# Complex class without friend functions

#include <iostream>

using namespace std;

class Complex{

public:

Complex();

Complex(double r);

Complex(double r, double i);

Complex(Complex &obj);

Complex add(Complex c);

Complex sub(Complex c);

Complex mult(Complex c);//Multiplication: z1. z2 = (x1x2 - y1y2) + i(x1y2 + x2y1)

Complex div(Complex c);

void print();

double getReal() const;

double getImag() const;

void setReal(double r);

void setImag(double i);

private:

double real, imag;

};

void reset(double,double, Complex&, Complex&);

Complex::Complex(){

real = imag = 0;

}

Complex::Complex(double r){

real = r;

imag = 0;

}

Complex::Complex(double r, double i){

real = r;

imag = i;

}

Complex::Complex(Complex &obj){

real = obj.real;

imag = obj.imag;

}

Complex Complex::add(Complex c){

Complex sum;

sum.real = real + c.real;

sum.imag = imag + c.imag;

return sum;

}

Complex Complex::sub(Complex c){

Complex sub;

sub.real = real - c.real;

sub.imag = imag - c.imag;

return sub;

}

Complex Complex::mult(Complex c){//real = x1, y1 = imag c.real = x2, c.imag = y2

Complex mult;//Multiplication: z1. z2 = (x1x2 - y1y2) + i(x1y2 + x2y1)

mult.real = real\*c.real - imag\*c.imag;

mult.imag = real\*c.imag + c.real\*imag;

return mult;

}

Complex Complex::div(Complex c){

Complex div;//a=real,b=imag,c=c.real,d=c.imag

div.real = (real\*c.real+imag\*c.imag)/(c.real\*c.real + c.imag\*c.imag);

div.imag = (imag\*c.real-real\*c.imag)/(c.real\*c.real + c.imag\*c.imag);

return div;

}

void Complex::print(){

cout << '(' << real << ") + (" << imag << ")i" << endl;

}

void reset(double r1, double i1,Complex&a, Complex&c){

a.setReal(r1);

a.setImag(i1);

c.setReal(0);

c.setImag(0);

}

double Complex::getReal() const{

return real;

}

double Complex::getImag() const{

return imag;

}

void Complex::setReal(double r){

real = r;

}

void Complex::setImag(double i){

imag = i;

}

Complex class with friend functions

#include <iostream>

using namespace std;

class ComplexNumber {

public:

ComplexNumber(double real = 0.0, double imag = 0.0);

friend ComplexNumber operator+ (const ComplexNumber &a, const ComplexNumber &b);

friend ComplexNumber operator- (const ComplexNumber &a, const ComplexNumber &b);

friend ComplexNumber operator\* (const ComplexNumber &a, const ComplexNumber &b);

friend ComplexNumber operator/ (const ComplexNumber &a, const ComplexNumber &b);

friend ostream& operator<<(ostream &out, const ComplexNumber &c);

private:

float r;

float i;

};

ComplexNumber::ComplexNumber(double rr, double ii) : r(rr), i(ii) {

}

ComplexNumber operator+ (const ComplexNumber &a, const ComplexNumber &b) {

ComplexNumber result;

result.r = a.r + b.r;

result.i = a.i + b.i;

return result;

}

ComplexNumber operator- (const ComplexNumber &a, const ComplexNumber &b) {

ComplexNumber result;

result.r = a.r - b.r;

result.i = a.i - b.i;

return result;

}

ComplexNumber operator\* (const ComplexNumber &a, const ComplexNumber &b) {

ComplexNumber result;

result.r = (a.r \* b.r - a.i \* b.i);

result.i = (a.r \* b.i + a.i \* b.r);

return result;

}

ComplexNumber operator/ (const ComplexNumber &a, const ComplexNumber &b) {

ComplexNumber result;

result.r = (a.r \* b.r + a.i \* b.i) / (b.r \* b.r + b.i \* b.i);

result.i = (a.i \* b.r - a.r \* b.i) / (b.r \* b.r + b.i \* b.i);

return result;

}

ostream& operator<< (ostream &out, const ComplexNumber &c) {

out << c.r << "+" << c.i << "i";

return out;

}

# Palindrome without functions

#include <stdio.h>

#include <string.h>

int main()

{

char text[100];

int begin, middle, end, length = 0;

gets(text);

while ( text[length] != '\0' )

length++;

end = length - 1;

middle = length/2;

for( begin = 0 ; begin < middle ; begin++ )

{

if ( text[begin] != text[end] )

{

printf("Not a palindrome.\n");

break;

}

end--;

}

if( begin == middle )

printf("Palindrome.\n");

return 0;

}

# Palindrome with functions

#include <stdio.h>

int is\_palindrome(char\*);

void copy\_string(char\*, char\*);

void reverse\_string(char\*);

int string\_length(char\*);

int compare\_string(char\*, char\*);

int main()

{

char string[100];

int result;

printf("Enter a string\n");

gets(string);

result = is\_palindrome(string);

if ( result == 1 )

printf("\"%s\" is a palindrome string.\n", string);

else

printf("\"%s\" is not a palindrome string.\n", string);

return 0;

}

int is\_palindrome(char \*string)

{

int check, length;

char \*reverse;

length = string\_length(string);

reverse = (char\*)malloc(length+1);

copy\_string(reverse, string);

reverse\_string(reverse);

check = compare\_string(string, reverse);

free(reverse);

if ( check == 0 )

return 1;

else

return 0;

}

int string\_length(char \*string)

{

int length = 0;

while(\*string)

{

length++;

string++;

}

return length;

}

void copy\_string(char \*target, char \*source)

{

while(\*source)

{

\*target = \*source;

source++;

target++;

}

\*target = '\0';

}

void reverse\_string(char \*string)

{

int length, c;

char \*begin, \*end, temp;

length = string\_length(string);

begin = string;

end = string;

for ( c = 0 ; c < ( length - 1 ) ; c++ )

end++;

for ( c = 0 ; c < length/2 ; c++ )

{

temp = \*end;

\*end = \*begin;

\*begin = temp;

begin++;

end--;

}

}

int compare\_string(char \*first, char \*second)

{

while(\*first==\*second)

{

if ( \*first == '\0' || \*second == '\0' )

break;

first++;

second++;

}

if( \*first == '\0' && \*second == '\0' )

return 0;

else

return -1;

}