```
template <class T>
class DynArray {
private:
    T * arr;
    int len;
public:
    T & operator [] (int index) {
      if (index >= len) {
        T * newarr = new T[index+1];
        ...
      }
      ..
    }
    ...
};
```

can have multiple parameters

```
Use: DynArray<int> da1(10);
     DynArray<float> da2(10);
     DynArray<C> da3(10);
     C obj1, obj2;
     obj1 = da3[3];
```

Can also template global functions:

```
template <class T>
void swap(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}
```

Use:

```
int x = 3, int y = 2;
swap(x, y);
```

Two step process: 1. instantiate the template
2. compile the resultant procedure

```
double x = 3.0; y = 4.0;
swap(x, y);
```

STL

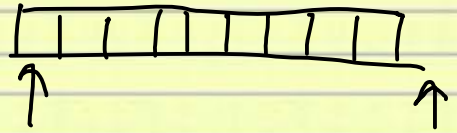
There is heavy use in the STL of iterators, which are generalization of the explicit pointers of C.

Uses of pointers:

```
char name[] = "word";
char ch, *p;
p = name;
ch = p[1];   // char 'o'
ch = *(p + 3)   //char 'd'
*p = 'c';  // changes 'w' to 'c'
while (*p && *p++ != 'r');
```

Iterators are an abstraction of these ideas

A collection is considered as having a start and an end.



begin() end() points just past the last one

STL has a set of standard templates which implement vector, list, deque (sequence templates)

Operation	vector	list	deque
access 1st. elem.	const	const	const
access last elem.	const	const	const
access random elem.	const	const	linear
add/del at start	linear	const	const
add/del at end	const	const	const
add/del at random	linear	const	const

All of these operations use iterators to point to the target of the operation.

vector example

```
#include <vector>

vector<int>  v;
int val, nitems = 0;
while (cin >> val, cin.good()) {
   v.push_back(val);
   cout.width(6);
   cout << ++nitems << ": " <<
         v[nitems-1] << endl;
}
```

Other operations : pop_back
swap
insert
size
capacity

List example

```
#include <list>
list<string> yr;
char *things[] = {"ab", "cd", ...};
list<string>::iterator iter;
for (int i=0; i<10; i++)
    yr.push_back(things[i]);
for (iter = yr.begin(); iter != yr.end(); ++iter)
    cout << *iter << endl;
```

dereferencing

deque example

```
#include <algorithm>
#include <deque>

class Deck {
 private:
    deque<Card> cards;
 public:
    void shuffle() {
        random_shuffle(cards.begin(), cards.end());
    }
    ...
};

Deck deck;
deck.shuffle();
```

random_shuffle: a templated global function

Sequence adapter

- rework the public interface of an existing class
- stack, queue, priority_queue

stack has operations:

stack	container
top()	back()
push()	push_back()
pop()	pop_back()
empty()	empty()
	⋮

Can we use any underlying sequence for the stack, (vector, list, deque)

```
stack<const char *, vector<const char *>>  s;
s.push("abcd");
s.push("efgh");
s.pop();                    s.top(); // value "abcd"
```

Associative containers

- map, set

```
string token, value, line;
map<string, double> m1;
getline(fs, line);
value = line.substr(ip + 1);
m1[token] = atof(value.c_str());
iter = m1.find("abc");
if (iter != m1.end()) {
    ... m1["abc"] ...
}
```