14.10.2020

Sprawozdanie
Programowanie Komputerów 3
Laboratorium 1

Paulina Czapla
Informatyka AEI,sem. 3, gr.4

1. Napisz klasę liczba zespolona z polami prywatnymi(IM,RE) i osobnymi metodami dostępowymi do tych pól jako metody publiczne (set,get) Zastosuj podział kodu źródłowego na pliki cpp i h. czy klasa liczba zespolona ma zawierać konstruktor przenoszący i operator przeniesienia ?

2. Do klasy liczba zespolona jako metody publiczne dodaj konstruktory: bezargumentowy jednoargumentowy wieloargumentowy kopiujący oraz destruktor każdy dodany element ma wypisywać informacje na ekranie gdy zostanie wywołany np. „konstruktor kopiujący zostal wywolany”

4. Dodaj do klasy liczba zespolona pole statyczne licznik które wskazuje liczbę istniejących w pamięci obiektów klasy zespolona w danej chwili

plik .h

```
//klasa Liczba_zespolona nie musi posiadać konstruktora przenoszącego i operatora
//przeniesienia, ponieważ nie wpłynie to znacząco wydajność
class Liczba_zespolona
{
	float IM;
	float RE;
	static int licznik;

public:
	Liczba_zespolona(float, float);
	Liczba_zespolona(float);
	Liczba_zespolona();
	Liczba_zespolona(Liczba_zespolona&);
	~Liczba_zespolona();
	void set_IM(float);
	void set_RE(float);
	float get_IM();
	float get_RE();
	void wyswietl();
	void pokaz_ile_obiektow();

};
```

plik .cpp

```
#include "Liczba_zespolona.h"
#include <iostream>

int Liczba_zespolona::licznik = 0;

Liczba_zespolona::Liczba_zespolona ( float rez, float imz)
{
	licznik++;
	IM = imz;
	RE = rez;
	std::cout << " Konstruktor wieloargumentowy zostal wywolany. " << std::endl;
}

Liczba_zespolona::Liczba_zespolona(float rez)
{
	licznik++;
```

```
	IM = 0;
	RE = rez;
	std::cout << " Konstruktor jednoargumentowy zostal wywolany. " << std::endl;
}
Liczba_zespolona::Liczba_zespolona() //: RE(0), IM(0)
{
	licznik++;
	IM = 0;
	RE = 0;
	std::cout << " Konstruktor bezargumentowy zostal wywolany. " << std::endl;
}
Liczba_zespolona::Liczba_zespolona(Liczba_zespolona& liczba)
{
	licznik++;
	IM = liczba.IM;
	RE = liczba.RE;
	std::cout << " Konstruktor kopiujacy zostal wywolany. " << std::endl;
}

Liczba_zespolona::~Liczba_zespolona()
{
	Liczba_zespolona::licznik--;
}

void Liczba_zespolona::set_IM(float IM)
{
	this->IM = IM;
}

void Liczba_zespolona::set_RE(float RE)
{
	this->RE = RE;
}

float Liczba_zespolona::get_IM()
{
	return IM;
}

float Liczba_zespolona::get_RE()
{
	return RE;
}

void Liczba_zespolona::wyswietl()
{
	std::cout << " IM=" << this->IM << " RE=" << this->RE<< std::endl;
}

void Liczba_zespolona::pokaz_ile_obiektow()
{
	std::cout << " W programie jest " << licznik << " liczb zespolonych. " << std::endl;
}
```

3. Stwórz klasę do obsługi dynamicznej tablicy wielowymiarowej liczb zespolonych. Użyj listę inicjalizacyjną do nadania rozmiarów tablicy zaimplantuj operator przeniesienia i konstruktor przenoszący dla tej klasy

plik .h

```
#include "Liczba_zespolona.h"
class Tablica_dwuwymiarowa
{
```

```
	int k;
	int w;
	Liczba_zespolona ** tab;

public:
	Tablica_dwuwymiarowa(int, int);
	Tablica_dwuwymiarowa();
	void wyswietl();
	void dodaj (Liczba_zespolona, int, int);
	Liczba_zespolona** zwroc_wskaznik();

	Tablica_dwuwymiarowa& operator=(Tablica_dwuwymiarowa&& other)
	{
		if (this != &other)
		{
			for (int i = 0; i < w; i++)
				delete[] tab[i];

			delete[] * tab;

			this->k = other.k;
			this->w = other.w;

			this->tab = other.tab;

			other.k = 0;
			other.w = 0;
		}
	return *this;
	}

	Tablica_dwuwymiarowa(Tablica_dwuwymiarowa&& other);
	~Tablica_dwuwymiarowa();
};
```

