ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 8

Дисциплина:

Объектно-ориентированное программирование

Тема:

Указатели и массивы.

Выполнил(а): студент(ка) группы 211-7210
Салов Д.К.
(Фамилия И.О.)

Дата, подпись 12.06.22
(Дата) (Подпись)

Проверил: _____ _____
(Фамилия И.О., степень, звание)
(Оценка)
Дата, подпись _____ _____
(Дата) (Подпись)

Замечания: _____
_____
_____

Москва

2021

Цель: Получить практические навыки в использовании указателей и работе с массивами через указатели.

Задания:

4. Добавьте деструктор в программу связного списка. Он должен удалять все элементы списка при удалении объекта класса linklist. Элементы должны удаляться по очереди, в соответствии с их расположением в списке. Протестируйте деструктор путем вывода сообщения об удалении каждого из элементов списка; удалено должно быть также количество элементов, какое было положено в список (деструктор вызывается автоматически для каждого существующего объекта).

```
// lab9_4.cpp
// linked list includes destructor
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
struct link                             //one element of list
   {
   int data;                            //data item
   link* next;                          //pointer to next link
   };
////////////////////////////////////////////////////////////////
class linklist                          //a list of links
   {
   private:
      link* first;                      //pointer to first link
   public:
      linklist()                        //no-argument constructor
         { first = NULL; }              //no first link
      ~linklist();                      //destructor
      void additem(int d);              //add data item (one link)
      void display();                   //display all links
   };
//-------------------------------------------------------------
void linklist::additem(int d)           //add data item
   {
   link* newlink = new link;            //make a new link
   newlink->data = d;                   //give it data
   newlink->next = first;               //it points to next link
   first = newlink;                     //now first points to this
   }
//-------------------------------------------------------------
void linklist::display()                //display all links
   {
   link* current = first;               //set ptr to first link
   while( current != NULL )             //quit on last link
      {
      cout << endl << current->data;    //print data
      current = current->next;          //move to next link
      }
   }
//-------------------------------------------------------------
linklist::~linklist()                   //destructor
   {
   link* current = first;               //set ptr to first link
   while( current != NULL )             //quit on last link
      {
      link* temp = current;             //save ptr to this link
      current = current->next;          //get ptr to next link
      delete temp;                      //delete this link
```

```
      }
   }
////////////////////////////////////////////////////////////
int main()
   {
   linklist li;       //make linked list

   li.additem(25);    //add four items to list
   li.additem(36);
   li.additem(49);
   li.additem(64);

   li.display();      //display entire list
   cout << endl;
   return 0;
   }
```

5. Предположим, что в функции main() определены три локальных массива одинакового размера и типа (скажем, float). Первые два уже инициализированы значениями. Напишите функцию addarrays(), которая принимает в качестве аргументов адреса грех массивов, складывает соответствующие элементы двух массивов и помещает результат в третий массив. Четвертым аргументом этой функции может быть размерность массивов. На всем протяжении программы используйте указатели.

6. Создайте свою версию библиотечной функции strcmp(s1, s2), которая сравнивает две строки и возвращает -1, если s1 идет первой по алфавиту, О, если в s1 и s2 одинаковые значения, и 1, если s2 идет первой по алфавиту. Назовите вашу функцию compstr(). Она должна принимать в качестве аргументов два указателя на строки char *, сравнивать эти строки посимвольно и возвращать число int. Напишите функцию main() для проверки работы вашей функции с разными строками. Используйте указатели во всех возможных ситуациях.

