ИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ
КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СТРОИТЕЛЬСТВА, ТРАНСПОРТА
И АРХИТЕКТУРЫ ИМ.Н.ИСАНОВА

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С++

Методическое пособие
для бакалавров направлений:
710400 – Программная инженерия
710300 – Прикладная информатика

Бишкек-2019

Объектно-ориентированное программирование на языке С++: Методическое пособие для бакалавров направления 710400 – Программная инженерия/Кырг.Гос.Унив-т строит-ва, транста и архит-ы им.Н.Исанова; Сост.: С.А.Мукамбетова, Э.Б. Шеримбекова, А.Ж. Алтымышева-Бишкек, 2019.-55стр.

Табл. 2. Библиогр.7 назв.Рис.8

В методическом пособии кратко рассматривается объектно-ориентированный подход к программированию на примере языка программирования С++.

В методическом пособии даны восемь лабораторных работ. В каждой работе указываются цель и краткая теория изучаемых вопросов, порядок выполняемых работ, а также контрольные вопросы. В первых двух лабораторных работах излагаются основные понятия и описываются возможности языка С++. Остальные работы посвящены ООП подход к программированию и ее концепции.

Методическое пособие предназначено для студентов второго курса по направлению 710400 - Программная инженерия, а также может быть рекомендовано студентам первого курса по направлению 710300 – Прикладная информатика при изучении языка программирования С++, читаемого по аналогичной программе.

Рецензент: к.ф.-м.н., ст.преп. каф.КЛиМК ИНИТ КГУСТА
Абыкеев К.А.

СОДЕРЖАНИЕ

Введение

Объектно – ориентированное программирование (ООП) предлагает способ решение проблемы сложности программ. Вместо того, чтобы рассматривать программу как набор последовательно выполняемых инструкций, в ООП программа представляется в виде совокупности объектов, обладающих сходными свойствами и набором действий, которые можно с ними производить. ООП содержит несколько концепций. К этим концепциям относятся объекты и классы, наследование и полиморфизм, составляющие основу ООП подхода программирования.

Целью методических указаний является обучение студентов составлению программы на основе ООП подход при решении инженерных задач на языке С++, а также методике работы при разработке программы.

В методическом руководстве даны девять лабораторных работ. В каждой работе указываются цель и краткая теория изучаемых вопросов, порядок выполняемых работ, а также контрольные вопросы. В первых двух лабораторных работах излагаются основные понятия и описываются возможности языка С++. Остальные работы посвящены ООП подход к программированию и ее концепции.

В ходе подготовки студента к лабораторным занятиям необходимо ознакомиться с теорией и методикой выполнения лабораторных работ по данным методических указаний.

Лабораторная работа №1.

Операторы языка С++. Массивы, функция и указатели.

Цель занятия

1. Приобретение навыков программирования с операторами языка С++ и работа массивами.
2. Изучение функций, указателей и приобретение навыков использования их при программировании.

Постановка задачи

1. Для заданного варианта разработать алгоритм решения задачи на ПК.
2. Текст программы и результаты ее выполнения.

Методические указания

1.1. Программирование разветвленных алгоритмов

Условный оператор является средством реализации разветвления вычислительного процесса. Структура условного оператора имеет следующий вид:

IF <условие> <оператор1> ELSE <оператор2>.

Условный оператор работает по следующему алгоритму. Вначале вычисляется условное выражение <условие>. Если результат есть TRUE (ИСТИНА), то выполняется <оператор1> , а <оператор2> не выполняется; если результат есть FALSE (ЛОЖЬ), наоборот, <оператор1> пропускается, а выполняется <оператор2>. Если при этом часть условного оператора, начиная со слова ELSE, отсутствует, то немедленно управление передается оператору, следующему за условным.

Составить программу, реализующую следующий алгоритм. Ввести значение x и вычислить значение функции

$$f(x)=\begin{cases}5x^2+6, & \text{если } -2<x<5;\\ x^3+7, & \text{если } x\ge 5.\end{cases}$$

Листинг программы:

```
#include "stdafx.h"

#include<iostream>

#include<math.h>

using namespace std;

int main()

{

	setlocale(LC_ALL, "rus");

		int x;

		float y;

		cout << "Введите x\n";  cin >> x;

		if (x<5 && x>-2) y = 5 * x*x + 6;

		else if (x >= 5)  y = x*x*x + 7;
```

```
        cout << "Значение y=" << y;  cout << endl;
        system("pause");
        return 0;
}
```

1.2. Программирование циклических структур алгоритмов

На языке С++ в циклических структурах имеются три различных оператора: **for, while, do.**

Цикл **for**

Цикл **for** организует выполнение программы фиксированное число раз, которое заранее известно. Оператор цикла с параметром **for** имеет такую структуру: **for(i=0; i<15; i++).** Первое называют инициализирующим, второе – условием проверки, а третье – инкрементирующим. Эти три выражения, как правило, содержат одну переменную, которую называют счетчиком цикла. Она определяется до того, как начнет исполняться тело цикла.

Под телом цикла понимается та часть кода, которая периодически исполняется в цикле.

Цикл **while.**

Если нам заранее не известно, сколько раз понадобится выполнить цикл, то используем оператор цикла **while**, его вид: **while<условие**>.

При выполнении оператора while вначале вычисляется и проверяется <*условие*>. Если выражение <*условие*> имеет значение **TRUE**, то выполняется <*оператор*>; после чего вычисление выражения <*условие*> и его проверка повторяется. Если <*условие*> имеет значение **FALSE**, то оператор **while** прекращает свою работу.

Цикл **do**.

В цикле **while** условие продолжения выполнения цикла помещалось в начало цикла. Это означало, что в случае невыполнения условия при первой проверке тело цикла вообще не исполнялось. В некоторых случаях это целесообразно, но возможны и ситуации, когда необходимо выполнить тело цикла хотя бы один раз вне зависимости от истинности проверяемого условия. Для этого следует использовать цикл **do**, в котором условие продолжения цикла располагается не перед, а после тела цикла. Вид

do {тело цикла}, while <условие>.

Оператор <*тело цикла*> выполняется хотя бы один раз, после чего вычисляется выражение <*условие*>; если его значение есть **FALSE**, операторы <*тело цикла*> повторяются, в противном случае оператор **do** завершает свою работу.

Составить программу вычисления суммы с заданной точностью 10-3.

Считать, что требуемая точность достигнута, если очередное слагаемое по модулю будет меньше, чем E. Вычислить $y = \sum_{k=1}^{\infty} \frac{x}{k^3 + k\sqrt{|x|+1}}$.

Листинг программы:

```
#include<iostream>
```

```
#include<math.h>
using namespace std;
int main()
{
	setlocale(LC_ALL, "rus");
	int k;
	float y, s, x;
	const float e = 0.001;
	cout << "Введите x\n";
	cin >> x;  s = 0; k = 1; y = 0.1;  while (y>e)
	{
		y = x / (k*k*k + k*sqrt(abs(x) + 1));
		s = s + y;   k++;
	}
	cout << "Значение y=" << y << endl;
	cout << "Значение s=" << s << endl;
	cout << "Значение k=" << k << endl;
	system("pause");
	return 0;
}
```

Программирование структур данных типа массив

Массив – это структура данных, состоящая из фиксированного числа компонентов одного типа. К компонентам массива обеспечен доступ при помощи указания индексов компонентов. Описание типа массива задается следующим образом: тип данных массива, имя и размер массива.

int age[4]. Здесь **int** - тип, **age** - имя, в квадратных скобках - размер массива или элементы массива. В данном примере массив имеет 4 элемента.

Инициализация массива. Когда массив определен, мы можем присвоить его элементам значения. В примере 12 элементам массива присваивается количество дней для каждого месяца.

```
#include "stdafx.h"

#include<iostream>

using namespace std;

int main()

{

	setlocale(LC_ALL, "rus");

		int month, day, total_days;

	int days_per_month[12] = { 31, 28, 31.30, 31, 30, 31, 31, 30, 31, 30, 31 };

	cout << "\n Введите месяц(от 1 до 12) :"; cin >> month;

	cout << "\n Введите день(от 1 до 31) :"; cin >> day;

	total_days = day;

	for (int i = 0; i<month-1; i++) total_days += days_per_month[i];

	cout << " Общее число дней с начала года : " << total_days << endl;
```

```
    system("pause");

    return 0;

}
```

Результат:

Введите месяц (от 1 до 12): 3

Введите день (от 1 до 31):11

Общее число дней с начала года: 70

Аналогично описание двумерного массива **int age[4][4],** который требует двух индексов.

Ниже приводится фрагмент программ для ввода и вывода двумерных массивов:

```
#include<iostream>
#include<conio>
#include<math.h>
  int main()
{  int matr[100][100],k,i,j,n;  cout<<"Введите n=";  cin>>n;  cout<<endl;  k=1;
  for(i=1;i<=n;i++)
{ for(j=1;j<=n;j++)
{ matr[i][j]=k; k++;  cout<<matr[i][j];
}  cout<<endl;}  cout<<endl;  k=1;  for(i=1;i<=n;i++)
{  for(j=1;j<=n;j++) {  matr[i][j]=k; k++;  cout<<matr[i][j];}  cout<<endl;}
getch();
return 0;}
```

Составить программу для поворота массива на 900.

```
#include<iostream>
#include<conio>
#include<math.h>
int main()
{
 int matr[100][100],  mat[100][100],k,i,j,n;  cout<<"Введите  n=";  cin>>n;
  cout<<endl;  k=1;  for(i=1;i<=n;i++)
{ for(j=1;j<=n;j++)
{ matr[i][j]=k; k++;  cout<<matr[i][j];}  cout<<endl;}  cout<<endl;
  k=1;  for(i=1;i<=n;i++) {  for(j=1;j<=n;j++) {  k++;  mat[i][j]=matr[n+1-j][i];
  cout<<mat[i][j];}  cout<<endl;
}
 return 0;}
```

1.3. Функция

Функция представляет собой основу, на которой строится любая программа C++. Каждая программа обязательно должна включать главную функцию с именем main(). Функция представляет собой именованное объединение группы операторов. Это объединение может быть вызвано из других частей программы.

Определение функции. Определение функции, в котором выделяются две части – заголовок и тело, имеет следующий формат:

Тип_функции, имя_функции (*формальных_параметров*)

Тело_функции.

Здесь тип_функции – тип возвращаемого функцией значения, в том числе **void**, если функция никакого значения не возвращает. Имя_функции –идентификатор. Тело_функции – это всегда блок или составной оператор, т.е. последовательность описаний и операторов, заключенная в фигурные скобки. Очень важным оператором тела функции является оператор возврата в точку вызова: **return** выражения или **return**.

Выражение в операторе **return** определяет возвращаемое функцией значение. Именно это значение будет результатом обращения к функции. Тип возвращаемого значения определяется типом функции. Если функция не возвращает никакого значения, то выражение в операторе **return** опускается.

Вызов функции – это имя функции и круглые скобки со список фактических параметров.

```
# include<iostream>
double Func () { тело функции     }
int main ()    {Func ()     ----------     return 0;          }
```

Передача аргументов в функцию. Аргументы позволяют функции оперировать различными значениями или выполнять различные действия в зависимости от переданных ей значений.

Пример -

```
# include<iostream>
void repchar (char ch,int n)
{
for (int i=0; i<n; i++) cout<<ch;
cout<<endl;
}
int main()
{
 char chin;
 int nin;
 cout<<”Введите символ:”;
  cin>>chin;
  cout<<”Введите число повторений символа:”;
  cin>>nin;
  repchar (chin, nin);
  return 0;
  }
```

Результат:

```
Введите символ: *
Введите число повторений символов: 20
**********
```

В этой программе переменные *chin* и *nin* функции **main ()** передаются в качестве аргументов в функцию **repchar ():**

```
repchar (char ch,int n); // вызов функции.
```

Рекурсия функции. Существование функций делает возможным использование такого средства программирования, как рекурсия. Рекурсия позволяет функции вызывать саму себя на выполнение. Рассмотрим функцию вычисления факториала целого числа.

```
int fact (int n)
{if (n==1) return 1;
else return n*fact (n-1); // вызов самой себя}
int main()
{
 setlocale(LC_ALL, "RUSSIAN");
 int n;
 cout << "Введите символ : ";
 cin >> n;
 fact(n);
```

```
system("pause");
return fact(n);
}
```
Функция *fact* вызывает сама себя с модифицированными аргументами.

1.4. Указатели

Адреса и указатели

Каждая переменная в программе – это объект, имеющий имя и значение. По имени можно обратиться к переменной и получить ее значение. С точки зрения машинной реализации имя переменной соответствует адресу того участка памяти, который для нее выделен, а значение переменной - содержимому этого участка памяти. Загружаясь в память, наша программа занимает некоторое количество этих адресов. Это означает, что каждая переменная и каждая функция нашей программы начинается с какого-либо конкретного адреса. Указатели позволяют получить значения по адресу. Чтобы получить адрес в явном виде, применяют унарную операцию &. Адреса имеют целочисленные беззнаковые значения, и их можно обрабатывать как целочисленные величины. Имея возможность с помощью операции & определять адрес переменной или другого объекта программы, нужно уметь его сохранять, преобразовывать и передавать. Для этих целей в языке С++ введены переменные типа указатель. Как и всякие переменные, указатели нужно определять и описывать, для этих целей используется разделитель “*”. Кроме разделителя, в определениях и описаниях указателей задается тип объектов, на которые ссылаются указатели. Тогда указатель объявляется следующим образом: тип * <имя переменной>.

Важность указателей становится очевидной в том случае, когда мы не имеем прямого доступа к переменной.

Операции над указателями

В языке С++ допустимы следующие операции над указателями: присваивание, получение адреса самого указателя, унарные операции изменения значения указателя, арифметические операции и операции сравнений.

Операция присваивания предполагает, что слева от знака операции присваивания помещено имя указателя, справа – адрес любого объекта того же типа, что и указатель слева, либо уже имеющий значение или константа NULL, определяющая условное нулевое значение указателя.

Примеры

char *z; int *k, *i; int x=10; i=&x; k=i; z= NULL;.

Соотношение между именем, адресом и значением указателя показано на рисунке 4.

Рисунок 4

Унарные операции “++” и “--”. Числовые значения переменных типа указатель меняются по- разному в зависимости от типа данных, с которыми связаны эти переменные. Если указатель связан с типом char, то при выполнении операции “++” и “--” его числовое значение изменяется на 1(на 1 байт), если **int** – на 2, если **float**–3.

Аддитивные операции. Две переменные типа указатель нельзя суммировать, однако к указателю можно прибавить целую величину. При этом вычисляемое значение зависит от типа объекта, с которым связан указатель. Например, пусть указатель Р имеет значение 2000 и указывает на целое (**int**). Тогда в результате выполнения оператора P=P+3; значение указателя Р будет 2006. Общая формула P=P+ n*(количество байт памяти базового типа указателя).

