operator[] not defined in list)

```
list<int> data;
data.push_back(1);
data.push_back(2);
data.push_back(3);

list<int>::iterator it;
for(it = data.begin(); it != data.end(); it++)
    cout<< *it << endl;
```

Iterator

- Iterator can point to an element inside the container. It can iterate through the elements using the increment(++), begin(), and end().

```
list<int> data;
list<int>::iterator it;
```

← declare container
← declare iterator

`data.begin()` ←to the first element

`data.end()` ←to the **past-the-end** element

```
for(it = data.begin(); it != data.end(); it++)
    cout<< *it << endl;
```