













**A卷第一题（*BY LuHaozhe*）**

#include <iostream>

using namespace std;

class Complex {

int shi;

int xu;

public:

Complex() {

shi = 0;

xu = 0;

}

void input(int a,int b) {

shi = a;

xu = b;

}

friend Complex operator+(Complex& a, Complex& b) {

Complex temp;

temp.shi = a.shi + b.shi;

temp.xu = a.xu + b.xu;

return temp;

}

friend Complex operator-(Complex& a, Complex& b) {

Complex temp;

temp.shi = a.shi - b.shi;

temp.xu = a.xu - b.xu;

return temp;

}

friend Complex operator\*(Complex& a, Complex& b) {

Complex temp;

temp.shi = a.shi \* b.shi - a.xu \* b.xu;

temp.xu = a.shi \* b.xu + b.shi \* a.xu;

return temp;

}

void output() {

if (shi == 0) {

if (xu == 0) {

cout << "0" << endl;

}

else {

cout << xu << "i" << endl;

}

}

else {

if (xu == 0) {

cout << shi << endl;

}

else {

if (xu > 0) {

cout << shi << "+" << xu << "i" << endl;

}

else {

cout << shi << xu <<"i" <<endl;

}

}

}

}

};

int main() {

int a1, a2, b1, b2;

cin >> a1 >> a2 >> b1 >> b2;

Complex com1;

Complex com2;

Complex com3;

Complex com4;

Complex com5;

com1.input(a1, a2);

com2.input(b1, b2);

com3 = com1 + com2;

com3.output();

com4 = com1 - com2;

com4.output();

com5 = com1 \* com2;

com5.output();

return 0;

}

A卷第二题（***BY JiangFengyi***）

#include<iostream>

#include<string>

using namespace std;

class File

{

protected:

string filnname;

int filesize;

public:

File(string name, int size) :filnname(name), filesize(size) {};

virtual ~File() {};

virtual void UpdateFile(int encoder, string newname) = 0;

virtual void UpdataFile(int encoder, int newsize) = 0;

virtual void show() = 0;

};

class ChangeEncode\_name : public File

{

int fileEncoder;

public:

ChangeEncode\_name(string name, int size, int encoder) :File(name, size), fileEncoder(encoder) {};

void UpdateFile(int encoder, string newname)

{

fileEncoder = encoder;

filnname = newname;

}

void UpdataFile(int encoder, int newsize)

{

fileEncoder = encoder;

filesize = newsize;

}

void show()

{

cout << "change\_encodeAndsize:" << filnname << " " << filesize << " ";

switch (fileEncoder)

{

case 0:

cout << "ASCII";

break;

case 1:

cout << "UNICODE";

break;

case 2:

cout << "UTF8";

break;

case 3:

cout << "ANSI";

break;

}

cout << endl;

}

};

class ChangeEncode\_size :public File

{

int fileEncoder;

public:

ChangeEncode\_size(string name, int size, int encoder) :File(name, size), fileEncoder(encoder) {};

void UpdataFile(int encoder, int newsize)

{

fileEncoder = encoder;

filesize = newsize;

}

void UpdateFile(int encoder, string newname)

{

fileEncoder = encoder;

filnname = newname;

}

void show()

{

cout << "change\_encodeAndsize:" << filnname << " " << filesize << " ";

switch (fileEncoder)

{

case 0:

cout << "ASCII";

break;

case 1:

cout << "UNICODE";

break;

case 2:

cout << "UTF8";

break;

case 3:

cout << "ANSI";

break;

}

cout << endl;

}

};

int main()

{

string filename;

int filesize;

int encoder;

cin >> filename >> filesize >> encoder;

char ch;

cin >> ch;

if (ch == 'N')

{

ChangeEncode\_name file(filename, filesize, encoder);

cin >> filename >> encoder;

file.UpdateFile(encoder, filename);

file.show();

}

if (ch == 'S')

{

ChangeEncode\_size file(filename, filesize, encoder);

cin >> filesize >> encoder;

file.UpdataFile(encoder, filesize);

file.show();

}

}

A卷第三题（***BY Luhaozhe***）

#include<iostream>

#include<string>

using namespace std;

template <class T>

class list {

struct node {

T data;

node\* next;

};

node\* head;

node\* tail;

public:

list() : head(nullptr), tail(nullptr) {}

void Insert(T item) {

node\* newNode = new node;

newNode->data = item;

newNode->next = head;

head = newNode;

if (tail == nullptr) {

tail = head;

}

}

void Append(T item) {

node\* newNode = new node;

newNode->data = item;

newNode->next = nullptr;

if (tail == nullptr) {

head = tail = newNode;

} else {

tail->next = newNode;

tail = newNode;

}

}

int count() {

int count = 0;

node\* current = head;

while (current != nullptr) {

count++;

current = current->next;

}

return count;

}

void htot() {

if (head != nullptr && head != tail) {

node\* firstNode = head;

head = head->next;

tail->next = firstNode;

firstNode->next = nullptr;

tail = firstNode;

}

}

void ttoh() {

if (head != nullptr && head != tail) {

node\* lastNode = tail;

node\* current = head;

while (current->next != tail) {

current = current->next;

}

current->next = nullptr;

tail = current;

lastNode->next = head;

head = lastNode;

}

}

void display() {

if (head == nullptr) {

std::cout << "emptylist";

} else {

node\* current = head;

while (current != nullptr) {

std::cout << current->data << " ";

current = current->next;

}

}

std::cout << std::endl;

}

void SortList() {

if (head != nullptr && head != tail) {

node\* current = head;

node\* index = nullptr;

T temp;

while (current != nullptr) {

index = current->next;

while (index != nullptr) {

if (current->data > index->data) {

temp = current->data;

current->data = index->data;

index->data = temp;

}

index = index->next;

}

current = current->next;

}

display();

}

}

};

int main() {

string listType;

cin >> listType;

int length;

cin >> length;

if (listType == "int") {

list<int> link1;

list<int> link2;

for (int i = 0; i < length; i++) {

int item;

cin >> item;

link1.Append(item);

link2.Insert(item);

}

cout << link1.count() << endl;

link1.display();

link1.ttoh();

link1.display();

cout << link2.count() << endl;

link2.display();

link2.htot();

link2.display();

cout << "Sorted List: ";

link1.SortList();

} else if (listType == "char")

{ list<int> link1;

list<int> link2;

for (int i = 0; i < length; i++) {

char item;

cin >> item;

link1.Append(item);

link2.Insert(item);

}

cout << link1.count() << endl;

link1.display();

link1.ttoh();

link1.display();

cout << link2.count() << endl;

link2.display();

link2.htot();

link2.display();

cout << "Sorted List: ";

link1.SortList();

}

return 0;

}