Операция вычитание. В отличие от операции сложения, операция вычитания применима не только к указателю и целой величине, но и к двум указателям на объекты одного типа. С ее помощью можно находить разность двух указателей и тем самым определить “расстояние” между размещением в памяти двух объектов. При этом “расстояние” вычисляется в единицах, кратных “длине” отдельного элемента данных того типа, к которому отнесен указатель. Например, int x[5], *i, *k, j;int i=&x[0]; k=&x[4]; j=k-i; j принимает значение 4, а не 8, как можно было предположить, исходя из того, что каждый элемент массива x[ ] занимает два байта.

Другие арифметические операции над указателями запрещены, например, нельзя умножить указатель на число и т.д.

Указатели можно сравнивать. Применимы все 6 операций: <,>, <=, >=, =, = = и !=.

Пример – Вывод значения по адресу.

```
# include <iostream>
# include <conio>
void main ()
{
int var1=11;
int *ptr; ptr=& var1; // помещаем в ptr адрес переменной var1
cout<< *ptr;
}
```

Результат 11.

Звездочка, стоящая перед именем переменной, как в выражении ***ptr**, называется операцией разыменования. Она позволяет получить значения переменной, хранящейся по адресу. Таким образом, выражении ***ptr** представляет собой значение переменной, на которую указывает указатель **ptr.**

Указатель на **void**. Адрес, который помещается в указатель, должен быть одного с ним типа. Нельзя присвоить указателю на **int** адрес переменной **float.**

float var1 = 11; int *ptrint=&var1// так нельзя

Однако есть одно исключение. Это указатель на **void** и определяется следующим образом: void*ptr; f loat var1 = 11; void *ptrint=&var1// так можно.

Указатели и массивы

В языке С++ существует связь между указателями и массивами, так как имя массива – это адрес памяти, начиная с которого расположен массив, т.е. адрес первого элемента массива. Таким образом, если был объявлен массив **int mas[5];** то mas является указателем на массив, точнее, на первый элемент массива. Для того чтобы получить значения 3-го элемента массива mas, можно написать **mas[2]** или ***(mas+2).** Результат будет один и тот же. Рассмотрим на примерах указатели и массивы.

```
//Обычный доступ к элементам массива
# include <iostream>
# include <conio>
void main ()
{
int mas[5]={31, 64, 77, 52, 93 };
for (int i=0; i<5; i++)
cout<< mas[i];
}
```

Результат: 31 54 77 52 93.

Доступ к элементам массива через указатель

```
# include <iostream>
# include <conio>
void main ()
{
int mas[5]={31, 64, 77, 52, 93
 };
```

```
for (int i=0; i<5; i++)};
cout<< *(mas+i));
}
```

Результат: 31 54 77 52 93.

Указатели – константы и указатели – переменные. Можем ли мы записать ***(mas++),** т.е. вместо прибавления шага к **i** использовать операцию увеличения. Сделать так мы не можем, поскольку имя массива- это адрес памяти и является константой, поэтому это указатель константы. Однако мы можем увеличить указатель, который содержит этот адрес. Покажем на примере.

```
# include <iostream>
# include <conio>
void main ()
{int mas[5]={31, 64, 77, 52, 93 };
int *ptrint;
ptrint= mas;
for (int i=0; i<5; i++)
cout<<*(ptrint++); }
```

Результат: 31 54 77 52 93.

Здесь мы определили указатель на **int** – **ptrint** – и затем присвоили значение адреса массива mas. Теперь мы можем получить доступ к элементам массива, используя выражение ***(ptrint++).**

Варианты заданий.

1. Два выпуклых многоугольника заданы на плоскости перечислением координат вершин в порядке обхода границы. Определить площади многоугольников и проверить, лежит ли один из них строго внутри другого.
2. Из заданного на плоскости множества точек выбрать три различные точки так, чтобы разность между площадью круга, ограниченного окружностью, проходя щей через эти три точки, и площадью треугольника с вершинами в этих точках была минимальной.
3. Даны два множества точек на плоскости. Выбрать три различные точки первого множества так, чтобы круг, ограниченный окружностью, проходящей через эти три точки, содержал все точки второго множества и имел минимальную площадь.
4. Даны два множества точек на плоскости. Выбрать четыре различные точки первого множества так, чтобы квадрат с вершинами в этих точках накрывал все точки второго множества и имел минимальную площадь.
5. Даны два множества точек на плоскости. Выбрать три различные точки первого множества так, чтобы треугольник с вершинами в этих точках накрывал все точки второго множества и имел минимальную площадь.
6. Даны два множества точек на плоскости. Найти радиус и центр окружности, проходящей через n (n>=3) точек первого множества и содержащей строго внутри себя равное число точек первого и второго множеств.
7. Даны два множества точек на плоскости. Из первого множества выбрать три различные точки так, чтобы треугольник с вершинами в этих точках содержал (строго внутри себя) равное количество точек первого и второго множеств.
8. На плоскости заданы множество точек М и круг. Выбрать из М две различные точки так, чтобы наименьшим образом различалр1сь количества точек в круге, лежащие по разные стороны от прямой, проходящей через эти точки.
9. Дано 3n точек на плоскости, причем никакие три из них не лежат на одной прямой. Построить множество n треугольников с вершинами в этих точках так, чтобы никакие два треугольника не пересекались и не содержали друг друга.
10. Выбрать три различные точки из заданного множества точек на плоскости так, чтобы была минимальной разность между количествами точек, лежащих внутри и вне треугольника с вершинами в выбранных точках.

11. Определить радиус и центр окружности, проходящей по крайней мере через три различные точки заданного множества точек на плоскости и содержащей внутри наибольшее количество точек этого множества.
12. На плоскости заданы множество точек A и точка d вне его. Подсчитать количество различных неупорядоченных троек точек a,b,c,из A таких, что четырехугольник abcd является параллелограммом.
13. На плоскости заданы множество точек A и множество окружностей B, Найти две такие различные точки из A, что проходящая через них прямая пересекается с максимальным количеством окружностей из B.
14. Задано множество точек на плоскости. Найти все четверки точек, являющихся вершинами квадратов. Найти квадрат, внутри которого лежит наибольшее количество точек множества.
15. Определить радиус и центр окружности минимального радиуса, проходящей хотя бы через три различные точки заданного множества точек на плоскости.
16. Найти три треугольника с вершинами в заданном множестве точек па плоскости так, чтобы второй треугольник лежал строго внутри первого, а третий внутри второго.
17. Дано множество точек на плоскости. Построить все возможные треугольники с вершинами в этом множестве точек и найти среди них такой, стороны которого пересекаются с максимальным количеством треугольников.
18. На плоскости заданы множество точек и окружность радиусом R с центром в начале координат. Построить множество всех треугольников с вершинами в заданных точках, все три стороны которых пересекаются с окружностью, и найти среди них треугольник с минимальной площадью.

Лабораторная работа №2

Работа со структурами.

Цель занятия

1. Приобретение навыков формирования и преобразования структурных типов данных.
2. Знакомство с объектно-ориентрованным программированием.

Содержание отчета

1. Постановка задачи для конкретного варианта.
2. Текст программы и результаты ее выполнения.

Методические указания

Структура является объединением разнородных данных. Переменные , входящие в состав структуры, называются полями структуры. Структуры являются одной из составляющих главных концепций языка – объектов и классов. На практике отличие структуры от класса заключается в следующем: структура, как правило, используется в качестве объединения данных, а классы – в качестве объединения данных и функций. Таким образом, изучая структуры, мы тем самым закладываем основы для понимания классов и объектов.

Структуры – это составные типы данных, построенные с использованием других типов. Определение структуры. Определение структуры начинается с ключевого слова **struct**. Затем следует имя структуры, объявления полей, заключенные в фигурные скобки. После закрывающей фигурной скобки следует точка с запятой (**;**) – символ, обозначающий конец определения структуры.

```
#include "stdafx.h"
#include<iostream>
using namespace std;
struct part
{
        int modelnumber;
```

```
        int partnumber;
        float cost;
};

int main()
{
        setlocale(LC_ALL, "rus");
        {part part1 = { 6244, 373, 217.55 };// Инициализация переменной
        cout << "модель" << part1.modelnumber;
        cout << "деталь" << part1.partnumber;
        cout << " стоимость $" << part1.cost << endl; }
    system("pause");
return 0;
}
```

Результат: Модель 6244, деталь 373, стоимость $ 217.55

Описание структуры выглядит следующим образом:

```
struct List
{
int number;
char name[10];
char surname[10];
int age;
};
```

Ключевое слово struct начинает определение структуры. Идентификатор List – имя структуры. Имя структуры используется при объявлении переменных структур данного типа. В этом примере имя нового типа – List. Имена,объявленные в фигурных скобках описания структуры – это элементы структуры. Элементы одной и той же структуры должны иметь уникальные имена, но две разные структуры могут содержать не конфликтующие элементы с одинаковыми именами. Каждое определение структуры должно заканчиваться точкой с запятой.

Определение List содержит четыре элемента: строки name и surname из 10 символов каждая и целочисленные переменные number и age типа **int**. Элементы структуры могут быть любого типа, и одна структура может содержать элементы многих разных типов.

Структура не может, однако, содержать экземпляры самой себя. Например, элемент типа List не может быть объявлен в определении структуры List.

Определение структуры данных не резервирует никакого пространства в памяти; определение только создает новый тип данных, который используется для объявления переменных. Переменные структуры объявляются так же, как переменные других типов. Объявление

List Object, listArray[10], *listPtr;

объявляет Object переменной типа List, listArray – массивом с 10 элементами типа Man, а listPtr – указателем на объект типа List.

Доступ к элементам структуры

Для доступа к элементам структуры используются операции доступа к элементам – *операция точка* (**.**) и *операция стрелка* (->). Операция точка обращается к элементу структуры по имени переменной объекта или по ссылке на объект. Например, чтобы напечатать элемент name структуры Object используется оператор

cout << Object.name;

Операция стрелка, состоящая из знака минус (-) и знака больше (>), записанных без пробела, обеспечивает доступ к элементу структуры через указатель на объект. Допустим, что указатель listPtr был уже объявлен как указывающий на объект типа List и что адрес структуры Object был уже присвоен listPtr, при помощи оператора listPtr=&Object. Тогда, чтобы напечатать элемент

name структуры Object с указателем listPtr, можно использовать оператор

cout << listPtr ->name;

Выражение listPtr ->name; эквивалентно (*listPtr).name, которое разыменовывает указатель и делает доступным элемент name через операцию точка. Скобки нужны здесь потому, что операция точка имеет более высокий приоритет, чем операция разыменования указателя (*). Операции стрелка и точка наряду с круглыми и квадратными скобками имеют второй наивысший приоритет и ассоциативность слева направо.

Пример программы.

Создать анкету для 5 человек, содержащую имя, фамилию и возраст человека. Вывести на экран анкетные данные тех членов списка, возраст которых больше 18, но не превышает 27 лет.

```
#include<iostream>
using namespace std;
struct List
{
int number;
char name[10];
char surname[10];
int age;
};
void vvod (List &, int);
void poisk (List &);
int main( )
{
setlocale(LC_ALL, "Russian");
List anketa[5]; int i;
for (i=0; i<5; i++)
vvod(anketa[i], i+1);
for (i=0; i<5; i++)
poisk(anketa[i]);
system("Pause");
return 0;
}
void vvod (List &strc, int a)
{
strc.number=a;
cout << "Номер - "<<strc.number <<endl;
cout << "Ввести имя - ";
cin >> strc.name;
cout << "Ввести фамилию- ";
cin >> strc.surname;
cout << "Ввести возраст - ";
cin >> strc.age;
cout << endl;
}
void poisk (List &strc)
{
if ((strc.age>=18)&&(strc.age<=27))
{
cout << "№ " << strc.number << endl;
cout << "Имя - " << strc.name << endl;
cout << "Фамилия - " << strc.surname << endl;
```

```
cout << "Возраст - " << strc.age << endl;
}
}
```

Варианты заданий

1. Создать структуру для работы с рациональными дробями (вида m/n). Данные структуры: числитель, знаменатель, десятичный вариант дроби. Создать функции: ввода числителя и знаменателя, вывода дроби рациональном и десятичном вариантах с точность до 3 знака после запятой.
2. Создать структуру для работы с арифметическими комплексными числами. Комплексное число в арифметической форме задается своей вещественной и мнимой частями ($a+b\cdot i$, где i – мнимая единица$-\sqrt{-1}$). Данные структуры: вещественная и мнимая часть. Создать функции: ввод числа (a и b), вывод числа на экран в форме a+b×i, вывод на экран модуля комплексного числа ($\sqrt{a^2+b^2}$), вывод на экран аргумента комплексного числа (atan $\frac{a}{b}$).
3. Создать структуру для работы с тригонометрическими комплексными числами. Комплексное число в тригонометрической форме задается своим модулем ***r*** и аргументом (углом) φ в виде r×[cos φ+i×sin φ].Данные структуры: модуль ***r*** и аргумент φ. Создать функции: ввод числа (r и φ), вывод числа на экран в форме r×[cos φ+i×sin φ], возведение комплексного числа в степень. При возведении комплексного числа в целую степень, модуль возводится в ту же степень, а аргумент умножается на показатель степени (формула Муавра): r^n(cos(n· φ)+ i · sin(n· φ)).
4. Создать структуру, описывающую гармонический сигнал в виде s[n]=A·sin(ω·n·Δt+φ0). Данные структуры: массив s[n], амплитуда (A), частота (ω), начальная фаза (φ0), интервал дискретизации (Δt). Создать функции: ввод начальных значений и заполнение массива s[n]. Вывод на экран массива s[n].
5. Создать структуру, реализующую стек целых чисел типа LIFO. Данные структуры: массив стека, указатель на вершину стека. Создать функции: запись в вершину стека, чтение вершины стека.
6. Создать структуру, описывающую успеваемость студента. Данные структуры: фамилия, имя, оценки по трем предметам. Создать функции: ввод данных, вывод данных, вычисление среднего балла студента.
7. Создать структуру, реализующую очередь целых чисел типа FIFO. Данные структуры: массив очереди, указатели на начало очереди и на конец очереди. Создать функции: запись в начало очереди, чтение конца очереди.
8. Создать структуру, описывающую библиотечную карточку. Данные структуры: имя, фамилия, количество взятых книг. Создать три функции: ввод данных, увеличение или уменьшение количества взятых книг.
9. Создать структуру, описывающую багаж пассажира. Данные структуры: количество вещей и общий вес вещей. Создать две функции: ввод количества вещей и веса каждой вещи в отдельности, вывод на экран общего веса багажа.
10. Создать структуру, описывающую студенческую группу. Данные структуры: наименование группы, количество студентов, количество мужчин и женщин. создать функции: ввод данных, вывод данных, вычисление процента состава мужчин и женщин в группе.
11. Создать структуру для работы с массивом. Данные структуры: массив M(10). Создать функции: ввод массива, вывод массива, определение длины массива. Длина массива определяется как квадратный корень из суммы квадратов элементов массива.
12. Создать структуру для работы с матрицей. Данные структуры: матрица A(3×3). Создать функции: ввод матрицы, вывод матрицы в квадратной форме, вывод на экран транспонированной матрицы.