7. Модифицируйте класс person из программы, представленной ниже, чтобы он включал в себя не только имя человека, но и сведения о его зарплате в виде поля salary типа float. Вам будет необходимо изменить методы setName() и printName() на setData() и printData(), включив в них возможность ввода и вывода значения salary, как это можно сделать с именем. Вам также понадобится метод getSalary(). Используя указатели, напишите функцию salsortQ, которая сортирует указатели массива persPtr по значениям зарплаты. Попробуйте вместить всю сортировку в функцию salsort(), не вызывая других функций, как это сделано в программе PERS0RT. При этом не забывайте, что операция -> имеет больший приоритет, чем операция *, и вам нужно будет написать

```
if ( (*(pp+j))->getSalary() > (*(pp+k))->getSalary() )
{  /* меняем указатели местами */
```

```
// persort.cpp
// sorts person objects using array of pointers
#include <iostream>
#include <string>                 //for string class
using namespace std;
////////////////////////////////////////////////////////////
class person                      //class of persons
   {
   protected:
      string name;                //person's name
   public:
      void setName()              //set the name
         { cout << "Enter name: "; cin >> name; }
      void printName()            //display the name
         { cout << endl << name; }
      string getName()            //return the name
         { return name; }
   };
////////////////////////////////////////////////////////////
int main()
```

```
   {
   void bsort(person**, int);     //prototype
   person* persPtr[100];          //array of pointers to persons
   int n = 0;                     //number of persons in array
   char choice;                   //input char

   do {                           //put persons in array
      persPtr[n] = new person;    //make new object
      persPtr[n]->setName();      //set person's name
      n++;                        //count new person
      cout << "Enter another (y/n)? "; //enter another
      cin >> choice;              //   person?
      }
   while( choice=='y' );          //quit on 'n'

   cout << "\nUnsorted list:";
   for(int j=0; j<n; j++)         //print unsorted list
      { persPtr[j]->printName(); }

   bsort(persPtr, n);             //sort pointers

   cout << "\nSorted list:";
   for(j=0; j<n; j++)             //print sorted list
      { persPtr[j]->printName(); }
   cout << endl;
   return 0;
   }  //end main()
//--------------------------------------------------------------
void bsort(person** pp, int n)    //sort pointers to persons
   {
   void order(person**, person**);  //prototype
   int j, k;                      //indexes to array

   for(j=0; j<n-1; j++)           //outer loop
      for(k=j+1; k<n; k++)        //inner loop starts at outer
            order(pp+j, pp+k);       //order the pointer contents
   }
//--------------------------------------------------------------
void order(person** pp1, person** pp2)  //orders two pointers
   {                              //if 1st larger than 2nd,
   if( (*pp1)->getName() > (*pp2)->getName() )
      {
      person* tempptr = *pp1;     //swap the pointers
      *pp1 = *pp2;
      *pp2 = tempptr;
      }
   }
```

8. Исправьте функцию additem() из программы связного списка так, чтобы она добавляла новый элемент в конец списка, а не в начало. Это будет означать, что первый вставленный элемент будет выведен первым и результат работы программы будет следующим:

25
36
49
64

Для того чтобы добавить элемент, вам необходимо будет пройти по цепи до конца списка, а затем изменить указатель последнего элемента так, чтобы он указывал на новый элемент.

```
// linklist.cpp
// linked list
#include <iostream>
```

```
using namespace std;
////////////////////////////////////////////////////////////////
struct link                             //one element of list
   {
   int data;                            //data item
   link* next;                          //pointer to next link
   };
////////////////////////////////////////////////////////////////
class linklist                          //a list of links
   {
   private:
      link* first;                      //pointer to first link
   public:
      linklist()                        //no-argument constructor
         { first = NULL; }              //no first link
      void additem(int d);              //add data item (one link)
      void display();                   //display all links
   };
//-----------------------------------------------------------------
void linklist::additem(int d)           //add data item
   {
   link* newlink = new link;            //make a new link
   newlink->data = d;                   //give it data
   newlink->next = first;               //it points to next link
   first = newlink;                     //now first points to this
   }
//-----------------------------------------------------------------
void linklist::display()                //display all links
   {
   link* current = first;               //set ptr to first link
   while( current != NULL )             //quit on last link
      {
      cout << current->data << endl;    //print data
      current = current->next;          //move to next link
      }
   }
////////////////////////////////////////////////////////////////
int main()
   {
   linklist li;       //make linked list

   li.additem(25);    //add four items to list
   li.additem(36);
   li.additem(49);
   li.additem(64);

   li.display();      //display entire list
   return 0;
   }
```

9. Допустим, что нам нужно сохранить 100 целых чисел так, чтобы иметь к ним легкий доступ. Допустим, что при этом у нас есть проблема: память нашего компьютера так фрагментирована, что может хранить массив, наибольшее количество элементов в котором равно десяти (такие проблемы действительно появляются, хотя обычно это происходит с объектами, занимающими большое количество памяти). Вы можете решить эту проблему, определив 10 разных массивов по 10 элементов в каждом и массив из 10 указателей на эти массивы. Массивы будут иметь имена a0, a1, a2 и т. д. Адрес каждого массива будет сохранен в массиве указателей типа int*, который называется ap. Вы сможете получить доступ к отдельному целому используя выражение ap[j] [к], где j является номером элемента массива указателей, а к — номером элемента в массиве, на который этот указатель указывает. Это похоже на двумерный массив, но в действительности является группой одномерных массивов.