plik .cpp

```
#include "Tablica_dwuwymiarowa.h"
#include <iostream>

Tablica_dwuwymiarowa::Tablica_dwuwymiarowa(int kk, int ww) : k(kk), w(ww)
{
	tab = new Liczba_zespolona * [ww];
	for (int i = 0; i < ww; i++)
		tab[i] = new Liczba_zespolona[kk];
}

Tablica_dwuwymiarowa::Tablica_dwuwymiarowa() : tab(nullptr), w(0), k(0) {}

Tablica_dwuwymiarowa::Tablica_dwuwymiarowa(Tablica_dwuwymiarowa&& other) : tab(nullptr),
w(0), k(0)
{
	this->k = other.k;
	this->w = other.w;

	this->tab = other.tab;

	other.k = 0;
	other.w = 0;

	for (int i = 0; i < other.w; i++)
		delete[] other.tab[i];

	delete[] * other.tab;
```

```
}


void Tablica_dwuwymiarowa::wyswietl()
{
	for (int i = 0; i < w; i++)
	{
		for (int j = 0; j < k; j++)
		{
			tab[i][j].wyswietl();
			std::cout << std::endl;
		}

	}
}
void Tablica_dwuwymiarowa::dodaj( Liczba_zespolona zesp, int ww, int kk)
{
	tab[ww][kk] = zesp;
}

Liczba_zespolona** Tablica_dwuwymiarowa::zwroc_wskaznik()
{
	return tab;

}

Tablica_dwuwymiarowa::~Tablica_dwuwymiarowa()
{
	for (int i = 0; i < w; i++)
		delete[] tab[i];

	delete[] * tab;
}
```

5 . Stwórz klasę element listy której obiekt zawiera obiekt liczba zespolona i wskaźnik na kolejny element struktury danych. Użyj listę inicjalizacyjną do nadania jej wartości.kod zapisz w osobnych plikach cpp i h.

plik .h

```
#include "Liczba_zespolona.h"

class Element_listy
{
	Liczba_zespolona liczba;
	Element_listy* nastepny;

public:
	Element_listy(Liczba_zespolona&, Element_listy*);
	Element_listy(Liczba_zespolona&);
	Element_listy();
	Element_listy(Element_listy&);
	void zmien_nastepny(Element_listy*);
	void wyswietl();
	Element_listy* zwroc_nastepny();
};
```

plik .cpp

```
#include "Element_listy.h"
#include <iostream>
```

```
Element_listy::Element_listy(Liczba_zespolona& zespolona, Element_listy* nast) :
liczba(zespolona), nastepny(nast) {}

Element_listy::Element_listy(Liczba_zespolona& zespolona) : liczba(zespolona),
nastepny(nullptr) {}

Element_listy::Element_listy() : liczba(), nastepny(nullptr) {}

Element_listy::Element_listy(Element_listy& element) : liczba(element.liczba),
nastepny(element.nastepny) {}

void Element_listy::zmien_nastepny(Element_listy* nastepny)
{
	this->nastepny = nastepny;
}

void Element_listy::wyswietl()
{
	liczba.wyswietl();
}

Element_listy* Element_listy::zwroc_nastepny()
{
	return this->nastepny;
}
```

