//顺序表的实现

#include <iostream>

using namespace std;

#define max\_list 60

enum Error\_code {

success, fail, scope\_error, underflow, overflow,

fatal, not\_present, duplicate\_error, entry\_inserted, entry\_found, internal\_error

};

//顺序表类List的定义

template <class List\_entry>

class List {

public:

//ADT中的10个基本操作

List();

int size() const;

bool full() const;

bool empty() const;

void clear();

void traverse(void (\*visit)(List\_entry&));

Error\_code retrieve(int position, List\_entry& x) const;

Error\_code replace(int position, const List\_entry& x);

Error\_code remove(int position, List\_entry& x);

Error\_code insert(int position, const List\_entry& x);

protected:

int count;//记录当前表中有效元素的个数

List\_entry entry[max\_list]; //entry数组，作为顺序表存储元素的核心

};

template <class List\_entry>

List<List\_entry>::List()

{

count = 0;

}

template <class List\_entry>

int List<List\_entry>::size() const

{

return count;

}

template <class List\_entry>

bool List<List\_entry>::full() const

{

return (count == max\_list);

}

template <class List\_entry>

bool List<List\_entry>::empty() const

{

return (count == 0);

}

template <class List\_entry>

void List<List\_entry>::clear()

{

count = 0;

}

template <class List\_entry>

void List<List\_entry>::traverse(void (\*visit)(List\_entry&))

{

for (int i = 0; i < count; i++)

(\*visit)(entry[i]);

}

template <class List\_entry>

Error\_code List<List\_entry>::retrieve(int position, List\_entry& x) const

{

if (empty())

return underflow;

if (position < 0 || position >= count)

return scope\_error;

x = entry[position];

return success;

}

template <class List\_entry>

Error\_code List<List\_entry>::replace(int position, const List\_entry& x)

{

if (empty())

return underflow;

if (position < 0 || position >= count)

return scope\_error;

entry[position] = x;

return success;

}

template <class List\_entry>

Error\_code List<List\_entry>::remove(int position, List\_entry& x)

{

if (empty())

return underflow;

if (position < 0 || position >= count)

return scope\_error;

x = entry[position];

for (int i = position; i < count; i++)

entry[i] = entry[i + 1];

count--;

return success;

}

template <class List\_entry>

Error\_code List<List\_entry>::insert(int position, const List\_entry& x)

{

if (full())

return overflow;

if (position < 0 || position > count)

return scope\_error;

for (int i = count - 1; i >= position; i--)

entry[i + 1] = entry[i];

entry[position] = x;

count++;

return success;

}

template <class List\_entry>

void print(List\_entry x) {

cout << x << endl;

}

int main() {

List<float>grade;

int count = 0;

cout << grade.empty() << endl;

cout << grade.full() << endl;

float j[10] = { 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5 };

for (int i=0; i < 10;i++)

{

grade.insert(i, j[i]);

count=count+1;

}

cout << grade.empty() << endl;

cout << grade.full() << endl;

cout << count << endl;

grade.traverse(print);

float x;

grade.retrieve(4,x);

cout << x << endl;

grade.replace(4, 10);

float y;

grade.retrieve(4, y);

cout << y << endl;

grade.insert(5, 8.8);

count += 1;

cout << count << endl;

grade.traverse(print);

float z;

grade.remove(4,z);

count -= 1;

cout << count << endl;

grade.traverse(print);

grade.clear();

cout << grade.empty();

}