Заполните группу массивов тестовыми данными (скажем, номерами 0, 10, 20 и т. д.), а затем выведите их, чтобы убедиться, что все работает правильно.

10. Описанный в упражнении 9 подход нерационален, так как каждый из 10 массивов объявляется отдельно, с использованием отдельного имени, и каждый адрес получают отдельно. Вы можете упростить программу, используя операцию new, которая позволит вам выделить память для массивов в цикле и одновременно связать с ними указатели:

```
for ( j = 0: j < NUMARRAYS; j++ ) // создаем NUMARRAYS массивов
    *( ap + j ) = new int [ MAXSIZE ]: //no MAXSIZE целых чисел в каждом
```

Перепишите программу упражнения 9, используя этот подход. Доступ к отдельному элементу массивов вы сможете получить, используя то же выражение, что и в упражнении 9, или аналогичное выражение с указателями: *(*(ap+j)+k).

11. Создайте класс, который позволит вам использовать 10 отдельных массивов из упражнения 10 как один одномерный массив, допуская применение операций массива. То есть мы можем получить доступ к элементам массива, записав в функции main() выражение типа a[j], а методы класса могут получить доступ к полям класса, используя двухшаговый подход. Перегрузим операцию [ ], чтобы получить нужный нам результат. Заполним массив данными и выведем их. Хотя для интерфейса класса использованы операции индексации массива, вам следует использовать указатели внутри методов класса.

12. Указатели сложны, поэтому давайте посмотрим, сможем ли мы сделать работу с ними более понятной (или, возможно, более непонятной), используя их симуляцию в классе.

Для разъяснения действия наших доморощенных указателей мы смоделируем память компьютера с помощью массивов. Так как доступ к массивам всем понятен, то вы сможете увидеть, что реально происходит, когда мы используем для доступа к памяти указатели.

Мы будем использовать один массив типа char для хранения всех типов переменных. Именно так устроена память компьютера: массив байтов (тип char имеет тот же размер), каждый из которых имеет адрес (или, в терминах массива, индекс). Однако C++ не позволит нам хранить данные типа float или int в массиве типа char обычным путем (мы можем использовать объединения, но это другая история). Поэтому мы создадим симулятор памяти, используя отдельный массив для каждого типа данных, которые мы хотим сохранить. В этом упражнении мы ограничимся одним типом float, и нам понадобится массив для него. Назовем этот массив fmemory. Однако значения указателей (адреса) тоже хранятся в памяти, и нам понадобится еще один массив для их хранения. Так как в качестве модели адресов мы используем индексы массива, то нам потребуется массив типа int, назовем его pmemory, для хранения этих индексов.

Индекс массива fmemory (назовем его fmem_top) показывает на следующее по очереди доступное место, где можно сохранить значение типа float. У нас есть еще похожий индекс массива pmemory (назовем его pmem_ top). Не волнуйтесь о том, что наша «память» может закончиться. Мы предполагаем, что эти массивы достаточно большие, чтобы хранить все, что мы захотим, и нам не надо заботиться об управлении памятью.

Создадим класс Float, который мы будем использовать для моделирования чисел типа float, которые будет храниться в fmemory вместо настоящей памяти. Класс Float содержит поле, значением которого является индекс массива fmemory, хранящего значения типа float. Назовем это поле addr. В классе также должны быть два метода. Первый — это конструктор, имеющий один аргумент типа float для инициализации значения. Конструктор помещает значение аргумента в элемент массива fmemory, на который указывает указатель fmem_top, а затем записывает значение fmem_top в массив addr. Это похоже на то, как компоновщик и компилятор хранят обычные переменные в памяти. Второй метод является перегружаемой операцией &. Он просто возвращает значение указателя (индекса типа int) в addr.

Создадим второй класс ptrFloat. Объект этого класса содержит адрес (ин¬декс) в pmemory. Метод класса инициализирует этот «указатель» значением типа int. Второй метод перегружает операцию * (операция разыменования). Его действия более сложны. Он

получает адрес из массива pmemory, в котором хранятся адреса. Затем полученный адрес используется как индекс массива fmemory для получения значения типа float, которое располагалось по нужному нам адресу, floats ptrFloat: operator ( )
{
return fmemory [pmemory [addr]];
}

Таким образом мы моделируем действия операции разыменования (*). Заметим, что вам нужно возвращаться из этой функции по ссылке, чтобы можно было использовать операцию * слева от знака равно.