13. Создать структуру для работы с датой. Данные структуры: день, месяц, год. Создать функции: ввод даты, вывод даты в европейском стандарте (ДД:ММ:ГГ), вывод даты в американском стандарте (ММ:ДД:ГГ).
14. Создать структуру, описывающую почтовую сортировку. Данные структуры: город, улица, дом, квартира, количество посылок, ценность. Создать функции: ввод данных, вывод данных, вычисление суммарного количества посылок и суммарной ценности.
15. Создать структуру, описывающую запись в книге учета постояльцев в гостинице. Данные структуры: Имя и адрес постояльца, цена проживания в сутки, срок проживания. Создать функции: ввод данных, вывод данных, вычисление стоимости проживания в гостинице.
16. Создать структуру для работы с матрицей. Данные структуры: матрица B(3×3). Создать функции: ввод матрицы, вывод матрицы в квадратной форме, вывод на экран нормы матрицы (максимального значения суммы значений столбцов).
17. Создать структуру, описывающую товар. Данные структуры: наименование товара, цена товара, срок гарантии. Создать функции: ввод данных, вывод данных, снижение цены товара на 50%.
18. Создать структуру, описывающую студенческую группу. Данные структуры: количество студентов в группе, количество иногородних студентов, количество иностранных студентов. Создать функции: вывод на экран общего количества студентов, вывод на экран количества иногородних студентов, вывод на экран количества иностранных студентов.

Лабораторная работа № 3

Работа с классами и объектами

Цель занятия

1. Приобретение навыков формирования и преобразования структурных типов данных.
2. Знакомство с объектно-ориентированным программированием.

Постановка задачи

1. Для заданного варианта разработать алгоритм решения задачи на ЭВМ.
2. Составить программу на языке C++, которая работает с любым допустимым набором данных.
3. Входную информацию и результаты обработки вывести на печать, снабдив их соответствующими заголовками.

Содержание отчета

1. Постановка задачи для конкретного варианта.
2. Текст программы и результаты ее выполнения.

Методические указания

Классы в программировании состоят из свойств и методов. Свойства — это любые данные, которыми можно характеризовать объект класса. В нашем случае, объектом класса является студент, а его свойствами — имя, фамилия, оценки и средний балл.

У каждого студента есть имя — name и фамилия last_name . Также, у него есть промежуточные оценки за весь семестр. Эти оценки мы будем записывать в целочисленный массив из пяти элементов. После того, как все пять оценок будут проставлены, определим средний балл успеваемости студента за весь семестр — свойство average_ball.

Методы — это функции, которые могут выполнять какие-либо действия над данными (свойствами) класса. Добавим в наш класс функцию calculate_average_ball(), которая будет определять средний балл успеваемости ученика.

- **Методы** класса — это его функции.
- **Свойства** класса — его переменные.

```
class Students {
   public:
      // Функция, считающая средний балл
      void calculate_average_ball()
      {
         int sum = 0; // Сумма всех оценок
         for (int i = 0; i < 5; ++i) {
            sum += scores[i];
         }
         // считаем среднее арифметическое
         average_ball = sum / 5.0;
      }

      // Имя студента
      std::string name;
      // Фамилия
      std::string last_name;
      // Пять промежуточных оценок студента
      int scores[5];

   private:
      // Итоговая оценка за семестр
      float average_ball;
};
```

Функция calculate_average_ball() просто делит сумму всех промежуточных оценок на их количество.

Модификаторы доступа public и private

Все свойства и методы классов имеют права доступа. По умолчанию, все содержимое класса является доступным для чтения и записи только для него самого. Для того, чтобы разрешить доступ к данным класса извне, используют модификатор доступа public. Все функции и переменные, которые находятся после модификатора public, становятся доступными из всех частей программы.

Закрытые данные класса размещаются после модификатора доступа private. Если отсутствует модификатор public, то все функции и переменные, по умолчанию являются закрытыми (как в первом примере).

Обычно, приватными делают все свойства класса, а публичными — его методы. Все действия с закрытыми свойствами класса реализуются через его методы. Рассмотрим следующий код.

```
class Students {
   public:
      // Установка среднего балла
      void set_average_ball(float ball)
      {
         average_ball = ball;
      }
      // Получение среднего балла
      float get_average_ball()
      {
         return average_ball;
      }
```

```
    std::string name;
    std::string last_name;
    int scores[5];

  private:
    float average_ball;
};
```

Мы не можем напрямую обращаться к закрытым данными класса. Работать с этими данными можно только посредством методов этого класса. В примере выше, мы используем функцию get_average_ball() для получения средней оценки студента, и set_average_ball() для выставления этой оценки.

Функция set_average_ball() принимает средний балл в качестве параметра и присваивает его значение закрытой переменной average_ball. Функция get_average_ball() просто возвращает значение этой переменной.

Программа учета успеваемости студентов

Создадим программу, которая будет заниматься учетом успеваемости студентов в группе. Создайте заголовочный файл **students.h**, в котором будет находиться класс Students.

```
/* students.h */
#include <string>

class Students {
  public:
    // Установка имени студента
    void set_name(std::string student_name)
    {
      name = student_name;
    }
    // Получение имени студента
    std::string get_name()
    {
      return name;
    }
    // Установка фамилии студента
    void set_last_name(std::string student_last_name)
    {
      last_name = student_last_name;
    }
    // Получение фамилии студента
    std::string get_last_name()
    {
      return last_name;
    }
    // Установка промежуточных оценок
    void set_scores(int student_scores[])
    {
      for (int i = 0; i < 5; ++i) {
        scores[i] = student_scores[i];
      }
    }
    // Установка среднего балла
```

```
    void set_average_ball(float ball)
    {
       average_ball = ball;
    }
    // Получение среднего балла
    float get_average_ball()
    {
       return average_ball;
    }

  private:
    // Промежуточные оценки
    int scores[5];
    // Средний балл
    float average_ball;
    // Имя
    std::string name;
    // Фамилия
    std::string last_name;
};
```

Мы добавили в наш класс новые методы, а также сделали приватными все его свойства. Функция set_name() сохраняет имя студента в переменной name, а get_name() возвращает значение этой переменной. Принцип работы функций set_last_name() и get_last_name() аналогичен.

Функция set_scores() принимает массив с промежуточными оценками и сохраняет их в приватную переменную int scores[5].

Теперь создайте файл **main.cpp** со следующим содержимым.

```
/* main.cpp */
#include "stdafx.h"
#include <iostream>
#include "students.h"

int main()
{
        setlocale(LC_ALL, "rus");
        // Создание объекта класса Student
        Students student;

        std::string name;
        std::string last_name;

        // Ввод имени с клавиатуры
        std::cout << "Имя: ";
        getline(std::cin, name);

        // Ввод фамилии
        std::cout << "Фамалия : ";
        getline(std::cin, last_name);

        // Сохранение имени и фамилии в объект класса Students
```

```
	student.set_name(name);
	student.set_last_name(last_name);

	// Оценки
	int scores[5];
	// Сумма всех оценок
	int sum = 0;

	// Ввод промежуточных оценок
	for (int i = 0; i < 5; ++i) {
		std::cout << "Предмет " << i + 1 << ": ";
		std::cin >> scores[i];
		// суммирование
		sum += scores[i];
	}

	// Сохраняем промежуточные оценки в объект класса Student
	student.set_scores(scores);
	// Считаем средний балл
	float average_ball = sum / 5.0;
	// Сохраняем средний балл в объект класса Students
	student.set_average_ball(average_ball);
	// Выводим данные по студенту
	std::cout << "Средний балл" << student.get_name() << " "
		<< student.get_last_name() << " __ "
		<< student.get_average_ball() << std::endl;
	system("pause");
	return 0;
}
```

```
   // Сохраняем промежуточные оценки в объект класса Student
   student.set_scores(scores);
   // Считаем средний балл
   float average_ball = sum / 5.0;
   // Сохраняем средний балл в объект класса Students
   student.set_average_ball(average_ball);
   // Выводим данные по студенту
   std::cout << "Average ball for " << student.get_name() << " "
       << student.get_last_name() << " is "
       << student.get_average_ball() << std::endl;

   return 0;
}
```

В самом начале программы создается объект класса Students. Дело в том, что сам класс является только описанием его объекта. Класс Students является описанием любого из студентов, у которого есть имя, фамилия и возможность получения оценок.

Объект класса Students характеризует конкретного студента. Если мы захотим выставить оценки всем ученикам в группе, то будем создавать новый объект для каждого из них. Использование классов очень хорошо подходит для описания объектов реального мира.

После создания объекта student, мы вводим с клавиатуры фамилию, имя и промежуточные оценки для конкретного ученика. Пускай это будет Артем Иванов, у которого есть пять оценок за семестр — две тройки, две четверки и одна пятерка.

Введенные данные мы передаем **set**-функциям, которые присваивают их закрытым переменным класса. После того, как были введены промежуточные оценки, мы высчитываем средний балл на основе этих оценок, а затем сохраняем это значение в закрытом свойстве average_ball, с помощью функции set_average_ball().

Конструктор и деструктор класса

Конструктор класса — это специальная функция, которая автоматически вызывается сразу после создания объекта этого класса. Он не имеет типа возвращаемого значения и должен называться также, как класс, в котором он находится. По умолчанию, заполним двойками массив с промежуточными оценками студента.

```
class Students {
   public:
      // Конструктор класса Students
      Students(int default_score)
      {
         for (int i = 0; i < 5; ++i) {
            scores[i] = default_score;
         }
      }

   private:
      int scores[5];
};

int main()
{
   // Передаем двойку в конструктор
   Students *student = new Students(2);
   return 0;
}
```

Мы можем исправить двойки, если ученик будет хорошо себя вести, и вовремя сдавать домашние задания. А на «нет» и суда нет :-)

Деструктор класса вызывается при уничтожении объекта. Имя деструктора аналогично имени конструктора, только в начале ставится знак тильды ~. Деструктор не имеет входных параметров.

```
#include <iostream>

class Students {
   public:
      // Деструктор
      ~Students()
      {
         std::cout << "Memory has been cleaned. Good bye." << std::endl;
      }
};
```

```
int main()
{
   Students *student = new Students;
   // Уничтожение объекта
   delete student;
   return 0;
}
```

Варианты заданий

1. Создать класс для работы с рациональными дробями (вида m/n). Закрытые данные класса: числитель, знаменатель, десятичный вариант дроби. Открытые функции класса: ввод числителя и знаменателя, вывод дроби в рациональном и десятичном вариантах с точность до 3 знака после запятой.
2. Создать класс для работы с арифметическими комплексными числами. Комплексное число в арифметической форме задается своей вещественной и мнимой частями ($a+b\cdot i$, где i – мнимая единица $\sqrt{-1}$). Данные класса: вещественная и мнимая часть. Открытые функции: ввод числа (a и b), вывод числа на экран в форме a+b×i, вывод на экран модуля комплексного числа ($\sqrt{a^2+b^2}$), вывод на экран аргумента комплексного числа (atan $\frac{a}{b}$).
3. Создать класс для работы с тригонометрическими комплексными числами. Комплексное число в тригонометрической форме задается своим модулем ***r*** и аргументом (углом) φ в виде r×[cos φ+i×sin φ].Данные класса: модуль ***r*** и аргумент φ. Открытые функции класса: ввод числа (r и φ), вывод числа на экран в форме r×[cos φ+i×sin φ], возведение комплексного числа в степень. При возведении комплексного числа в целую степень, модуль возводится в ту же степень, а аргумент умножается на показатель степени (формула Муавра): r^n (cos(n· φ)+ i · sin(n· φ)).
4. Создать класс, описывающий гармонический сигнал в виде s[n]=A·sin(ω·n·Δt+φ0). Данные класса: массив s[n], амплитуда (A), частота (ω), начальная фаза (φ0), интервал дискретизации (Δt). Открытые функции класса: ввод начальных значений и заполнение массива s[n]. Вывод на экран массива s[n].
5. Создать класс, реализующий стек целых чисел типа LIFO. Данные класса: массив стека, указатель на вершину стека. Открытые функции класса: запись в вершину стека, чтение вершины стека.
6. Создать класс, описывающую успеваемость студента. Данные класса: фамилия, имя, оценки по трем предметам. Открытые функции класса: ввод данных, вывод данных, вычисление среднего балла студента.
7. Создать класс, реализующую очередь целых чисел типа FIFO. Данные класса: массив очереди, указатели на начало очереди и на конец очереди. Открытые функции класса: запись в начало очереди, чтение конца очереди.
8. Создать класс, описывающую библиотечную карточку. Данные класса: имя, фамилия, количество взятых книг Открытые функции класса: ввод данных, увеличение или уменьшение количества взятых книг.
9. Создать класс, описывающий багаж пассажира. Данные класса: количество вещей и общий вес вещей. Открытые функции класса: ввод количества вещей и веса каждой вещи в отдельности, вывод на экран общего веса багажа.
10. Создать класс, описывающую студенческую группу. Данные класса: наименование группы, количество студентов, количество мужчин и женщин. Открытые функции класса: ввод данных, вывод данных, вычисление процента состава мужчин и женщин в группе.
11. Создать класс для работы с массивом. Данные класса: массив M(10).Открытые функции класса: ввод массива, вывод массива, определение длины массива. Длина массива определяется как квадратный корень из суммы квадратов элементов массива.

12. Создать класс для работы с матрицей. Данные класса: матрица A(3×3). Открытые функции класса: ввод матрицы, вывод матрицы в квадратной форме, вывод на экран транспонированной матрицы.
13. Создать класс для работы с датой. Данные класс: день, месяц, год. Открытые функции класса: ввод даты, вывод даты в европейском стандарте (ДД:ММ:ГГ), вывод даты в американском стандарте (ММ:ДД:ГГ).
14. Создать класс, описывающую почтовую сортировку. Данные класса: город, улица, дом, квартира, количество посылок, ценность. Открытые функции класса: ввод данных, вывод данных, вычисление суммарного количества посылок и суммарной ценности.
15. Создать класс, описывающую запись в книге учета постояльцев в гостинице. Данные класса: Имя и адрес постояльца, цена проживания в сутки, срок проживания. Открытые функции класса: ввод данных, вывод данных, вычисление стоимости проживания в гостинице.
16. Создать класс для работы с матрицей. Данные класса: матрица B(3×3). Открытые функции класса: ввод матрицы, вывод матрицы в квадратной форме, вывод на экран нормы матрицы (максимального значения суммы значений столбцов).
17. Создать класс, описывающий товар. Данные класса: наименование товара, цена товара, срок гарантии. Открытые функции класса: ввод данных, вывод данных, снижение цены товара на 50%.
18. Создать класс, описывающую студенческую группу. Данные класса: количество студентов в группе, количество иногородних студентов, количество иностранных студентов. Открытые функции класса: вывод на экран общего количества студентов, вывод на экран количества иногородних студентов, вывод на экран количества иностранных студентов.

