Міністерство освіти і науки України
Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

ЗВІТ

про виконання комп’ютерного практикуму № 5

з дисципліни «Програмування»

Виконав:
Студент І курсу
Групи КА-95
Петренко Д.М.
Варіант № 18
Перевірила:
Гуськова В.Г.

Київ – 2020

1. Завдання:

Варіант 18.

Описати клас «Текст», у якому передбачити поле з заголовком, динамічний масив об'єктів типу «Фраза» та розмірністю цього масиву.

Передбачити конструктори, деструктор і усі функції, які вважаєте за доцільне. Обовязково визначити оператор індексації для доступу до інформації про фразу. Для одного з похідних класів передбачити можливість перетворення типу до об'єкту базового і навпаки.

Функцію `view()` у класах «Фраза» та усіх похідних переробити на віртуальну.

2. Лістинг програми:

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstdio>
using namespace std;

class Alphabet {
public:
	char* letters;
	char* signs;
	double l1, s1;

public:
	Alphabet();
	Alphabet(char*, char*, double, double);
	Alphabet(Alphabet&);
	~Alphabet();
	friend ostream& operator<<(ostream& out, Alphabet& v);
	friend istream& operator>>(istream& in, Alphabet& v);
	Alphabet& Set_Letters(char*);
```

```
        Alphabet& Set_Signs(char*);
        Alphabet& Setl1(double);
        Alphabet& Sets1(double);
        char* Get_Letters();
        char* Get_Signs();
        double Getl1();
        double Gets1();
        void PrintAlphabet();
        void ShortPrintAlphabet();
        Alphabet& operator=(const Alphabet&);
};


Alphabet::Alphabet() {
        letters = new char[50];
        signs = new char[50];
        strcpy(letters, "noletters");
        strcpy(signs, "nosigns");
        l1 = s1 = 0;
}
Alphabet::Alphabet(char* lett, char* sign, double l2, double s2) {
        letters = new char[strlen(lett) + 1];
        signs = new char[strlen(sign) + 1];
        strcpy(letters, lett);
        strcpy(signs, sign);
        l1 = l2;
        s1 = s2;
}
Alphabet::Alphabet(Alphabet& a) {
```

```
        l1 = a.l1;
        s1 = a.s1;
        letters = new char[strlen(a.letters) + 1];
        signs = new char[strlen(a.signs) + 1];
        strcpy(letters, a.letters);
        strcpy(signs, a.signs);
}
Alphabet::~Alphabet() { if (letters) delete[] letters; if (signs) delete[] signs; };
ostream& operator<<(ostream& out, Alphabet& v) {
        return out << "" << v.letters << "" << ',' << ' ' << "" << v.signs << "" << ',' <<
' ' << '(' << v.l1 << ',' << v.s1 << ')' << ';' << endl;
}
istream& operator>>(istream& in, Alphabet& v) {
        return in >> v.letters >> v.signs;
}
Alphabet& Alphabet::Set_Letters(char* lett) {
        delete[]letters;
        letters = new char[strlen(lett) + 1];
        strcpy(letters, lett);
        return*this;
};
Alphabet& Alphabet::Set_Signs(char* sign) {
        delete[] signs;
        signs = new char[strlen(sign) + 1];
        strcpy(signs, sign);
        return*this;
};
Alphabet& Alphabet::Setl1(double l2) { l1 = l2; return*this; };
Alphabet& Alphabet::Sets1(double s2) { s1 = s2; return*this; };
```

```
char* Alphabet::Get_Letters() { return letters; };
char* Alphabet::Get_Signs() { return signs; };
double Alphabet::Getl1() { return l1; };
double Alphabet::Gets1() { return s1; };
void Alphabet::PrintAlphabet() {
	cout << letters << ' ' << signs << ' ' << l1 << ' ' << s1 << endl;
};
void Alphabet::ShortPrintAlphabet() {
	cout << l1 << ' ' << s1 << endl;
}
Alphabet& Alphabet::operator=(const Alphabet& other)
{
	Alphabet* temp = new Alphabet;
	strcpy(temp->letters, other.letters);
	strcpy(temp->signs, other.signs);
	temp->l1 = other.l1;
	temp->s1 = other.s1;
	return *temp;
}


class Phrase {
public:
	char* phrase;
	Alphabet alph;
public:
	Phrase();
	Phrase(char*, Alphabet&);
	Phrase(Phrase&);
```