Классы Float и ptrFloat похожи, но класс Float хранит данные типа float в массиве, представляющем собой память, а класс ptrFloat хранит поля типа int (являющиеся у нас указателями, но на самом деле индексами массива) в другом массиве, который тоже представляет собой память.

Это типичное использование этих классов в функции main():

Float varl - 1.234: // определяем и инициализируем
Float var2 = 5.678; // две вещественные переменные
ptrFloat ptrl = &varl: // определяем и инициализируем
ptrFloat ptr2 = &var2; // два указателя
cout « ” *ptrl = " « *ptrl:// получаем значения переменных
cout « " *ptr2 = “ « *ptr2: // и выводим на экран
*ptrl - 7.123: // присваиваем новые значения
*ptr2 = 8.456: // переменным, адресованным через указатели
cout « " *ptrl = " « *ptrl:// снова получаем значения
cout « " *ptr2 = " « *ptr2: // и выводим на экран

Заметим, что за исключением других имен типов переменных, это выглядит так же, как действия с настоящими переменными. Далее результат работы программы:
*ptr1 -1.234 *ptr2 = 5.678

Код:

Двойной щелчок для открытия кода.

```
#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>
#include <string_view>


using namespace std;



//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++5 //works
void addarrays(int* x, int* y, int* z, int size)
{
	for (int i = 0; i < size; i++)
	{
		*(z + i) = *(x + i) + *(y + i);
		cout << "Output is: " << *(z + i) << "\n";
	}
}

void fun5()
```

```
{
	const int size = 3;
	int arr_1[size] = { 1, 2, 3 };
	int arr_2[size] = { 1, 2, 3 };
	int arr_3[size];
	addarrays(arr_1, arr_2, arr_3, size);
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++6 //ne
ponyatno //works
int compstr(const char* s1, const char* s2)
{
	while (*s1 && *s2)
	{
		if (*s1 > *s2)
			return 1;
		else if (*s1 < *s2)
			return -1;
		s1++; s2++;
	}
	if (!*s1 && !*s2)
		return 0;
	else if (!*s1)
		return -1;
	else
		return 1;
}

void fun6() {
	const char* str1{ "aaa" }, * str2{ "bbb" };
	const char* str3{ "bbb" }, * str4{ "bbb" };
	const char* str5{ "ccc" }, * str6{ "bbb" };
	cout << compstr(str1, str2);
	cout << compstr(str3, str4);
	cout << compstr(str5, str6);
}


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++7 //works
class person {
protected:
	string name;
	float salary;
public:
	void setData()
	{
		cout << "Enter name: ";
		cin >> name;
		cout << "Enter salary: ";
		cin >> salary;
	}
	void printData() const
	{
		cout << endl << "Name: " << name;
		cout << endl << "Salary: " << salary;
	}
	float getSalary() const
	{
		return salary;
	}
};
void salsort(person** pp, int n)
{
	for (int j = 0; j < n - 1; j++)
		for (int k = j + 1; k < n; k++)
			if ((*(pp + j))->getSalary() > (*(pp + k))->getSalary())
```

```
                        swap(*(pp + j), *(pp + k));
}

void fun7()
{
	person* people[100];
	int n = 0;
	char choice;
	do {
		people[n] = new person;
		cout << "Person: " << n << endl;
		people[n]->setData(); // a->b is (*a).b
		n++;
		cout << "Continue (y/n)? ";
		cin >> choice;
	} while (choice == 'y');
	cout << "\nUnsorted list";
	for (int j = 0; j < n; j++) {
		cout << endl << "Person: " << j;
		(*(people + j))->printData();
	}
	cout << endl;
	salsort(people, n);
	cout << "\nSorted:";
	for (int j = 0; j < n; j++) {
		cout << endl << "Person: " << j;
		(*(people + j))->printData();
	}
	cout << endl;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++8 //minimal
changes //works
struct link {
	int data;
	link* next;
};
class linklist {
private:
	link* first;
public:
	linklist() {
		first = NULL;
	}
	void additem(int d);
	void display();
};
void linklist::additem(int d) {
	link* newlink = new link;
	newlink->data = d;
	newlink->next = NULL;