Лабораторная работа № 4

Создание дружественных функций

Цель занятия

1. Приобретение навыков формирования и преобразования структурных типов данных.
2. Приобретение навыков создания и использования дружественных функций.

Постановка задачи

1. Для заданного варианта разработать алгоритм решения задачи на ЭВМ.
2. Составить программу на языке С++, которая работает с любым допустимым набором данных.
3. Входную информацию и результаты обработки вывести на печать, снабдив их соответствующими заголовками.

Содержание отчета

1.Постановка задачи для конкретного варианта.

2.Текст программы и результаты ее выполнения.

Методические указания

Дружественные функции класса определяются вне области действия этого класса, но имеют право доступа к закрытым элементам **private** данного класса. Функции или класс в целом могут быть объявлены ***другом*** (**friend**) другого класса. Дружественные функции используются для повышения производительности. Дружественные функции применяются,
как правило, для перегрузки операций, используемых классами, и для создания классов итераторов. Объекты класса итератора используются, чтобы последовательно выделять элементы или выполнять операции над элементами в объекте класса контейнера. Объекты классов контейнеров способны хранить множество элементов в форме, подобной массиву.
Чтобы объявить функцию как друга (**friend**) класса, перед ее прототипом в описании класса ставится ключевое слово **friend**.

Спецификаторы доступа к элементам private, protected и public не имеют отношения к объявлениям дружественности, так что эти объявления дружественности могут помещаться в любом месте в описании класса.

```
#include "stdafx.h"
#include<iostream>
using namespace std;
class Count {
        friend void setX(Count &, int);
public:
        Count(){ x = 0; }
        inline void print() const { cout << "x=" << x << endl; }
private:
        int x;
};
void setX(Count &c, int val)
{
        c.x = val;
}
int _tmain(int argc, _TCHAR* argv[])
{
        Count obj;
        cout << "Obj.x before setX:";
        obj.print();
        cout << "Obj.x after setX:";
        setX(obj, 10);
        obj.print();
        system("Pause");
        return 0;
}
```

Варианты заданий

1. На основе созданного класса для работы с рациональными дробями (вида m/n), создать дружественную функцию. Функция должна менять местами числитель и знаменатель
2. На основе созданного класса для работы с арифметическими комплексными числами создать дружественную функцию. Функция должна менять местами действительную и мнимую части.
3. На основе созданного класса для работы с тригонометрическими комплексными числами создать дружественную функцию. Функция должна менять местами модуль и аргумент.
4. На основе созданного класса описывающего гармонический сигнал создать дружественную функцию. Функция должна вдвое увеличивать частоту сигнала и вдвое уменьшать интервал дискретизации.
5. На основе созданного класса для работы со стеком создать дружественную функцию. Функция должна удалять вершину стека.
6. На основе созданного класса описывающего успеваемость студента создать дружественную функцию. Функция должна изменять оценки по всем трем предметам.
7. На основе созданного класса для работы с очередью создать дружественную функцию. Функция должна удалять конец очереди.
8. На основе созданного класса описывающего библиотечную карточку создать дружественную функцию. Функция должна обнулять количество всех взятых книг.
9. На основе созданного класса описывающего багаж пассажира создать дружественную функцию. Функция должна добавлять к багажу одну вещь

10. На основе созданного класса описывающего студенческую группу создать дружественную функцию. Функция должна добавлять в группу заданное количество иностранных студентов.
11. На основе созданного класса для работы с массивом создать дружественную функцию. Функция должна сортировать массив по убыванию.
12. На основе созданного класса для работы с матрицей создать дружественную функцию. Функция должна сортировать 3-ю строку по убыванию.
13. На основе созданного класса для работы с датой создать дружественную функцию. Функция должна добавлять к текущей дате определенное количество дней.
14. На основе созданного класса описывающего почтовую сортировку создать дружественную функцию. Функция должна добавлять одну посылку, доставляемую по определенному адресу.
15. На основе созданного класса описывающего запись в книге учета постояльцев в гостинице создать дружественную функцию. Функция должна добавлять один день к сроку проживания в гостинице.
16. На основе созданного класса для работы с матрицей создать дружественную функцию. Функция должна менять местами 1-й и последний элемент.
17. На основе созданного класса описывающего товар создать дружественную функцию. Функция должна менять цену товара.
18. На основе созданного класса описывающего студенческую группу создать дружественную функцию. Функция должна добавлять в группу заданное количество иностранных студентов

Лабораторная работа №5

Перегрузка операций

Цель занятия

1. Приобретение навыков перегрузки стандартных операций в классе.

Постановка задачи

1. Для заданного варианта разработать алгоритм решения задачи на ЭВМ.
2.Составить программу на языке С++, которая работает с любым допустимым набором данных.
3.Входную информацию и результаты обработки вывести на печать, снабдив их соответствующими заголовками.

Содержание отчета

1. Постановка задачи для конкретного варианта.
2.Текст программы и результаты ее выполнения.

Методические указания

Операции перегружаются путем составления описания функции (с заголовком и телом), за исключением того, что в этом случае имя функции состоит из ключевого слова **operator**, после которого записывается перегружаемая операция. Например, имя функции **operator**+ можно использовать для перегрузки операции сложения.

Чтобы использовать операцию над объектами классов, эта операция *должна* быть перегружена, но есть два исключения. Операция присваивания (=) может быть использована с каждым классом без явной перегрузки. По умолчанию операция присваивания сводится к *побитовому копированию* данных-элементов класса. Мы увидим вскоре, что такое побитовое копирование опасно для классов с элементами, которые указывают на динамически выделенные области памяти; для таких классов мы будем явно перегружать операцию присваивания. Операция адресации (&) также может быть использована с объектами любых классов без перегрузки; она просто возвращает адрес объекта в памяти. Но операцию адресации можно также и перегружать.

Ограничения на перегрузку операции

Большинство операций С++ перегружать можно. Они показаны в таблице
1. В таблице 2 показаны операции, которые перегружать нельзя.

Таблица 1. Операции, которые могут быть перегружены

+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	I=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	, ->	[ ]	( )	**new**	**delete**	

Таблица 2. Операции, которые не могут быть перегружены

.	.*	::	?:	sizeof

Старшинство операций не может быть изменено перегрузкой. Это могло бы привести к щекотливым ситуациям, в которых операция перегружается таким образом, для которого установленное в языке старшинство не подходит. Однако с помощью скобок можно принудительно изменить последовательность оценки перегруженных операций в выражениях.

Ассоциативность операций не может быть изменена перегрузкой. С перегруженными операциями нельзя использовать аргументы по умолчанию. Изменить количество операндов, которое берет операция, невозможно: перегруженные унарные операции остаются унарными, перегруженные бинарные операции остаются бинарными. В C++ не может быть перегружена единственная тернарная операция ?:. Каждая из операций &, *, + и – может иметь унарный и бинарный варианты; эти унарные и бинарные варианты могут перегружаться раздельно. Создавать новые операции невозможно; перегружать можно только уже существующие операции.

Нельзя изменить с помощью перегрузки операции смысл работы операции с объектом встроенного типа. Программист, например, не может изменить смысл того, как с помощью + складываются два целых числа. Перегрузка операций применима только для работы с объектами типов, определенных пользователем, или со смесью объектов типов, определенных пользователем, и встроенных типов.

Основы перегрузки операций

При перегрузке операций ( ), [ ], -> или = функция перегрузки операции должна быть объявлена как элемент класса. Для других операций функции перегрузки операций могут не быть функциями-элементами (тогда они обычно объявляются друзьями).

Когда функция-операция реализована как функция-элемент, крайний левый (или единственный) операнд должен быть объектом того класса (или ссылкой на объект того класса), элементом которого является функция.

Если левый операнд должен быть объектом другого класса или встроенного типа, такая функция-операция не может быть реализована как функция-элемент. Функция-операция, реализованная не как функция-элемент, должна быть другом, если эта функция должна иметь прямой доступ к закрытым или защищенным элементам этого класса. Перегруженная операция << должна иметь левый операнд типа ostream & (такой, как cout в выражении cout << classObject), так что она не может быть функцией-элементом. Аналогично, перегруженная операция >> должна иметь левый операнд типа istream & (такой, как cin в выражении cin << classObject), так что она тоже не может быть функцией-элементом. К тому же каждая из этих перегруженных функций-операций может потребовать доступа к закрытым элементам данным объекта класса, являющегося входным или выходным потоком, так что эти перегруженные функции-операции делают иногда функциями-друзьями класса из соображений эффективности.

Перегрузка унарных операций

Унарную операцию класса можно перегружать как нестатическую функцию-элемент без аргументов, либо как функцию, не являющуюся элементом, с одним аргументом; этот аргумент должен быть либо объектом класса, либо ссылкой на объект класса. Функции-элементы, которые реализуют перегруженные операции, должны быть нестатическими, чтобы они могли иметь доступ к данным класса. Напомним, что статические функции-элементы могут иметь доступ только к статическим данным-элементам класса.

При перегрузке унарных операций предпочтительнее создавать функции операции,

являющиеся элементами класса, вместо дружественных функций, не являющихся элементами. Дружественных функций и дружественных классов лучше избегать до тех пор, пока они не станут абсолютно необходимыми.
Использование друзей нарушает инкапсуляцию класса.

```
#include"stdafx.h"
#include<iostream>
using namespace std;
class Value1
{
	friend void operator++(Value1 &); // префиксная форма
	friend void operator++(Value1 &, int); // постфиксная форма
public:
	Value1(int, int);
	void print();
private:
	int x1;
	int x2;
};
Value1::Value1(int a, int b) { x1 = a; x2 = b; }
void Value1::print(){
	cout << "x1=" << x1 << endl;
	cout << "x2=" << x2 << endl;
}
void operator++(Value1 &B)
{
	B.x1 = B.x1 + 1;
	B.x2 = B.x2 + 1;
}
void operator++(Value1 &B, int)
{
	B.x1 = B.x1 + 1;
	B.x2 = B.x2 + 1;
}
int main()
{
	Value1 A(10, 20);
	A.print();
	++A;
	A.print();
	A++;
	A.print();
	system("PAUSE");
	return 0;
}
```

Перегрузка бинарных операций

Бинарную операцию можно перегружать как нестатическую функцию элемент с одним аргументом, либо как функцию, не являющуюся элементом, с двумя аргументами (один из этих аргументов должен быть либо объектом класса, либо ссылкой на объект класса).
Файл **value2.h**. Описание класса Value2.

```
class Value2{
public:
Value2 & operator+(Value2 &);
void operator=(Value2 &);
```

```
Value2(int=0, int=0);
void print( );
private:
int x1;
int x2;
int c1;
int c2;
};
```

Файл **value2.cpp**. Реализация класса Value2.

```
#include<iostream.h>
#include"value2.h"
Value2::Value2(int a, int b)
{
x1=a;
x2=b;
}
void Value2::print( ){
cout<<"x1="<<x1<<endl;
cout<<"x2="<<x2<<endl;
}
Value2 & Value2::operator+(Value2 &v1)
{
c1=x1+v1.x1;
c2=x2+v1.x2;
return *this;
}
void Value2::operator=(Value2 &v)
{
x1=v.c1;
x2=v.c2;
}
```

Файл **main.cpp**. Драйвер класса Value2.

```
#include<iostream.h>
#include"value2.h"
main( )
{
Value2 Obj1(1,1), Obj2(2,2), Obj3;
Obj3=Obj2+Obj1;
cout<<"Obj1:"<<endl;
Obj1.print();
cout<<"Obj2:"<<endl;
Obj2.print();
cout<<"Obj3:"<<endl;
Obj3.print();
system("Pause");
return 0;
}
```

Варианты заданий

1. На основе созданного класса для работы с рациональными дробями, перегрузить операции «+», «-» и «=» для осуществления операций сложения и вычитания целых чисел с дробями.
2. На основе созданного класса для работы с арифметическими комплексными числами, перегрузить операции «+», «-» и «=» для осуществления операций сложения и вычитания двух комплексных чисел.
3. На основе созданного класса для работы с тригонометрическими комплексными числами, перегрузить операции «**», «/» и «=» для осуществления операций умножения и деления двух комплексных чисел.
4. На основе созданного класса для работы гармоническим сигналом. Перегрузить операцию «+=» для увеличения амплитуды на дробное число. Перегрузить операцию «-=» для уменьшения фазы определенное количество градусов. Перегрузить операцию «*=» для умножения частоты на целое число.
5. На основе созданного класса, описывающего стек типа LIFO, перегрузить операции «++» для добавления элемента в вершину стека, и операцию «--» для чтения элемента из вершины стека. Операции должны перегружаться для префиксной и постфиксной формы.
6. На основе созданного класса, описывающего успеваемость студента, перегрузить операции «<=», «=>» и «==» для сравнения двух объектов класса по среднему баллу.
7. На основе созданного класса, описывающего очередь типа FIFO, перегрузить операции «++» для добавления элемента в начало очереди, и операцию «--» для чтения элемента из конца очереди. Операции должны перегружаться для префиксной и постфиксной формы.
8. На основе созданного класса описывающего библиотечную карточку. Перегрузить операции «>>», «<<» для ввода и вывода данных карточки. Перегрузить операции «++» и «--» для увеличения или уменьшения количества взятых книг на 1.
9. На основе созданного класса, описывающего багаж пассажира, перегрузить операции «+», «-» и «=» для добавления в багаж или удаления из багажа определенного числа вещей.
10. На основе созданного класса, описывающего студенческую группу, перегрузить операции «+», «-» и «=» для добавления в группу или удаления из группы определенного числа иностранных студентов.
11. На основе созданного класса, описывающего работу с массивом, перегрузить операции «>>», «<<» для ввода и вывода массива. Перегрузить операции «++» и «--» увеличения или уменьшения каждого элемента массива на 1.
12. На основе созданного класса, описывающего работу с матрицей, перегрузить операции «>>», «<<» для ввода и вывода матрицы. Перегрузить операции «++» и «--» увеличения или уменьшения каждого элемента матрицы на 1.
13. На основе созданного класса, описывающего работу с датой, перегрузить операции «+», «-» и «=» для осуществления операций сложения и вычитания текущей даты и целого числа дней.
14. На основе созданного класса, описывающего работу почтовую сортировку, перегрузить операции «<», «>» и «==» для сравнения двух объектов класса по суммарному количеству посылок.
15. На основе созданного класса, описывающего запись в книге учета постояльцев в гостинице, перегрузить операции «++» и «--» для увеличения или уменьшения количества дней проживания на 1. Перегрузить операцию «<<» для вывода даты отъезда.
16. На основе созданного класса, описывающего работу с матрицей, перегрузить операции «>>», «<<» для ввода и вывода матрицы. Перегрузить операции «+», «-» и «=» для осуществления операций сложения и вычитания каждого элемента матрицы и целого числа.
17. На основе созданного класса, описывающего работу с товаром, перегрузить операции «<», «>» и «==» для сравнения двух объектов класса по цене товара.

