STL Allocator + Mem Pool

STL Allocator Interface

- An Example – allocator as a template parameter

```
template < class T, class Alloc = allocator<T>
> class vector;
template < class T, class Alloc = allocator<T>
> class list;
```

STL Allocator Interface

- What does an allocator have? (xmemory0)

Member types:

```
typedef void _Not_user_specialized;
typedef _Ty value_type;
typedef value_type *pointer;
typedef const value_type *const_pointer;
typedef value_type& reference;
typedef const value_type& const_reference;
typedef size_t size_type;
typedef ptrdiff_t difference_type;
typedef true_type
propagate_on_container_move_assignment;
typedef true_type is_always_equal;
```

Member function:

```
pointer address(reference _Val) const
_NOEXCEPT
const_pointer address(const_reference
_Val) const _NOEXCEPT
void deallocate(pointer _Ptr, size_type
_Count)
_DECLSPEC_ALLOCATOR pointer
allocate(size_type _Count)
template<class _Uty> void destroy(_Uty
*_Ptr)
template<class _Objty,
class... _Types>
void construct(_Objty *_Ptr, _Types&&...
_Args)
```

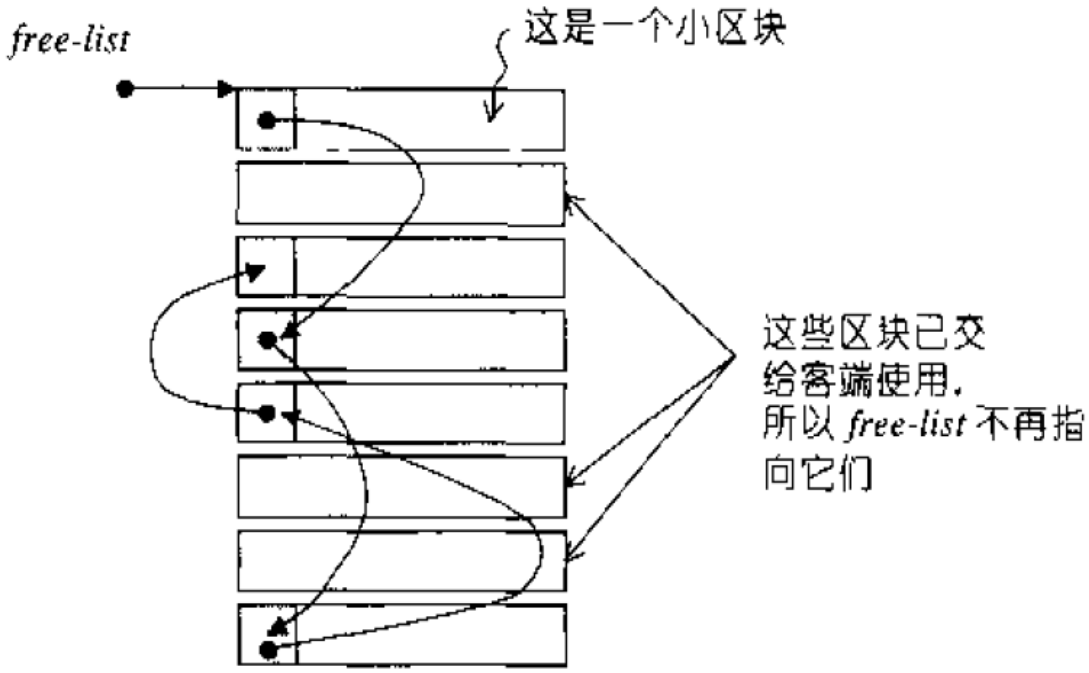
STL Allocator Interface

- An example code

```
namespace JJ
{
template <class T>
inline T* _allocate(ptrdiff_t size, T*) {
T *tmp = (T*)(::operator new((size_t)(size*sizeof(Y))));
            if (tmp == 0) {
            cerr << "out of memory"<< endl;
            exit(1);
            }
            return tmp;
}
template <class T>
inline void _deallocate(T *buffer) {
            ::operator delete(buffer);
}
……
```

Mem pool

- Speed up of a larger number of dynamic memory allocation



An example: Mem pool using block based allocation strategy

Howework

- Task
  - Implement an allocator with mem-pool to support std::vector
  - Test the ability of the allocator with 10000 times of various size of vector construction and destruction (use random value)

Requirement

- Implement an memory allocator for STL vector and list. The interface for the allocator is in the xmemory0 file.
- The allocator should optimize memory allocation speed using memory pool
- The allocator should support arbitrary memory size allocation request.

How to test your allocator

- First, create more than ten thousand vectors with different number of elements
- Second, pick up 1000 random vectors and resize the vectors with random size
- Third, release all vectors

How to specify the vector size, type through a file

```
// to create vectors
15000//number of vectors to be created
10      int // an integer vector with 10 elements
34      Float //a float vector with 34 elements
…….

// to create lists //number of lists to be created
100 //number of lists to be created
10      int // an integer list with 10 elements
34      Float //a float float with 34 elements


10      int // an integer vector with 10 elements
34      Float //a float vector with 34 elements
…….

//random resize
23 30 //random vector index  new vector size
56 78
1345 1000
6700 2000
```

You can write a program to generate this file to test your allocator

About submission

- Can be a team work (1~2 students)
- Besides the code, you should submit a report including :
  - The team members
  - The method description
  - The evaluation
  - C++ standard version
- Zip the codes together with the report.
- Submit it individually.