6. Stwórz listę obiektów liczba zespolona jako osobna klasa w osobnych plikach cpp i h klasa ma zawierać metody dodaj usuń wypisz

plik .h

```
#include "Element_listy.h"

class Lista
{
	Element_listy* head;
	int licznik;

public:
	Lista();
	~Lista();
	void dodaj(Element_listy* element);
	void usun(Element_listy* element);
	void wypisz();
};
```

plik .cpp

```
#include "Lista.h"
#include <iostream>

Lista::Lista()
{
	head = nullptr;
	licznik = 0;
}

Lista::~Lista() //usuwanie całej listy
{
    if (head)
```

```
    {
        Element_listy* tmp;

        while (head)
        {
            tmp = head->zwroc_nastepny();
            delete head;
            head = tmp;
        }
    }
}
void Lista::dodaj(Element_listy* wsk)
{
       wsk->zmien_nastepny(Lista::head);
       this->head = wsk;
       licznik++;
}

void Lista::usun(Element_listy* wsk)
{
    Element_listy* tmp(head);

    if (head == wsk)
    {
        tmp = head->zwroc_nastepny();
        delete head;
        head = tmp;
    }
    else
    {
        Element_listy* tmp2, *poprzedni(head) ;
        while (tmp)
        {
            if (tmp == wsk)
            {
                tmp2 = tmp->zwroc_nastepny();
                delete tmp;
                poprzedni->zmien_nastepny(tmp2);
                break;
            }
            poprzedni = tmp;
            tmp = tmp->zwroc_nastepny();
        }
    }
}

void Lista::wypisz()
{
    Element_listy* tmp = head;
    int i = 1;
    while (head)
    {
        std::cout << " Element "<<i<<"= ";
        head->wyswietl();
        std::cout << std::endl;
        head = head->zwroc_nastepny();
        i++;
    }
    std::cout << std::endl << " Laczna liczba elementow: " << licznik;
    head = tmp;
}
```

7. zaimplantuj klasę zawierającą z wskaźnik na liczbę zespoloną mającą konstruktor przenoszący i operator przeniesienia

plik .h

```
#include "Liczba_zespolona.h"

class Wskaznik
{
	Liczba_zespolona* liczba;
public:
	Wskaznik(Liczba_zespolona);
	Wskaznik(Wskaznik&&);
	Wskaznik();
	~Wskaznik();
	Wskaznik& operator=(Wskaznik&& other)
	{
		if (this != &other)
		{
			delete liczba;

			this->liczba = other.liczba;
			this->liczba->set_IM(other.liczba->get_IM());
			this->liczba->set_RE(other.liczba->get_RE());

			other.liczba= nullptr;

			return *this;
		}

	}
	void wyswietl();

};
```

plik .cpp

```
#include "Wskaznik.h"
#include <iostream>


Wskaznik::Wskaznik(Liczba_zespolona liczba)
{
	this->liczba = new Liczba_zespolona;
	*(this->liczba) = liczba;
}

Wskaznik::Wskaznik()
{
	liczba = nullptr;
}

Wskaznik::Wskaznik(Wskaznik&& other) : liczba(nullptr)
{
	this->liczba = other.liczba;
	this->liczba->set_IM( other.liczba->get_IM());
	this->liczba->set_RE( other.liczba->get_RE());

	other.liczba = nullptr;

}

Wskaznik::~Wskaznik()
{
	delete[] this->liczba;
```

```
}

void Wskaznik::wyswietl()
{
	liczba->wyswietl();
}
```

Zadanie – Kapelusz czarnoksiężnika (wykonanie zadań 1-5 dla klasy Zwierzątko)
Zad 1,2,4

plik .h

```
//klasa Zwierzatko powinna posiadać konstruktor przenoszący i operator przeniesienia
class Zwierzatko
{
	char* zwierzatko;
	int numer;
	static int licznik;

public:
	Zwierzatko(char*);
	Zwierzatko();
	Zwierzatko(Zwierzatko&);
	~Zwierzatko();
	void zmien_zwierzatko(char*);
	char* zwroc_zwierzatko();
	int zwroc_numer();

};
```