18. На основе созданного класса, описывающего студенческую группу, перегрузить операции «+», «-» и «=» для добавления в группу или удаления из группы определенного числа иностранных студентов.

Лабораторная №6

Реализация механизмов наследования

Цель занятия

1. Приобретение навыков создания базовых и производных классов в языке C++.

Постановка задачи

1. Для заданного варианта разработать алгоритм решения задачи на ЭВМ.
2. Составить программу на языке C++, которая работает с любым допустимым набором данных.
3. Входную информацию и результаты обработки вывести на печать, снабдив их соответствующими заголовками.

Содержание отчета

1. Постановка задачи для конкретного варианта.
2. Текст программы и результаты ее выполнения.

Методические указания

Проектирование программного обеспечения с помощью наследования

Наследование – это способ повторного использования программного обеспечения, при котором новые классы создаются из уже существующих классов путем заимствования их атрибутов и функций и обогащения этими возможностями новых классов.
При создании нового класса вместо написания полностью новых данных элементов и функций-элементов программист может указать, что новый класс является *наследником* данных-элементов и функций-элементов ранее определенного *базового класса.* Этот новый класс называется *производным классом.* Каждый производный класс сам является кандидатом на роль базового класса для будущих производных классов. При *простом наследовании* класс порождается одним базовым классом. При *множественном наследовании* производный класс наследует нескольким базовым классам (возможно неродственным).

Каждый объект производного класса является также объектом соответствующего базового класса. Однако, обратное утверждение неверно: объект базового класса не является объектом классов, порожденных этим базовым классом.

Производный класс не может иметь доступ к закрытым элементам своего базового класса; разрешение такого доступа явилось бы нарушением инкапсуляции базового класса. Производный класс может, однако, иметь доступ к открытым и защищенным элементам своего базового класса.

Элементы базового класса, которые не должны быть доступны производному классу через наследование, объявляются в базовом классе закрытыми. Производный класс может иметь доступ к закрытым элементам своего базового класса только посредством функций доступа, предусмотренных в открытом интерфейсе базового класса.

Защищенный уровень доступа (protected) служит промежуточным уровнем защиты между открытым доступом и закрытым доступом. Защищенные элементы базового класса могут быть доступны только элементам и друзьям базового класса и элементам и друзьям производного класса.

Одной из проблем наследования является то, что производный класс может наследовать открытые функции-элементы, когда в этом нет необходимости или когда это недопустимо. Имеется возможность избежать этого. Когда какой-то элемент базового класса не подходит для задач производного класса, этот элемент может быть соответствующим образом переопределен в производном классе.

Базовые классы и производные классы

Чтобы указать, что класс Student порожден классом Anketa, класс Student должен быть определен следующим образом:

```
class Student: public Anketa {
};
```

Это называется *открытым наследованием (public inheritance)*. Существует также *закрытое наследование (private inheritance)* и *защищенное наследование (protected inheritance)*. При открытом наследовании открытые и защищенные элементы базового класса наследуются как открытые и защищенные элементы производного класса соответственно. Помните, что закрытые элементы базового класса не доступны в производных классах.

Спецификатор доступа к элементам в базовом классе	Тип наследования		
	public – открытое наследование	protected – защищенное наследование	private – закрытое наследование
public	public в производном классе	protected в производном классе	private в производном классе
protected	protected в производном классе	protected в производном классе	private в производном классе
private	невидим в производном классе	невидим в производном классе	невидим в производном классе

Наследование позволяет избежать дублирования лишнего кода при написании классов. Пусть в базе данных ВУЗа должна храниться информация о всех студентах и преподавателях. Представлять все данные в одном классе не получится, поскольку для преподавателей нам понадобится хранить данные, которые для студента не применимы, и наоборот.

Создание базового класса

Для решения этой задачи создадим базовый класс human, который будет описывать модель человека. В нем будут храниться имя, фамилия и отчество.

Создайте файл human.h:

```
// human.h
#ifndef HUMAN_H_INCLUDED
#define HUMAN_H_INCLUDED

#include <string>
#include <sstream>

class human {
   public:
      // Конструктор класса human
      human(std::string last_name, std::string name, std::string second_name)
      {
         this->last_name = last_name;
         this->name = name;
         this->second_name = second_name;
      }

      // Получение ФИО человека
      std::string get_full_name()
      {
         std::ostringstream full_name;
         full_name << this->last_name << " "
            << this->name << " "
```

```
            << this->second_name;
        return full_name.str();
    }

  private:
    std::string name; // имя
    std::string last_name; // фамилия
    std::string second_name; // отчество
};

#endif // HUMAN_H_INCLUDED
```

Наследование от базового класса

Теперь создайте новый класс student, который будет наследником класса human. Поместите его в файл student.h.

```
// student.h
#ifndef STUDENT_H_INCLUDED
#define STUDENT_H_INCLUDED

#include "human.h"
#include <string>
#include <vector>

class student : public human {
  public:
    // Конструктор класса Student
    student(
        std::string last_name,
        std::string name,
        std::string second_name,
        std::vector<int> scores
    ) : human(
        last_name,
        name,
        second_name
    ) {
        this->scores = scores;
    }

    // Получение среднего балла студента
    float get_average_score()
    {
        // Общее количество оценок
        unsigned int count_scores = this->scores.size();
        // Сумма всех оценок студента
        unsigned int sum_scores = 0;
        // Средний балл
        float average_score;

        for (unsigned int i = 0; i < count_scores; ++i) {
            sum_scores += this->scores[i];
        }
```

```
        average_score = (float) sum_scores / (float) count_scores;
        return average_score;
    }

private:
    // Оценки студента
    std::vector<int> scores;
};
#endif // STUDENT_H_INCLUDED
```

Функция get_average_score вычисляет среднее арифметическое всех оценок студента.
Все публичные свойства и методы класса human будут доступны в классе student.

Конструктор базового класса

Для того, чтобы инициализировать конструктор родительского класса (в нашем случае — это сохранение имени, фамилии и отчества ученика), используется следующий синтаксис:

```
// Конструктор класса Student
student(
    // аргументы конструктора текущего класса
) : human(
    // инициализация конструктора родительского класса
) {
    // инициализация конструктора текущего класса
}
```

В конструктор класса human мы передаем инициалы человека, которые сохраняются в экземпляре класса. Для класса students, нам необходимо задать еще и список оценок студента. Поэтому конструктор students принимает все аргументы конструктора базового класса, а также дополнительные аргументы для расширения функционала:

```
// Конструктор класса Student
student(
    std::string last_name,
    std::string name,
    std::string second_name,
    std::vector<int> scores
) : human(
    last_name,
    name,
    second_name
) {
    this->scores = scores;
}
```

Список оценок студента хранится в векторе.

Создание объекта класса student

Реализуем пользовательский интерфейс для работы с классом student.

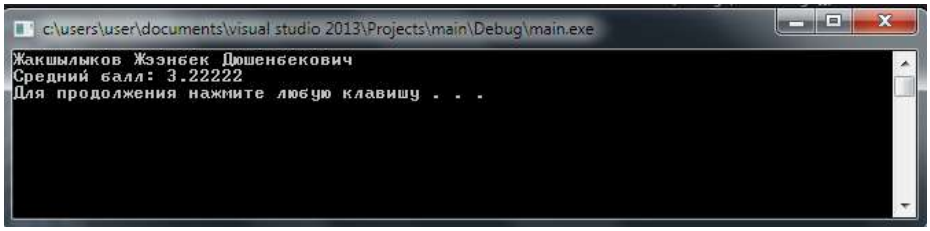
```
// main.cpp

#include <iostream>
```

```
#include <vector>

#include "human.h"
#include "student.h"

int main(int argc, char* argv[])
{

    // Оценки студента
    std::vector<int> scores;

    // Добавление оценок студента в вектор
    scores.push_back(5);
    scores.push_back(3);
    scores.push_back(2);
    scores.push_back(2);
    scores.push_back(5);
    scores.push_back(3);
    scores.push_back(3);
    scores.push_back(3);
    scores.push_back(3);

    // Создание объекта класса student
    student *stud = new student(""Жакшылыков", "Жээнбек", "Дюшенбекович"", scores);

    // Вывод полного имени студента (используется унаследованный метод класса human)
    std::cout << stud->get_full_name() << std::endl;
    // Вывод среднего балла студента
    std::cout << "Средний балл: " << stud->get_average_score() << std::endl;
system("pause");
    return 0;
}
```

В этом примере мы написали программу, которая создает объект класса student, сохраняя в нем его имя, фамилию, отчество и список оценок.

После инициализации объекта, происходит вывод полного имени студента с помощью функции get_full_name. Эта функция была унаследована от базового класса human.

Затем программа вычисляет средний балл студента и выводит его на экран. Этим занимается функция get_average_score, которую мы описали внутри класса student.



Создание класса-наследника teacher

Нужно создать еще один класс, в котором будут храниться данные преподавателей. Дадим ему название — teacher. Как вы уже поняли, мы не будем описывать все методы этого класса с

нуля, а просто унаследуем его от класса human. Тогда, не нужно будет реализовывать хранение имени, фамилии и отчества препода. Это уже есть в базовом классе human.

Создайте файл teacher.h:

```
// teacher.h
#ifndef TEACHER_H_INCLUDED
#define TEACHER_H_INCLUDED

#include "human.h"
#include <string>

class teacher : public human {
   // Конструктор класса teacher
   public:
      teacher(
         std::string last_name,
         std::string name,
         std::string second_name,
         // Количество учебных часов за семетр у преподавателя
         unsigned int work_time
      ) : human(
         last_name,
         name,
         second_name
      ) {
         this->work_time = work_time;
      }

      // Получение количества учебных часов
      unsigned int get_work_time()
      {
         return this->work_time;
      }

   private:
      // Учебные часы
      unsigned int work_time;
};

#endif // TEACHER_H_INCLUDED
```

У класса teacher появилось новое свойство — количество учебных часов, отведенное преподавателю на единицу времени (семестр). Весь остальной функционал наследуется от базового класса human. Если бы мы писали все с нуля, то одинакового кода бы получилось в разы больше, и его поддержка усложнилась бы на порядок.

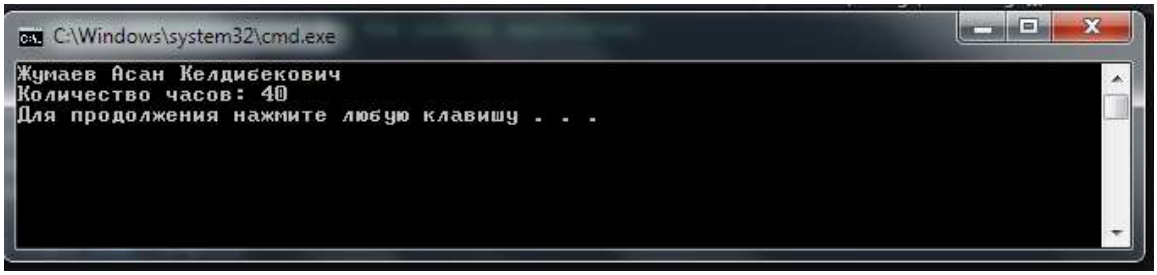
Создание объекта класса teacher

Изменим содержимое файла main.cpp, чтобы проверить работу класса teacher.

```
#include <iostream>

#include "human.h"
#include "teacher.h"
```

```
int main(int argc, char* argv[])
{

   // Количество учебных часов преподавателя
   unsigned int teacher_work_time = 40;

   teacher *tch = new teacher ("Жумаев", "Асан", "Келдибекович", teacher_work_time);

   std::cout << tch->get_full_name() << std::endl;
   std::cout << "Количество часов: " << tch->get_work_time() << std::endl;

   return 0;
}
```

Если сборка программы прошла без ошибок, то результат работы программы будет таким:



Можно таким же образом создать класс, в котором будут храниться данные обслуживающего персонала или руководящего состава. Наследование используют, когда у каждой группы объектов есть общие параметры, но для каждой из этих групп нужно хранить более кастомные данные.

Также, мы можем создать класс, который будет описывыть студента заочной формы обучения. Его мы унаследовали бы от класса student, добавив какие-либо дополнительные данные.

В класс human можно добавить еще больше свойств, которые будут описывать данные, имеющиеся у любого человека. Например, номер паспорта, дату рождения, прописку и место проживания.

Подобный подход позволяет в разы уменьшить дублирование кода в реальных проектах, и упросить его поддержку.

Варианты заданий

1. На основе класса «Дробь» создать производный класс «Правильная дробь». Конструктор класса проверяет m и n, и если m>n, приводит дробь к виду d·(k/n). Класс дополнительно содержит функцию вычисления высоты исходной дроби. В классе должна быть переопределена функция вывода дроби на экран.
2. На основе класса «Арифметическое комплексное число» создать производный класс «Тригонометрическое комплексное число». Конструктор класса, используя введенные значения a и b, рассчитывает значения r и φ. В классе должна быть переопределена функция вывода комплексного числа на экран. В классе должна быть перегружена операция == для сравнения двух комплексных чисел.
3. На основе класса «Тригонометрическое комплексное число» создать производный класс «Арифметическое комплексное число». Конструктор класса, используя введенные значения r и φ, рассчитывает значения a и b. В классе должна быть переопределена функция вывода комплексного числа на экран. В классе должна быть перегружена операция != для сравнения двух комплексных чисел.