```
        ~Phrase();
        friend ostream& operator<<(ostream& out, Phrase& v);
        friend istream& operator>>(istream& in, Phrase& v);
        friend bool operator==(const Phrase& a, const Phrase& b);
        Phrase& setphrase(char*);
        Phrase& setalph(char*, char*, double, double);
        char* getphrase();
        Alphabet& getalph(char* gl1, char* gl2, double gl3, double gl4);
        void printPhrase();
        void shortPrintPhrase();

        virtual void view() { cout << "Phrase " << getphrase(); }

        Phrase& operator=(const Phrase&);
};



Phrase::Phrase() {
        phrase = new char[50];
        strcpy(phrase, "nophrase");
}
Phrase::Phrase(char* phr, Alphabet& alpha) :alph(alpha) {
        phrase = new char[strlen(phr) + 1];
        strcpy(phrase, phr);
}
Phrase::Phrase(Phrase& b) : alph(b.alph) {
        phrase = new char[strlen(b.phrase) + 1];
        strcpy(phrase, b.phrase);
```

```
}
Phrase::~Phrase() { if (phrase) delete[] phrase; };
bool operator==(const Phrase& a, const Phrase& b) {
	return (a.phrase == b.phrase);
}
ostream& operator<<(ostream& out, Phrase& v) {
	return out << "" << v.phrase << "" << ';' << endl;
}
istream& operator>>(istream& in, Phrase& v) {
	return in >> v.phrase >> v.alph;
}
Phrase& Phrase::setphrase(char* phr) {
	delete[]phrase;
	phrase = new char[strlen(phr) + 1];
	strcpy(phrase, phr);
	return*this;
};
Phrase& Phrase::setalph(char* sl1, char* sl2, double sl3, double sl4) {
	alph.Set_Letters(sl1);
	alph.Set_Signs(sl2);
	alph.Setl1(sl3);
	alph.Sets1(sl4);
	return*this;
}
char* Phrase::getphrase() { return phrase; };
Alphabet& Phrase::getalph(char* gl1, char* gl2, double gl3, double gl4) {
	alph.Get_Letters();
	alph.Get_Signs();
	alph.Getl1();
```

```
        alph.Gets1();
        return alph;
}
void Phrase::printPhrase() {
        cout << phrase << endl;
        alph.PrintAlphabet();
};
void Phrase::shortPrintPhrase() {
        cout << phrase << endl;
}


class Number : public Phrase {
private:
        double number_system;
        double length_fraction;
public:
        Number();
        Number(char*, Alphabet&, double, double);
        Number(Number&);
        ~Number();
        friend Number operator+(Number&, Number&);
        friend bool operator<(Number&, Number&);
        friend ostream& operator<<(ostream& out, Number& v);
        Number& setnumber_system(double);
        Number& setlength_fraction(double);
        double getnumber_system();
        double getlength_fraction();
        void printNumber();
```

```
        void View_Number();


        void view()
        {
                cout << "\n" << "View\n";
                cout << "Phrase " << getphrase() << " have basis of the calculus system " << getnumber_system() << "\n";
        }


        Number& operator=(const Phrase&);


};
Number::Number() {
        number_system = length_fraction = 0;
}
Number::Number(char* phr, Alphabet& alpha, double ns, double lf) :Phrase(phr, alpha) {
        number_system = ns;
        length_fraction = lf;
}
Number::Number(Number& n) : Phrase(n) {
        number_system = n.number_system;
        length_fraction = n.length_fraction;
}
Number::~Number() {
}
Number operator+(Number& a, Number& b) {
        Number temp;
        temp.number_system = a.number_system + b.number_system;
```

```
        temp.length_fraction = a.length_fraction + b.length_fraction;
        return temp;
}
bool operator<(const Number& l1, const Number& l2) {
        return (l1 < l2);
}
ostream& operator<<(ostream& out, Number& v) {
        return out << '[' << v.number_system << ']' << ',' << ' ' << '[' <<
v.length_fraction << ']' << ';' << endl;
}
Number& Number::setnumber_system(double ns) {
        number_system = ns;
        return*this;
}
Number& Number::setlength_fraction(double lf) {
        length_fraction = lf;
        return*this;
}
double Number::getnumber_system() {
        return number_system;
}
double Number::getlength_fraction() {
        return length_fraction;
}
void Number::printNumber() {
        Phrase::printPhrase();
        cout << number_system << endl;
        cout << length_fraction << endl;
}
```