plik .cpp

```
#include "Zwierzatko.h"
#include <cstring>
#include <iostream>

int Zwierzatko::licznik = 0;


Zwierzatko::Zwierzatko(char * zwierzatko_)
{
	licznik++;
	this->numer = licznik;
	this->zwierzatko = new char[strlen(zwierzatko_)+1];
	strcpy_s(zwierzatko, strlen(zwierzatko_) + 1,  zwierzatko_);

	std::cout << " Konstruktor jednoargumentowy zostal wywolany. " << std::endl;
}
Zwierzatko::Zwierzatko()
{
	licznik++;
	this->numer = licznik;
	const char* krolik = "krolik";
	this->zwierzatko = new char[strlen(krolik) + 1];
	strcpy_s ( zwierzatko,strlen(krolik)+1, krolik);

	std::cout << " Konstruktor bezargumentowy zostal wywolany. " << std::endl;
}
Zwierzatko::Zwierzatko(Zwierzatko& zwierzatko_)
{
	licznik++;
	this->numer = licznik;
	this->zwierzatko = new char[strlen(zwierzatko_.zwierzatko)];
```

```
	strcpy_s( zwierzatko, strlen(zwierzatko_.zwierzatko) + 1, zwierzatko_.zwierzatko);

	std::cout << " Konstruktor kopiujacy zostal wywolany. " << std::endl;
}


Zwierzatko::~Zwierzatko()
{
	Zwierzatko::licznik--;
	delete[] zwierzatko;
}

void Zwierzatko::zmien_zwierzatko(char* zwierzatko_)
{
	delete[] zwierzatko;
	char* zwierzatko = new char[strlen(zwierzatko_) + 1];
	strcpy_s(zwierzatko, strlen(zwierzatko_) + 1, zwierzatko_);
}


char* Zwierzatko::zwroc_zwierzatko()
{
	return zwierzatko;
}

int Zwierzatko::zwroc_numer()
{
	return numer;
}
```

Zad. 3

plik .h

```
#include "Zwierzatko.h"
class Tablica_dwm_zwierzatka
{

	int k;
	int w;
	Zwierzatko** tab;

public:
	Tablica_dwm_zwierzatka(int, int);
	Tablica_dwm_zwierzatka();
	void dodaj(Zwierzatko, int, int);
	Zwierzatko** zwroc_wskaznik();

	Tablica_dwm_zwierzatka& operator=(Tablica_dwm_zwierzatka&& other)
	{
		if (this != &other)
		{
			for (int i = 0; i < w; i++)
				delete[] tab[i];

			delete[] * tab;

			this->k = other.k;
			this->w = other.w;

			this->tab = other.tab;

			other.k = 0;
```

```
                other.w = 0;
            }
            return *this;
        }

        Tablica_dwm_zwierzatka(Tablica_dwm_zwierzatka&& other);
        ~Tablica_dwm_zwierzatka();
        Zwierzatko wyswietl(int, int);
};
```

plik .cpp

```
#include "Tablica_dwm_zwierzatka.h"

#include <iostream>

Tablica_dwm_zwierzatka::Tablica_dwm_zwierzatka(int kk, int ww) : k(kk), w(ww)
{
        tab = new Zwierzatko * [ww];
        for (int i = 0; i < ww; i++)
            tab[i] = new Zwierzatko[kk];
}

Tablica_dwm_zwierzatka::Tablica_dwm_zwierzatka() : tab(nullptr), w(0), k(0) {}

Tablica_dwm_zwierzatka::Tablica_dwm_zwierzatka(Tablica_dwm_zwierzatka&& other) :
tab(nullptr), w(0), k(0)
{
        this->k = other.k;
        this->w = other.w;

        this->tab = other.tab;

        other.k = 0;
        other.w = 0;

        for (int i = 0; i < other.w; i++)
            delete[] other.tab[i];

        delete[] * other.tab;

}


void Tablica_dwm_zwierzatka::dodaj(Zwierzatko zesp, int ww, int kk)
{
        tab[ww][kk] = zesp;
}

Zwierzatko** Tablica_dwm_zwierzatka::zwroc_wskaznik()
{
        return tab;

}

Tablica_dwm_zwierzatka::~Tablica_dwm_zwierzatka()
{
        for (int i = 0; i < w; i++)
            delete[] tab[i];

        delete[] * tab;
}
Zwierzatko Tablica_dwm_zwierzatka::wyswietl(int ww, int kk)
{
```

```
	return tab[ww][kk];
}
```