4. На основе класса «Гармонический сигнал» создать производный класс «Амплитудно-модулированный сигнал». Класс содержит массив $A[n]=A0\cdot\sin(\Omega\cdot n\cdot\Delta t+\varphi 0)$, и коэффициент модуляции m. Класс преобразует сигнал в массив $s[n]=(1+m\cdot A[n])\cdot\sin(\omega\cdot n\cdot\Delta t+\varphi 0)$. В классе должна быть переопределена функция вывода значений на экран.
5. На основе класса LIFO создать производный класс «Сдвиговый регистр». В классе должна быть определена функция сдвигающая содержимое регистра влево и вправо.
6. На основе класса «Успеваемость студента» создать производный класс «Зачетка». Класс должен дополнительно содержать оценки по четырем зачетам. В классе должны быть переопределены функции расчета среднего значения, ввода и вывода значений на экран.
7. На основе класса «FIFO» создать производный класс «Дек». В классе должна быть определена функция чтения и записи в начало стека. В классе должна быть определена функция подсчета количества значений дека.
8. На основе класса «Библиотечная карточка» создать производный класс «Расширенная библиотечная карточка». Класс содержит количество взятых учебников, художественных книг и журналов. В классе должна быть переопределена функция вывода на экран записей карточки. В классе должна быть определена функция определения процентного состава взятых учебников, художественных книг и журналов.
9. На основе класса «Багаж» создать производный класс «Расширенный багаж». В классе содержится строковый массив, содержащий имя пассажира и лимит веса. В классе должна быть переопределена функция вывода на экран всех пассажиров с количеством и весом багажа. В классе должна быть определена функция определения оставшегося до лимита веса.
10. На основе класса «Студенческая группа» создать производный класс «Студенческая группа». Класс содержит номер группы, ФИО каждого студента и средний балл. В классе должна быть переопределена функция вывода на экран студентов группы.
11. На основе класса «Массив» создать производный класс «Записная книжка». Данный класс содержит строковый массив, каждый элемент которого содержит имя и связан с соответствующим элементом исходного массива. В классе должны быть переопределены функции ввода и вывода значений обоих массива. В классе должна быть определена функция поиска значения исходного массива по значению соответствующего строкового массива.
12. На основе класса «Матрица» создать производный класс «Система алгебраических уравнений». Класс содержит одномерный массив B[3]. В классе определить функцию решения системы алгебраических уравнений методом Крамера.
13. На основе класса «Дата» создать производный класс «Полная дата». В классе должно содержаться значение часа, минуты и секунды. В классе переопределить функцию вывода значений на экран в американском и европейском стандарте. В классе должная быть определена функция вычисления временного интервала в днях.
14. На основе класса «Почтовая сортировка» создать производный класс «Расширенная почтовая сортировка». Класс содержит тип доставки: воздушная, наземная, железнодорожная, стоимость доставки. В классе должна быть переопределена функция вывода на экран в зависимости от типа доставки. В классе должна быть определена функция расчет общей стоимости посылки с учетом доставки.
15. На основе класса, описывающего запись в книге учета постояльцев в гостинице создать класс «Услуги». Класс должен содержать наличие дополнительных услуг постояльцу: завтраки, обеды, ужины, наличие интернета, а также цены за все услуги. Переопределить функции вывода и расчета стоимости проживания в гостинице.
16. На основе класса «Матрица» создать производный класс «Система алгебраических уравнений». Класс содержит одномерный массив B[3]. В классе определить функцию решения системы алгебраических уравнений методом Крамера.

17. На основе класса «Товар» создать класс «Расширенный товар». Класс должен содержать сведения о стране производителе, весе товара, объеме товара. Переопределить функцию вывода на экран.
18. На основе класса «Студенческая группа» создать производный класс «Студенческая группа». Класс содержит номер группы, ФИО каждого студента и средний балл. В классе должна быть переопределена функция вывода на экран студентов группы.

Лабораторная работа № 7

Реализация механизмов полиморфизма

Цель занятия

1. Приобретение навыков создания абстрактных и конкретных классов в языке C++.

Постановка задачи

1. Для заданного варианта разработать алгоритм решения задачи на ЭВМ.
2. Составить программу на языке C++, которая работает с любым допустимым набором данных.
3.Входную информацию и результаты обработки вывести на печать, снабдив их соответствующими заголовками.

Содержание отчета

1. Постановка задачи для конкретного варианта.
2. Текст программы и результаты ее выполнения.

Методические указания

C++ включает такое свойство, как *полиморфизм* – возможность для объектов разных классов, связанных с помощью наследования, реагировать различным образом при обращении к одной и той же функции-элементу.

Полиморфизм реализуется посредством виртуальных функций. Если при использовании виртуальной функции запрос осуществляется с помощью указателя базового класса (или ссылки), то C++ выбирает правильную переопределенную функцию в соответствующем производном классе, связанном с данным объектом.

Виртуальные функции

Функция объявляется виртуальной с помощью ключевого слова virtual, предшествующего прототипу функции в базовом классе. Например, в базовом классе Ancestor можно написать

virtual void print( ) **const**;

Этот прототип объявляет, что функция print является константной функцией, которая не принимает никаких аргументов, ничего не возвращает и является виртуальной функцией. Если функция однажды объявлена виртуальной, то она остается виртуальной на любом более низком уровне иерархической структуры.

```
#include"stdafx.h"
#include<iostream>
using namespace std;
class Ancestor
{
public:
        void print() { cout << "Ancestor::print()" << endl; }
        virtual void print_v() { cout << "Ancestor::virtual print( )" << endl; }
};
class Descendant : public Ancestor
{
public:
        void print() { cout << "Descendant::print( )" << endl; }
        virtual void print_v() { cout << "Descendant::virtual print( )" << endl; }
};
int main()
```

```
{
        Descendant *DPtr = new Descendant;
        Ancestor *APtr = DPtr;
        DPtr->print();
        APtr->print();
        cout << endl;
        DPtr->print_v();
        APtr->print_v();
        system("PAUSE");
        return 0;
}
```

Абстрактные базовые классы и конкретные классы

Когда мы думаем о классе как о типе, мы предполагаем, что будут создаваться объекты этого типа. Однако имеются случаи, в которых полезно определять классы, для которых программист не намерен создавать какие-либо объекты. Такие классы называются *абстрактными классами.* Поскольку они применяются в качестве базовых классов в процессе наследования, мы обычно будем называть их *абстрактными базовыми классами.* Объекты абстрактного базового класса не могут быть реализованы.

Единственным назначением абстрактного класса является создание соответствующего базового класса, от которого другие классы могут унаследовать интерфейс и реализацию. Классы, объекты которых могут быть реализованы, называются *конкретными классами.*

Хотя не можем создавать объекты абстрактного базового класса, *можем* объявить указатели на абстрактный базовый класс. Эти указатели могут быть затем использованы, чтобы предоставить возможность для полиморфного оперирования объектами производных конкретных классов.

Класс делается абстрактным путем объявления одной или более его виртуальных функций чисто виртуальными. *Чистой виртуальной функцией* является такая функция, у которой в ее прототипе тело определено как 0 (*инициализатор* равен **0).**

Новые классы и динамическое связывание

Полиморфизм и виртуальные функции могут прекрасно работать, если все возможные классы известны заранее. Но они также работают, когда в систему добавляются новые типы классов.

Новые классы встраиваются при помощи динамического связывания (называемого также *поздним связыванием).* Во время компиляции нет необходимости знать тип объекта, чтобы скомпилировать вызов виртуальной функции. Во время выполнения программы вызов виртуальной функции будет соответствовать функции-элементу вызванного объекта.

Если функция в базовом классе объявлена как virtual и если мы, затем вызываем функцию через указатель базового класса, указывающий на объект производного класса, то программа будет динамически (т.е. во время выполнения программы) выбирать соответствующую функцию производного класса. Это называется *динамическим связыванием*

Когда виртуальная функция вызывается путем обращения к заданному объекту по имени и при этом используется операция доступа к элементу точка, тогда эта ссылка разрешается (обрабатывается) во время компиляции это называется *статическим связыванием* и в качестве вызываемой определяется функция класса данного объекта (или наследуемая этим классом).

Динамическое связывание позволяет независимым дистрибьютерам программного обеспечения распространять свою продукцию, не выдавая фирменных секретов. Распространяемое программное обеспечение может состоять только из заголовочных и объектных файлов. Чтобы не раскрывать секреты программного обеспечения, не должно прикладываться никаких исходных текстов. Разработчики программного обеспечения могут использовать наследование для создания новых производных классов на основе тех классов, которые предоставлены им дистрибьютерами программного обеспечения. Программные

средства, которые работают с классами, предоставленными дистрибьютерами, будут продолжать работать и с производными классами, используя (с помощью динамического связывания) переопределенные виртуальные функции, имеющиеся в этих классах.

Ниже показано, каким образом компилятор и программа (во время выполнения) управляют полиморфизмом с незначительными затратами.

Динамическое связывание требует, чтобы во время выполнения программы вызов виртуальной функции-элемента был бы направлен варианту виртуальной функции соответствующего класса. Для этого служит *таблица виртуальных методов* или *vtable,* которая реализуется в виде массива, содержащего указатели на функции. У каждого класса, который содержит виртуальные функции, имеется таблица виртуальных методов. Для каждой виртуальной функции в классе таблица имеет элемент, содержащий указатель на вариант виртуальной функции, используемый в объектах данного класса. Виртуальная функция, используемая в некотором классе, может быть определена в этом классе или прямо или косвенно наследоваться из базового класса, стоящего выше в иерархии.

Если базовый класс имеет виртуальную функцию-элемент, то производные классы могут переопределить эту функцию, но они могут этого и не делать. Таким образом, производный класс может использовать вариант виртуальной функции-элемента базового класса и это будет отражено в таблице виртуальных методов.

Каждый объект класса, содержащего виртуальные функции, имеет указатель на таблицу виртуальных методов этого класса, недоступный для программиста. Во время выполнения программы полиморфные вызовы виртуальных функций осуществляются через разыменование указателя объекта на таблицу виртуальных методов, что дает доступ к таблице виртуальных методов класса. Затем в таблице виртуальных методов находится соответствующий указатель на функцию, он разыменовывается, что и завершает вызов виртуальной функции во время выполнения программы. Просмотр таблицы виртуальных методов и операция разыменования указателя требуют минимальных затрат времени выполнения.

Виртуальные деструкторы

При использовании полиморфизма для обработки динамически размещенных объектов иерархии классов может появиться одна проблема. Если объект уничтожается явным использованием операции delete над указателем базового класса на объект, то вызывается деструктор базового класса данного объекта. Это происходит вне зависимости от типа объекта, на который указывает указатель базового класса и вне зависимости от того факта, что деструкторы каждого класса имеют разные имена.

Существует простое решение этой проблемы: объявление деструктора базового класса виртуальным. Это автоматически приведет к тому, что все деструкторы производных классов станут виртуальными, даже если они имеют имена, отличные от имени деструктора базового класса. В этом случае, если объект в иерархии уничтожен явным использованием операции delete, примененной к указателю базового класса на объект производного класса, то будет вызван деструктор соответствующего класса. Вспомним, что когда производный класс уничтожен, часть базового класса, содержащаяся в производном классе, также уничтожается. Деструктор базового класса автоматически выполняется после деструктора производного класса.

Классы могут наследовать и интерфейс, и реализацию базового класса. В иерархиях, проектируемых для наследования реализации, стремятся обеспечить функциональные возможности на возможно более высоких уровнях, чтобы каждый новый производный класс мог наследовать функции-элементы, описанные в базовом классе, и использовать эти описания. В иерархиях, проектируемых для наследования интерфейса, стремятся обеспечить функциональные возможности на возможно более низких уровнях: базовый класс объявляет функции, которые должны вызываться идентично для каждого объекта в иерархии (т.е. иметь одинаковую сигнатуру), а производные классы обеспечивают для них свои собственные реализации.

Базовый класс Shape состоит их четырех открытых виртуальных функций и не содержит каких-либо данных. Функции printShapeName и print являются чисто виртуальными, так как они переопределяются в каждом производном классе. Функции area и volume определены, причем по определению они возвращают 0.0. Эти функции переопределяются в тех производных классах, в которых требуется использовать их для соответствующих вычислений площади и объема.

Класс Point является производным от класса Shape с открытым наследованием. Точка не имеет ни площади, ни объема, так что функцииэлементы базового класса area и volume в данном случае не переопределены; они наследуют их определения из базового класса Shape. Функции printShapeName и print являются реализациями виртуальных функций, которые были определены в базовом классе как чисто виртуальные. Другими функциями-элементами являются функция setPoint, присваивающая новые значения координатам точки *x* и *y*, и функции getX и getY, возвращающие координаты *x* и *y*.

Класс Circle является производным от класса Point с открытым наследованием. Круг не имеет объема, так что функция-элемент базового класса volume для класса Circle не переопределяется; она наследуется из базового класса Shape (через класс Point). Круг имеет площадь; поэтому функция area в классе Circle переопределяется. Функции printShapeName и print являются реализациями виртуальных функций, которые были определены в базовом классе как чисто виртуальные. Если бы эти функции здесь не переопределялись, то наследовались бы версии этих функций из класса Point. Другими функциями-элементами являются функция setRadius, присваивающая новое значение радиусу radius, и функция getRadius, возвращающая значение радиуса.

Класс Cylinder является производным от класса Circle с открытым наследованием. Цилиндры имеют площадь и объем; поэтому в классе Cylinder обе функции area и volume переопределены. Функции printShapeName и print являются реализациями виртуальных функций, которые были определены в базовом классе как чисто виртуальные. Если бы эти функции здесь не переопределялись, то наследовались бы версии этих функций из класса Circle.

Другими функциями-элементами являются функция setHeight, присваивающая новое значение высоты цилиндра height, и функция getHeight, возвращающая значение высоты.

Программа драйвер начинает свою работу с создания объекта pnt класса Point, объекта crl класса Circle и объекта cnd класса Cylinder. Для каждого объекта вызывается функция printShapeName и каждый объект выводится с помощью своей перегруженной операции “поместить в поток”, чтобы показать правильную инициализацию объектов. Затем объявляется массив sPtr типа Shape *. Этот массив указателей базового класса используется для того, чтобы указывать на каждый созданный объект производного класса. Сначала элементу массива sPtr[0] присваивается адрес объекта pnt, затем элементу sPtr[1] присваивается адрес объекта crl, а элементу sPtr[2] присваивается адрес объекта cnd. После этого выполняется структура for, просматривающая массив sPtr и выполняющая на каждой итерации цикла следующие вызовы:

```
sPtr[i]->printShapeName;
sPtr[i]->print( );
sPtr[i]->area( );
sPtr[i]->volume( );
```

Каждый из приведенных выше вызовов активизирует эти функции для того объекта, на который указывает элемент массива sPtr[i]. Из выходныхрезультатов, следует, что функции вызываются должным образом. Сначала выводится строка "Точка:" и координаты, хранящиеся в объекте pnt; площадь и объем равны 0.00. Затем выводится строка "Круг:", координаты центра круга и радиус, хранящиеся в объекте crl; потом вычисляется площадь круга, а его объем равен 0.00. И, наконец, выводится строка "Цилиндр", координаты центра, радиус и высота цилиндра, хранящиеся в объекте cnd; потом вычисляются площадь его поверхности и объем. Все вызовы функций printShapeName, print, area и volume реализуются в процессе выполнения программы с помощью динамического связывания.