```
void Number::View_Number() {
	Phrase::printPhrase();
	cout << number_system << endl;
}
class Sentence : public Phrase {
	double Len_Alph;
	int Ignor_Register;
public:
	Sentence();
	Sentence(char*, Alphabet&, double, int);
	Sentence(Sentence&);
	~Sentence();
	friend ostream& operator<<(ostream& out, Sentence& v);
	Sentence& setlen_Alph(double);
	Sentence& setignor_register(int);
	double getlen_Alph();
	int getignor_register();
	void printSentence();
	void View_Sentence();

	void view()
	{
		cout << "\n" << "View\n";
		cout << getphrase() << " - have " << getlen_Alph() << " symbols\n";
	}
};

Sentence::Sentence() {
	Len_Alph = Ignor_Register = 0;
```

```
}
Sentence::Sentence(char* phr, Alphabet& alpha, double lA, int ir) :Phrase(phr, alpha) {
	Len_Alph = lA;
	Ignor_Register = ir;
}
Sentence::Sentence(Sentence& s) : Phrase(s) {
	Len_Alph = s.Len_Alph;
	Ignor_Register = s.Ignor_Register;
}
Sentence::~Sentence() {
}
ostream& operator<<(ostream& out, Sentence& v) {
	return out << '[' << v.Len_Alph << ']' << ',' << ' ' << '(' << v.Ignor_Register << ')' << ';' << endl;
}
Sentence& Sentence::setlen_Alph(double lA) {
	Len_Alph = lA;
	return*this;
}
Sentence& Sentence::setignor_register(int ir) {
	Ignor_Register = ir;
	return*this;
}
double Sentence::getlen_Alph() {
	return Len_Alph;
}
int Sentence::getignor_register() {
	return Ignor_Register;
```

```
}
void Sentence::printSentence() {
	Phrase::printPhrase();
	cout << Len_Alph << endl;
	cout << Ignor_Register << endl;
}
void Sentence::View_Sentence() {
	Phrase::printPhrase();
	cout << Len_Alph << endl;
}
class Text
{
private:
	char* name;
	Phrase* stack;
	int dimension;
public:
	Text();
	Text(const Text&);
	Text(int);

	~Text();

	Phrase& operator[](int index) { return stack[index]; }
};
int main()
{
	double l2, s2;
	char lett[50];
```

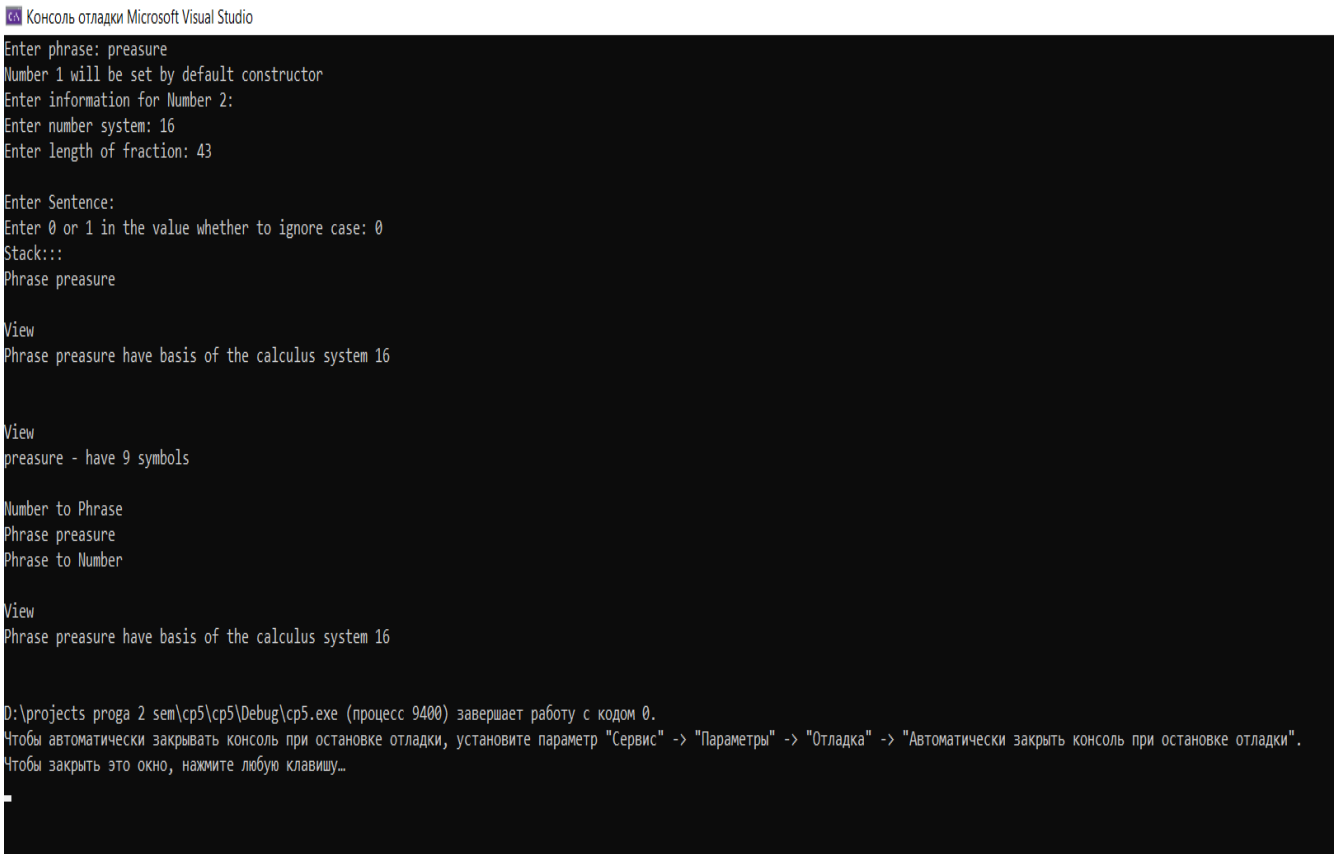
```
        char sign[50];
        char phr1[50];
        char phr2[50];
        double ns, lf1, lf2, lA;
        int ir;
        int choice;
        Alphabet obj1;
        cout << "Alphabet 1 will be set by default constructor" << "\nEnter information for Alphabet 2:" << endl;
        cout << "Enter letters: ";  cin >> lett;    l2 = strlen(lett);
        cout << "Enter signs: ";  cin >> sign;    s2 = strlen(sign);
        Alphabet obj2(lett, sign, l2, s2);
        Alphabet obj3 = obj2;
        cout << "\nEnter information for Alphabet 3:" << endl;
        cout << "Enter letters: ";  cin >> lett;    l2 = strlen(lett);
        cout << "Enter signs: ";  cin >> sign;    s2 = strlen(sign);
        obj3.Set_Letters(lett).Set_Signs(sign).Setl1(l2).Sets1(s2);
        cout << "\nThere are Alphabet 1, Alphabet 2, Alphabet 3: " << "\nAlphabet 1: " << obj1 << "Alphabet 2: " << obj2 << "Alphabet 3: " << obj3 << endl;