Zad. 5

plik .h

```
class Element_listy_zwierzatko
{
	Zwierzatko zwierzatko;
	Element_listy_zwierzatko* nastepny;

public:
	Element_listy_zwierzatko(Zwierzatko&, Element_listy_zwierzatko*);
	Element_listy_zwierzatko(Zwierzatko&);
	Element_listy_zwierzatko();
	Element_listy_zwierzatko(Element_listy_zwierzatko&);
	void zmien_nastepny(Element_listy_zwierzatko*);
	void wyswietl();
	Element_listy_zwierzatko* zwroc_nastepny();

};
```

plik .cpp

```
#include "Element_listy_zwierzatko.h"

#include <iostream>

Element_listy_zwierzatko::Element_listy_zwierzatko(Zwierzatko& zespolona,
Element_listy_zwierzatko* nast) : zwierzatko(zespolona), nastepny(nast) {}

Element_listy_zwierzatko::Element_listy_zwierzatko(Zwierzatko& zespolona) :
zwierzatko(zespolona), nastepny(nullptr) {}

Element_listy_zwierzatko::Element_listy_zwierzatko() : zwierzatko(), nastepny(nullptr) {}

Element_listy_zwierzatko::Element_listy_zwierzatko(Element_listy_zwierzatko& element) :
zwierzatko(element.zwierzatko), nastepny(element.nastepny) {}

void Element_listy_zwierzatko::zmien_nastepny(Element_listy_zwierzatko* nastepny)
{
	this->nastepny = nastepny;
}

void Element_listy_zwierzatko::wyswietl()
{
	for (int i = 0; i < strlen(zwierzatko.zwroc_zwierzatko()); i++)
		std::cout << zwierzatko.zwroc_zwierzatko()[i];

	std::cout << std::endl;
}

Element_listy_zwierzatko* Element_listy_zwierzatko::zwroc_nastepny()
{
	return this->nastepny;
}
```

Zad.6

plik .h

```
#include "Element_listy_zwierzatko.h"

class Lista_zwierzatek
{
	Element_listy_zwierzatko* head;
	int licznik;

public:
	Lista_zwierzatek();
	~Lista_zwierzatek();
	void dodaj(Element_listy_zwierzatko* element);
	void usun(Element_listy_zwierzatko* element);
	void wypisz();
	void wyswietl(Zwierzatko* zwierzatko);
};
```

plik .cpp

```
#include "Lista_zwierzatek.h"

#include <iostream>

Lista_zwierzatek::Lista_zwierzatek()
{
    head = nullptr;
    licznik = 0;
}

Lista_zwierzatek::~Lista_zwierzatek() //usuwanie całej listy
{
    if (head)
    {
        Element_listy_zwierzatko* tmp;

        while (head)
        {
            tmp = head->zwroc_nastepny();
            delete head;
            head = tmp;
        }
    }
}
void Lista_zwierzatek::dodaj(Element_listy_zwierzatko* wsk)
{
    wsk->zmien_nastepny(Lista_zwierzatek::head);
    this->head = wsk;
    licznik++;
}

void Lista_zwierzatek::usun(Element_listy_zwierzatko* wsk)
{
    Element_listy_zwierzatko* tmp(head);

    if (head == wsk)
    {
        tmp = head->zwroc_nastepny();
        delete head;
        head = tmp;
    }
    else
```

```
    {
        Element_listy_zwierzatko* tmp2, * poprzedni(head);
        while (tmp)
        {
            if (tmp == wsk)
            {
                tmp2 = tmp->zwroc_nastepny();
                delete tmp;
                poprzedni->zmien_nastepny(tmp2);
                break;
            }
            poprzedni = tmp;
            tmp = tmp->zwroc_nastepny();
        }
    }
}

void Lista_zwierzatek::wypisz()
{
    Element_listy_zwierzatko* tmp = head;
    int i = 1;
    while (head)
    {
        std::cout << " Element " << i << "= ";
        head->wyswietl();
        std::cout << std::endl;
        head = head->zwroc_nastepny();
        i++;
    }
    std::cout << std::endl << " Laczna liczba elementow: " << licznik;
    head = tmp;
}

void Lista_zwierzatek::wyswietl(Zwierzatko* zwierzatko)
{
    for (int i = 0; i < strlen(zwierzatko->zwroc_zwierzatko()); i++)
        std::cout << zwierzatko->zwroc_zwierzatko()[i];

    std::cout << std::endl;
}
```