Варианты заданий.

1. Создать абстрактный класс «Целое число», который является базовым для класса «Дробь». В классе должна быть определена чистая виртуальная функция для расчета десятичной дроби, переопределённая в классах «Дробь» и «Правильная дробь».
2. Создать абстрактный класс «Целое число», который является базовым для класса «Арифметическое комплексное число». В классе должна быть определена чистая виртуальная функция для расчета модуля комплексного числа, переопределённая в классах «Арифметическое комплексное число» и «Тригонометрическое комплексное число».
3. Создать абстрактный класс «Целое число», который является базовым для класса «Тригонометрическое комплексное число» создать производный класс «Арифметическое комплексное число». В классе должна быть определена чистая виртуальная функция для расчета аргумента комплексного числа, переопределённая в классах «Арифметическое комплексное число» и «Тригонометрическое комплексное число».
4. Создать абстрактный класс «Массив», который является базовым для класса «Гармонический сигнал» создать производный класс «Амплитудно-модулированный сигнал» ». В классе должны быть определены чистые виртуальная функция для ввода и вывода значений, переопределённые в классах «Гармонический сигнал» «Амплитудно модулированный сигнал»
5. Создать абстрактный класс «Массив», который является базовым для класса «LIFO». В классе должна быть определена чистая виртуальная функция для ввода значений, переопределённая в классах «LIFO» и «Сдвиговый регистр».
6. Создать абстрактный класс «Анкета», который является базовым для класса «Успеваемость студента» и содержит ФИО человека. В классе должны быть определены чистые виртуальная функция для ввода и вывода значений, переопределённые в классах «Успеваемость студента» и «Зачетка».
7. Создать абстрактный класс «Массив», который является базовым для класса «FIFO». В классе должна быть определена чистая виртуальная функция для ввода суммы значений, переопределённая в классах «FIFO» и «Дек».
8. Создать абстрактный класс «Анкета», который является базовым для класса «Библиотечная карточка» и содержит ФИО человека. В классе должны быть определены чистые виртуальная функция для ввода и вывода значений, переопределённые в классах «Библиотечная карточка» и «Расширенная библиотечная карточка».
9. Создать абстрактный класс «Целое число», который является базовым для класса «Багаж». В классе должны быть определены чистые виртуальная функция для ввода и вывода значений, переопределённые в классах «Багаж» и «Дек».
10. Создать абстрактный класс «Анкета», который является базовым для класса «Студенческая группа» и содержит ФИО человека. В классе должны быть определены чистые виртуальная функция для ввода и вывода значений, переопределённые в классах «Студенческая группа» и расширенная студенческая группа.
11. Создать абстрактный класс «Абстрактный массив», который является базовым для класса «Массив». В классе должна быть определена чистая виртуальная функция для суммы элементов массива, переопределённая в классах «Массив» и «Записная книжка».
12. Создать абстрактный класс «Массив», который является базовым для класса «Матрица». В классе должна быть определена чистая виртуальная функция ввода элементов, переопределённая в классах «Матрица» и «Система алгебраических уравнений».
13. Создать абстрактный класс «Целое число», который является базовым для класса «Дата». В классе должна быть определена чистая виртуальная функция для вывода значений на экран, переопределённая в классах «Дата» и «Полная дата».

14. Создать абстрактный класс «Адрес», который является базовым для класса «Почтовая сортировка» и содержит почтовый адрес. В классе должны быть определены чистые виртуальная функция для ввода и вывода значений, переопределённые в классах «Почтовая сортировка» и «Расширенная почтовая сортировка».
15. Создать абстрактный класс «Анкета», который является базовым для класса «Гостиница» и содержит ФИО человека. В классе должны быть определены чистые виртуальная функция для ввода и вывода значений, переопределённые в классах «Гостиница» и «Услуги».
16. Создать абстрактный класс «Массив», который является базовым для класса «Матрица». В классе должна быть определена чистая виртуальная функция ввода элементов, переопределённая в классах «Матрица» и «Система алгебраических уравнений».
17. Создать абстрактный класс «Запись», который является базовым для класса «Массив» хранящая значение наименование товара. В классе должна быть определена чистая виртуальная функция для вывода значений на экран, переопределённая в классах «Товар» и «Расширенный товар».
18. Создать абстрактный класс «Анкета», который является базовым для класса «Студенческая группа» и содержит ФИО человека. В классе должны быть определены чистые виртуальная функция для ввода и вывода значений, переопределённые в классах «Студенческая группа» и расширенная студенческая группа.

Лабораторная работа № 8

Создание оконных приложений в среде Microsoft Visual Studio 2013

Цель занятия

1. Приобретение навыков создания приложений на основе Windows Forms.
2. Создать программу «Калькулятор», согласно заданию по варианту.

Постановка задачи

1. Для заданного варианта разработать алгоритм решения задачи на ЭВМ.
2. Составить программу на языке С++, которая работает с любым допустимым набором данных.

Содержание отчета

1. Постановка задачи для конкретного варианта.
2. Текст программы и результаты ее выполнения.

Методические указания

Создание нового проекта Windows Forms в системе Visual Studio 2013

Запустить Visual Studio.

1. В меню **«Файл»** выбрать команду **«Создать → Проект»**.
2. Откроется диалоговое окно **«Новый проект»** (рис. 1).

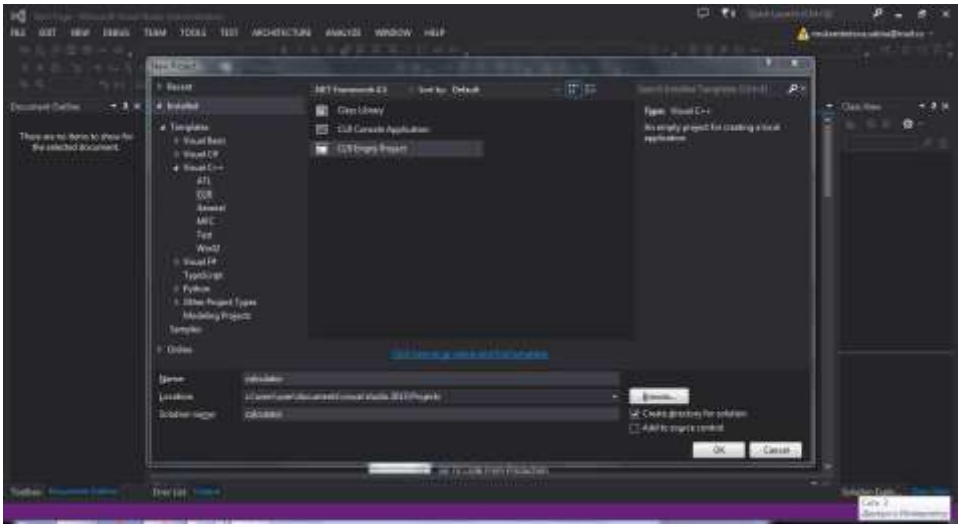


Рис. 1. Диалоговое окно "Новый проект"

3. В текстовом поле **«Имя»** задать имя проекта – **Calculator**.
4. В текстовом поле **«Расположение»** указать каталог, в котором требуется сохранить проект.
5. Нажать кнопку **ОК**.
6. Откроется конструктор Windows Forms и отобразится форма Form1 проекта (рис. 2).

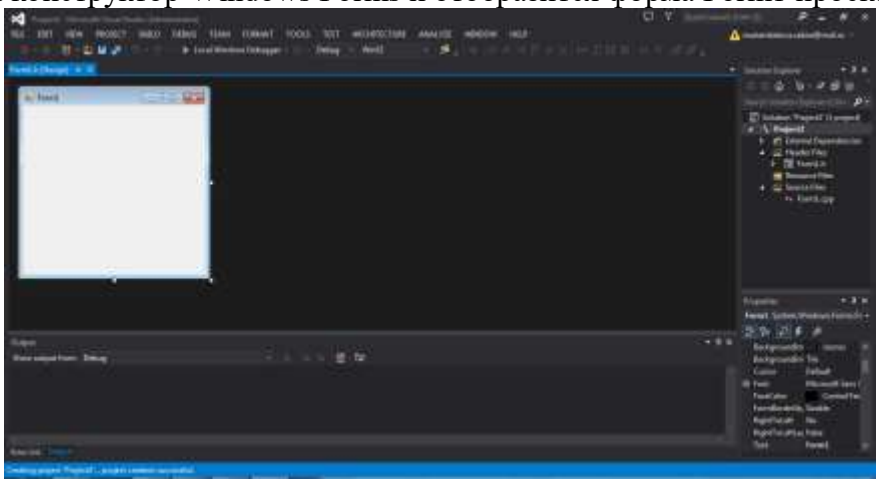


Рис. 2. Окно для разработки приложения Windows Form

Создание формы калькулятора и настройка сетки заполнения.

1. В конструкторе Windows Forms увеличить размер элемента управления **Form1**, перетаскивая маркер размера, находящийся в правом нижнем углу этого элемента управления, вниз и вправо. В правом нижнем углу Visual Studio находятся данные о размере и расположении элементов управления. Изменить размер элемента управления до тех пор, пока его
   ширина не будет равна 500, а высота – 400. Также размер форм можно установить в панели **Свойства**, в разделе **Size**.
2. В панели **Свойства** выбрать пункт **Text** и изменить значение с **«Form1»** на **«Калькулятор».**
3. В панели элементов открыть узел **«Контейнеры»**. Выбрать элемент управления **TableLayoutPanel** и перетащить его на рабочую поверхность.
4. Элемент управления **TableLayoutPanel** будет отображен в области конструктора с открытой панелью смарт-тега (рис 3).

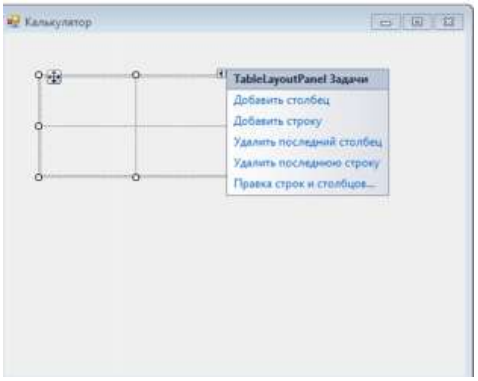


Рис. 3. Элемент TableLayoutPanel

5. В панели смарт-теге щелкнуть **«Правка строк и столбцов…»**. Откроется диалоговое окно **«Стили столбцов и строк»** (рис 4).
6. Нажать кнопку **«Добавить»** до тех пор, пока не будут отображаться пять столбцов. Выбрать все пять столбцов, и щелкнуть переключатель **«Процент»** в поле **«Тип размера»**. Установить в поле «Процент» значение 20. Все столбцы будут одинаковой ширины.
7. В раскрывающемся списке **«Показать»** выбрать **«Строки»**.
8. Нажать кнопку **«Добавить»** до тех пор, пока не будут отображаться пять строк. Выбрать все пять строк, и щелкнуть переключатель **«Процент»** в поле **«Тип размера»**. Установить в поле «Процент» значение 20. Все строки будут одинаковой ширины.
9. Нажать кнопку **ОК** для сохранения изменений и щелкнуть значок смарт-тега для закрытия панели.

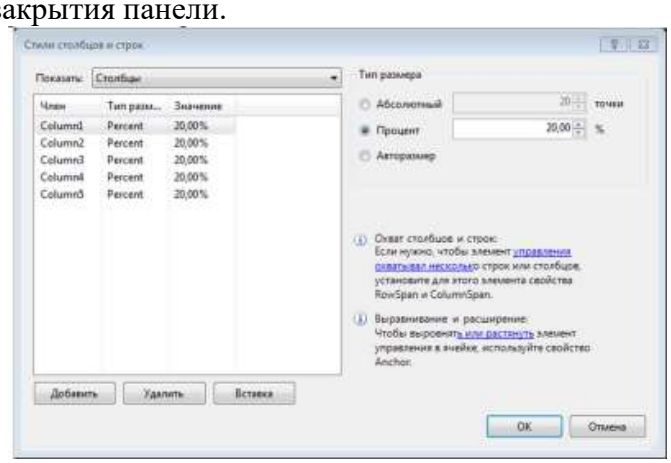


Рис. 4. Диалоговое окно «Стили столбцов и строк»

10. В окне **"Свойства"** изменить значение свойства **Dock** на **Fill** (рис 5).

Рис. 5. Поле Dock

Теперь разметка элемента управления настроена, можно заполнить форму **«Калькулятор»**, добавив в нее кнопки и окно.

Заполнение элементов сетки

1. Дважды щелкнуть значок элемента управления **TextBox** в панели элементов. Элемент управления **TextBox** будет помещен в первую ячейку элемента управления **TableLayoutPanel**.
2. В окне **"Свойства"** изменить значение свойства **ColumnSpan** элемента управления **TextBox** на 5.
3. Значение свойства **Dock** изменить на **Fill**.

4. Нажать кнопку смарт-тега поля и установить поле **Multiline**. Элемент управления **TextBox** переместится в середину строки и будет занимать все пять столбцов.
5. Изменить значение свойства **TextAlign** элемента управления **TextBox** на **Right**.
6. В окне **"Свойства"** развернуть узел **Font**(**Шрифт**). Установить **Size** равным 20, **Bold** равным **true** для элемента управления **TextBox**.
7. Свойство **ReadOnly** установить в значение **true**.
8. Выбрать элемент управления **TableLayoutPanel**.
9. Дважды щелкнуть значок элемента управления **Button** в панели элементов.
10. Элемент управления **Button** будет помещен в следующую открытую ячейку элемента управления **TableLayoutPanel**.
11. В панели элементов дважды щелкнуть значок **Button** еще четыре раза для заполнения второй строки элемента управления **TableLayoutPanel**.
12. Выбрать все пять элементов управления **Button**, щелкая их при удерживаемой клавише SHIFT. Нажать клавиши CTRL+C для копирования элементов управления в буфер обмена.
13. Трижды нажать клавиши CTRL+V для вставки копий элементов управления **Button** в оставшиеся строки элемента управления **TableLayoutPanel**.
14. Выбрать все 20 элементов управления **Button**, щелкая их при удерживаемой клавише SHIFT.
15. В окне **"Свойства"** изменить значение свойства **Dock** на **Fill**. Все элементы управления **Button** будут закреплены в ячейках, в которых они находятся.
16. В окне **"Свойства"** развернуть узел **Margin**. Присвоить свойству **All** значение 5.
17. Все элементы управления **Button** станут меньше, чтобы между ними образовались более широкие поля.
18. Выбрать **button10** и **button20**, затем нажать клавишу DELETE для удаления их из макета.
19. Выбрать **button5** и **button15** и изменить значения их свойства **RowSpan** на 2. Это будут кнопки **Сброс** и **=** для элемента управления Калькулятор.
20. Выбрать все 18 элементов управления **Button**, щелкая их при удерживаемой клавише SHIFT.
21. В окне **"Свойства"** развернуть узел **Font**. Присвоить свойству **Size** значение 15, а также свойству **Bold** – значение **true** для всех элементов управления **Button**. Панель приобретет вид, представленный на рис. 6.