	if (first) {
		link* current = first;
		while (current->next)
			current = current->next;
		current->next = newlink;
	}
	else
		first = newlink;
}
void linklist::display()
{
	link* current = first;
	while (current) {
		cout << current->data << "\n";
```

```
			current = current->next;
		}
}

void fun8()
{
		linklist getInfo;
		getInfo.additem(25);
		getInfo.additem(36);
		getInfo.additem(49);
		getInfo.additem(64);
		getInfo.display();
}


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++9 //works


void fun9()
{
		const int SIZE = 10;
		string a0[SIZE];
		string a1[SIZE];
		string a2[SIZE];
		string a3[SIZE];
		string a4[SIZE];
		string a5[SIZE];
		string a6[SIZE];
		string a7[SIZE];
		string a8[SIZE];
		string a9[SIZE];
		string* arr[] = { a0, a1, a2, a3, a4, a5, a6, a7, a8, a9 };
		for (int i = 0; i < SIZE; i++)
			for (int j = 0; j < SIZE; j++)
			{
				int buf = (i * SIZE + j) * 10;
				arr[i][j] = to_string(buf);
			}

		for (int i = 0; i < SIZE; i++)
			for (int j = 0; j < SIZE; j++)
				cout << arr[i][j] << "\n";
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++10 //works

void fun10()
{
		const int SIZE = 10;
		const int NUMBER = 10;

		string* arr[NUMBER];
		for (int i = 0; i < NUMBER; i++)
			*(arr + i) = new string[SIZE];

		for (int i = 0; i < NUMBER; i++) //initialization
			for (int j = 0; j < SIZE; j++)
			{
				int buf = (i * SIZE + j) * 10;
				arr[i][j] = to_string(buf);
			}

		for (int i = 0; i < NUMBER; i++) //output
			for (int j = 0; j < SIZE; j++)
				cout << arr[i][j] << "\n";
```

```
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++11 //ya
zaputalsya
class arrays {
private:
	const int NUMARRAYS;
	const int SIZE;
	int** arr;
public:
	arrays(int x, int y) : NUMARRAYS(x), SIZE(y) {
		arr = new int* [NUMARRAYS];
		for (int i = 0; i < NUMARRAYS; i++)
			*(arr + i) = new int[SIZE];
	}
	int& operator[] (int n) const {
		int j = n / SIZE;
		int k = n % SIZE;
		return *(*(arr + j) + k);
	}
};

int askValue() {
	cout << "Enter value: ";
	int input;
	cin >> input;
	return input;
}


void fun11()
{
	int value1 = askValue();
	int value2 = askValue();
	arrays arr(value1, value2);
	for (int i = 0; i < value1 * value2; i++)
	{
		arr[i] = i * 10;
		if (arr[i] % 100 == 0)
		{
			cout << "\n";
		}
		cout << arr[i] << setw(5);
	}

}


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++12 //niche
osobo ne vydumyval //wrode works
const int fm_SIZE = 100;
float fmemory[fm_SIZE];
int fmem_top = 0; //this is index
const int pm_SIZE = 100;
int pmemory[pm_SIZE];
int pmem_top = 0;

class Float {
private:
	int addr;
public:
	Float(float value) {
		fmemory[fmem_top] = value;
		addr = fmem_top;
		fmem_top++;
```

```
	}
	int operator & () const {
		return addr;
	}
};
class ptrFloat
{
private:
	int addr;
public:
	ptrFloat(int value) {
		pmemory[pmem_top] = value;
		addr = pmem_top;
		pmem_top++;
	}
	float& operator * () const {
		return fmemory[pmemory[addr]];
	}
};

void fun12()
{
	Float var1 = 1.234;
	Float var2 = 5.678;
	ptrFloat ptr2 = &var1;
	ptrFloat ptr1 = &var2;
	cout << " * ptr1 = " << *ptr1 << endl;
	cout << "\n * ptr2 = " << *ptr2 << endl;
	*ptr1 = 7.123;
	*ptr2 = 8.456;
	cout << *ptr1 << "\n";
	cout << *ptr2 << "\n";

}

int main()
{
	setlocale(LC_ALL, "Russian");
	int nomer;
	cout << "\nВведите номер задачи\n";
	cin >> nomer;
	switch (nomer)
	{
	case 5:
		fun5();
		break;
	case 6:
		fun6();
		break;
	case 7:
		fun7();
		break;
	case 8:
		fun8();
		break;
	case 9:
		fun9();
		break;
	case 10:
		fun10();
		break;
	case 11:
		fun11();
		break;
	case 12:
```

```
                fun12();
                break;
        }

        return 0;
}
```