        cout << "Enter information for Phrase 2: " << "\nEnter phrase: ";
        cin >> phr1;
        Phrase p2(phr1, obj3);



        cout << "Number 1 will be set by default constructor" << "\nEnter information for Number 2:" << endl;
        cout << "Enter number system: ";      cin >> ns;
        cout << "Enter length of fraction: ";  cin >> lf1;
```

```
        Number n2(phr1, obj3, ns, lf1);


        lA = l2 + s2;
        cout << "\nEnter Sentence: " << "\nEnter 0 or 1 in the value whether to
ignore case: ";  cin >> ir;
        Sentence sen2(phr1, obj3, lA, ir);




        Phrase* Stack[3];


        Stack[0] = &p2;
        Stack[1] = &n2;
        Stack[2] = &sen2;


        cout << "Stack:::" << endl;
        Stack[0]->view();
        cout << endl;
        Stack[1]->view();
        cout << endl;
        Stack[2]->view();
        cout << endl;


        cout << "Number to Phrase" << endl;
        p2 = n2;
        p2.view();
        cout << endl;
```

```
	cout << "Phrase to Number" << endl;
	n2 = p2;
	n2.view();
	cout << endl;

	return 0;
}



Text::Text()
{
	dimension = 0;
	name = new char[10];
	strcpy(name, "No name");
}
Text::Text(const Text& other)
{
	name = new char[10];
	strcpy(name, other.name);
	dimension = other.dimension;
	for (int i = 0; i < dimension; i++)
	{
		stack[i] = other.stack[i];
	}
}
Text::Text(int d)
{
	dimension = d;
}
```

```
Text::~Text()
{
	delete[] name;
}

Number& Number::operator=(const Phrase& otherPhrase)
{
	Number* temp = new Number;
	strcpy(temp->phrase, otherPhrase.phrase);
	return *temp;
}

Phrase& Phrase::operator=(const Phrase& otherPhrase)
{
	Phrase* temp = new Phrase;
	strcpy(temp->phrase, otherPhrase.phrase);
	return *temp;
}
```

3. Приклад роботи програми:



4. Висновок: у даній роботі я навчився визначати оператор індексації для доступу до інформації, реалізації перетворення типу об’єкта класу і використовувати віртуальні функції для вирішення певних задач в мові програмування C++ .