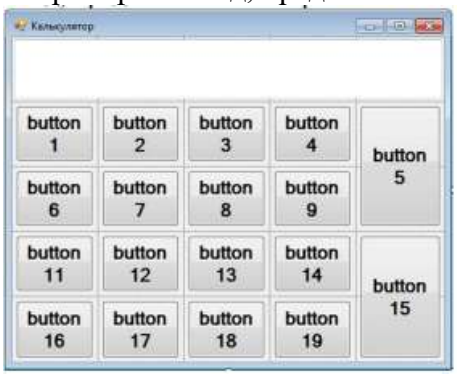


Рис. 6. Форма "Калькулятор"

Изменение параметров элементов управления.

1. В меню **«Вид»** выбрать **«Другие окна»**, затем **«Структура документа»**.

2.В окне **«Структура документа»** будет показано древовидное представление элемента управления «Калькулятор» и входящих в него элементов управления.

3. В окне **«Структура документа»** щелкнуть правой кнопкой мыши элемент управления **TextBox** и выбрать команду **«Переименовать»**. Изменить имя на **screen**.

4. В окне **«Структура документа»** переименовать элементы управления **Button** согласно следующему списку:
button1 на sevenButton
button2 на eightButton
button3 на nineButton
button4 на divisionButton
button5 на clearButton
button6 на fourButton
button7 на fiveButton
button8 на sixButton
button9 на multiplicationButton
button11 на oneButton
button12 на twoButton
button13 на threeButton
button14 на subtractionButton
button15 на equalsButton
button16 на changeSignButton
button17 на zeroButton
button18 на decimalButton
button19 на additionButton

1. В окне **«Структура документа»** и с помощью окон свойств изменить значения свойств **Text** элементов управления **Button** согласно следующему списку:
Изменить для элемента управления sevenButton свойство текста на **7**
Изменить для элемента управления eightButton свойство текста на **8**
Изменить для элемента управления nineButton свойство текста на **9**
Изменить для элемента управления divisionButton свойство текста на **/**
Изменить для элемента управления clearButton свойство текста на **CLR**
Изменить для элемента управления fourButton свойство текста на **4**
Изменить для элемента управления fiveButton свойство текста на **5**
Изменить для элемента управления sixButton свойство текста на **6**
Изменить для элемента управления multiplicationButton свойство текста на *****
Изменить для элемента управления oneButton свойство текста на **1**
Изменить для элемента управления twoButton свойство текста на **2**
Изменить для элемента управления threeButton свойство текста на **3**
Изменить для элемента управления subtractionButton свойство текста на **-**
Изменить для элемента управления equalsButton свойство текста на **=**
Изменить для элемента управления changeSignButton свойство текста на **+/-**
Изменить для элемента управления zeroButton свойство текста на **0**
Изменить для элемента управления decimalButton свойство текста на **.**
Изменить для элемента управления additionButton свойство текста на **+**

Создание формы Калькулятор завершено. Остается создать логическую часть калькулятора. Панель приобретет вид, представленный на рис. 7.

Рис. 7. Законченная форма "Калькулятор"

Создание логики калькулятора.

Функциональная часть калькулятора будет размешаться в отдельном классе **Calc**. Для его создания необходимо выполнить следующие действия:

1. В правой части, в панели **«Обозреватель решений»** щелкнуть правой кнопкой мыши на папку **«Заголовочные файлы»** и выбрать пункт **«Добавить → Класс»**.
2. В открывшемся окне выбрать пункт «**Класс C++**» и нажать кнопку **«Добавить»**.
3. Окно **«Мастер универсальных классов»** заполнить согласно рисунку 8, и нажать кнопку **«Готово»**

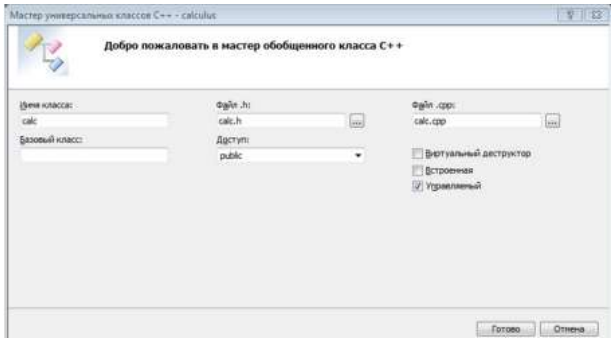


Рис. 8. Окно «Мастер универсальных классов»

4. Откроется окно заголовочного файла calc.h в котором будет представлена заготовка класса:

```
#pragma once

ref class calc
{
public:
calc(void);
};
```

4. Листинг класса calc, описывающий работу калькулятора представлен

ниже.

```
#pragma once
using namespace System;
ref class calc
{
public:
calc(void);
/*Описываются переменные для операндов и результата. Все переменные и функции класса должны быть статические, что бы не использовать  объекты класса.*/
```

```
static double x;
static double y;
static double rez;
/* Описываются строковые переменные для хранения значений, принимаемых из форм
Button и отображаемых в поле TextBox*. Тип  String ^ описан только в пространстве имен
System*/
static String ^String_x;
static String ^String_y;
static String ^String_scr;
/*Описывается переменная хранящая тип действия: 1 – деление; 2 –умножение; 3 –
вычитание; 4 – сложение.*/
static int operation;
/* Функция setValue устанавливает значение операндов в зависимости от нажатых кнопок
формы. Если переменная operation равно 0, значит, кнопка действия нажата не была и
заполняется первый операнд. Если operation неравно 0 – значит, была нажата одна из 4-х
кнопок действий и заполняется второй операнд. Значение в функцию передается в виде
текстовой строки. Первоначально принятое значение добавляется к существующему в
строковой переменной, после чего новое значение преобразуется в дробное значение при
помощи функции ToDouble, описанной в классе Convert */
static void setValue(String ^a)
{
if (operation==0) { String_x=String_x+a;
x=Convert::ToDouble(String_x);}
else { String_y=String_y+a; y=Convert::ToDouble(String_y);}
}
/* Функция setDecima устанавливает значение десятичной точки. Если переменная operation
равно 0, значит, кнопка действия нажата не была и заполняется первый операнд. Если
operation неравно 0 – значит, была нажата одна из 4-х кнопок действий и заполняется второй
операнд. Значение в функцию передается в виде текстовой строки.
Первоначально принятое значение добавляется к существующему в строковой
переменной*/
static void setDecimal(String ^a)
{
if (operation==0) { String_x=String_x+a;}
else { String_y=String_y+a;}
}
/* Функция setScrn подготавливает строковое значение для вывода на экран калькулятора.
В конечную строку записывается значение:
[операнд 1] [действие] [операнд 2].*/
static String ^ setScrn()
{
switch (operation)
{
case 1: String_scr = String_x + " / " + String_y; break;
case 2: String_scr = String_x + " * " + String_y; break;
case 3: String_scr = String_x + " - " + String_y; break;
case 4: String_scr = String_x + " + " + String_y; break;
default: String_scr = String_x;
}
return String_scr;
}
/* В функцию setOperation передается значение действия из одной из кнопок (/ * - +), которое
в свою очередь заносится в переменную operation.*/
```

```
static void setOperation(int a)
{
operation=a;
}
/* Функция clr( ) сбрасывает значения операндов, строк операндов, конечной строки вывода,
а также устанавливает в 0 значение переменной operation;*/
static String ^ clr()
{
String_x=""; String_y="" ; String_scr=""; operation=0;
return String_scr;
}
/* Функция changeSign меняет знак операндов в зависимости от нажатых кнопок формы.
Если переменная operation равно 0, значит, кнопка действия нажата не была и меняется
знак для первого операнда. Если operation не равно 0 – значит, была нажата одна из 4-х
кнопок действий и меняется знак для второго операнда. Значение операнда умножается
на -1, после чего операнд переводится в строковой вид для вывода на экран. */
static void changeSign()
{
if (operation==0) { x=x*(-1); String_x = x.ToString();}
else { y=y*(-1); String_y = y.ToString();}
}
/* Функция setEqual высчитывает результирующее значение. Для этого анализируется
значение переменной operation, и в зависимости от ее значения высчитывается нужный
результат. Далее полученное значение преобразуется в текстовую форму и заносится в
строку вывода String_scr. В переменную x заносится результат, в строку String_x значение
строки String_scr для продолжения дальнейшего расчета. Переменные y и String_y –
обнуляются. Строка String_scr возвращается в точку вызова.*/
static String ^ setEqual()
{
switch (operation)
{
case 1: rez = x/y; break;
case 2: rez = x*y; break;
case 3: rez = x-y; break;
case 4: rez = x+y; break;
default: String_scr = String_x;
}
String_scr = rez.ToString();
String_x = String_scr;
String_y = "0";
x=rez;
y=0;
return String_scr;
}
};
```

Создание обработчиков событий

Для полноценной работы калькулятора, необходимо подключить созданный класс к форме **«Калькулятор»**. Для этого открыть конструктор формы и нажать клавишу F7. Откроется редактор кода. В самом начале после строки #pragma once написать строчку #include "calc.h"

InitializeComponent(void) и в разделе //screen дописать строку this->scrn->TabIndex = 0;

таким образом, при запуске калькулятора на экране будет отображаться 0.
Кнопки в форме **«Калькулятор»** имеют обработчики событий, которые можно использовать для реализации логики калькулятора. Конструктор Windows Forms позволяет реализовать обработчики событий для всех кнопок одним двойным щелчком. Для создания обработчика событий кнопки 1, необходимо выполнить следующие действия:
1. В конструкторе форм дважды щелкнуть на кнопке 1. Откроется редактор кода, в котором появится обработчик событий для этой кнопки:

```
private: System::Void oneButton_Click(System::Object^ sender,
System::EventArgs^ e) {
}
```

2. Дополнить обработчик следующими строчками кода:

```
private: System::Void oneButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setValue("1");
screen->Text = calc::setScrn();
}
```

Строка calc::setValue("1"); устанавливает значение первого или второго операнда.
Строка screen->Text = calc::setScrn(); отображает на элементе screen значение операндов.
3. Строки кода для остальных кнопок приведены ниже. Строки кода в них эквиваленты приведённым:

```
private: System::Void twoButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setValue("2");
screen->Text = calc::setScrn();
}
private: System::Void threeButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setValue("3");
screen->Text = calc::setScrn();
}
private: System::Void fourButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setValue("4");
screen->Text = calc::setScrn();
}
private: System::Void fiveButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setValue("5");
}
private: System::Void sixButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setValue("6");
screen->Text = calc::setScrn();
}
private: System::Void sevenButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setValue("7");
screen->Text = calc::setScrn();
}
private: System::Void eightButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setValue("8");
screen->Text = calc::setScrn();
```

```
}
private: System::Void nineButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setValue("9");
screen->Text = calc::setScrn();
}
private: System::Void zeroButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setValue("0");
screen->Text = calc::setScrn();
}
private: System::Void clearButton_Click(System::Object^ sender,
System::EventArgs^ e) {
screen->Text = calc::clr();
}
private: System::Void divisionButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setOperation(1);
screen->Text = calc::setScrn();
}
private: System::Void multiplicationButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setOperation(2);
screen->Text = calc::setScrn();
}
private: System::Void subtractionButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setOperation(3);
screen->Text = calc::setScrn();
}
private: System::Void additionButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setOperation(4);
screen->Text = calc::setScrn();
}
private: System::Void decimalButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::setDecimal(",");
}
private: System::Void equalButton_Click(System::Object^ sender,
System::EventArgs^ e) {
screen->Text = calc::setEqual();
}
private: System::Void changeSignButton_Click(System::Object^ sender,
System::EventArgs^ e) {
calc::changeSign();
screen->Text = calc::setScrn();
}
```

4. Скомпилировать проект. Если выдаются сообщения об ошибках, исправить их.

Варианты заданий.

1. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для расчета синуса и косинуса значения.

2. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для расчета тангенса и котангенса значения.
3. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для возведения значения в квадрат и вычисления квадратного корня.
4. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для возведения значения в степень N и вычисления корня N-ой степени.
5. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для возведения экспоненты в степень значения и вычисления натурального логарифма от значения.
6. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для вычисления десятичного логарифма от значения и обыкновенного логарифма по степени **n**.
7. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для расчета гиперболического синуса и гиперболического косинуса значения.
8. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для перевода значения в процент и расчета обратного значения (1/x).
9. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для расчета 2^x и 10^x.
10. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для перевода в двоичную систему счисления и расчета остатка от деления двух значений.
11. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для перевода в восьмеричную систему счисления и округления числа до большего целого.
12. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для перевода в шестнадцатеричную систему счисления и округления числа до меньшего целого.
13. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для расчета факториала значения и модуля значения.
14. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для расчета арксинуса и арккосинуса значения.
15. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для расчета арктангенса и арккотангенса значения.
16. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для расчета гиперболического арксинуса и гиперболического арккосинуса значения.
17. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для расчета синуса и косинуса значения.
18. Создать приложение калькулятор, согласно примеру. Дополнить приложение кнопками для перевода в двоичную систему счисления и расчета остатка от деления двух значений.

Список литератур

1. Павловская Т. А. С/С++. Программирование на языке высокого уровня: Учебник для вузов, Санкт-Петербург: Питер, 2013 г. , 464 с.

2. Культин Н. Б. К90 С/С++ в задачах и примерах: 2-еизд., перераб. и доп. —. СПб.: БХВ-Петербург,2009. — 368 с .
3. Фридман, А. Л.Язык программирования Си++ [Текст] : курс лекций : учебное пособие / А. Л. Фридман ; Интернет-Университет информационных технологий. - Изд. 2-е, испр. - Москва : Интернет-Университет информационных технологий, 2010. - 261, [1] с.
4. Шилдт Г.: С++базовый курс. - СПб.: Питер, 2014. - 512 с
5. И.А.Волкова, А.В.Иванов, Л.Е.Карпов. Основы объектно-ориентированного программирования. Язык программирования С++. 2011.
http://cmcstuff.esyr.org/vmkbotva-r15/2%20курс/4%20Семестр/СП/Учебники/cpp.base.oop.pdf

Web-сайты

1. http://libr.aues.kz/facultet/frts/kaf_ie/32/umm/ect_58(1).htm

2. http://www.vlsu.ru/fileadmin/Kadry_dlja_regiona/7/2014_dec/7-2-4-01_2014_Programirovanie.pdf