Лабораторная работа №03. Перегрузка функций. Шаблоны функций

1 Цель и порядок работы

Цель работы – ознакомиться с возможностью перегрузки функций и научиться применять полученные знания на практике. Научиться использовать шаблоны функции и функции с переменным количеством параметров.

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя, согласно своему варианту;
- написать программу и отладить ее на ЭВМ.

2 Краткая теория

Каждая программа на C++ – это совокупность функций, каждая из которых должна быть определена или описана до её использования в конкретном модуле программы. Рассмотрим более сложные примеры использования функций.

2.1 Перегрузка функций

Перегрузкой функций называется использование нескольких функций с одним и тем же именем, но с различными списками параметров. Перегруженные функции должны отличаться друг от друга либо типом хотя бы одного параметра, либо количеством параметров, либо и тем и другим одновременно. Перегрузка является видом полиморфизма и применяется в тех случаях, когда одно и то же по смыслу действие реализуется по-разному для различных типов или структур данных. Компилятор сам определяет, какой именно вариант функции вызвать, руководствуясь списком аргументов.

Если же алгоритм не зависит от типа данных, лучше реализовать его не в виде группы перегруженных функций для различных типов, а в виде шаблона функции. В этом случае компилятор сам сгенерирует текст функции для конкретных типов данных, с которыми выполняется вызов, и программисту не придется поддерживать несколько практически одинаковых функций.

Небольшие перегруженные функции удобно применять при отладке программ.

Допустим, вам требуется промежуточная печать различного вида: в одном месте требуется выводить на экран структуру, в другом – пару целых величин с пояснениями или вещественный массив. Поэтому проще сразу оформить печать в виде функций, например таких:

```
void print(char *str, const int i, const int j)
{
   cout << str << '|' << oct << setw(4) << i << '|' << setw(4) << j
      << '|' << endl;
}

void print(float array[], const int n)
{
   cout << "Массив:" << endl;
   cout.setf(ios::fixed);
   cout.precision(2);
   for (int i = 0; i < n; i++)
   {
```

```
        cout << array[i] << " ";
        if ((i + 1) % 4 == 0) cout << endl;
    }
    cout << endl;
}

void print(Man m)
{
    cout.setf(ios::fixed);
    cout.precision(2);
    cout << setw(40) << m.name << ' ' << m.birthday << ' '
       << m.pay << endl;
}
```

В первой из этих функций на экран выводятся строка и два целых числа в восьмеричной форме, разделенных вертикальными черточками для читаемости. Под каждое число отводится по 4 позиции (действие манипулятора `setw` распространяется только на ближайшее выводимое поле).

Во второй функции для вывода вещественных значений по четыре числа на строке задается вид вывода с фиксированной точкой и точностью в два десятичных знака после запятой. Для этого используются методы установки флагов `setf`, установки точности `precision` и константа `fixed`, определенная в классе `ios`. Точность касается только вещественных чисел, ее действие продолжается до следующей установки. Третья функция выводит поля знакомой нам по шестому семинару структуры так, чтобы они не склеивались между собой. Манипулятор `setw` устанавливает ширину следующего за ним поля. Это приведет к тому, что фамилии будут выведены с отступом от края экрана. Вызов этих функций в программе может выглядеть, например, так:

```
print("После цикла ", i, j);
print(a, n);
print(m);
```

По имени функции сразу понятно, что она делает, кроме того, при необходимости вызов функции легче закомментировать или перенести в другое место, чем группу операторов печати. Конечно, промежуточная печать – не единственный метод отладки, но зато универсальный, потому что отладчик не всегда доступен.

При написании перегруженных функций основное внимание следует обращать на то, чтобы в процессе поиска нужного варианта функции по ее вызову не возникало неоднозначности.

Неоднозначность может возникнуть по нескольким причинам. Во-первых, из-за преобразований типов, которые компилятор выполняет по умолчанию. Их смысл сводится к тому, что более короткие типы преобразуются в более длинные. Если соответствие между формальными параметрами и аргументами функции на одном и том же этапе может быть получено более чем одним способом, вызов считается неоднозначным и выдается сообщение об ошибке.

Неоднозначность может также возникнуть из-за параметров по умолчанию и ссылок. Рассмотрим создание перегруженных функций на примере.

Пример 12.1. Перегрузка функций

Написать программу, которая находит расстояние между двумя точками. Координаты могут задаваться как в декартовой, так и в полярной системе координат.

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

```
#include <math.h>

using namespace std;

//объявим две структуры для хранения информации о координатах точек
//в декартовой системе
struct cartesian{
   double x, y;
};

//и в по системе координат
struct polar{
   double r, pi;
};

//теперь определим перегружаемую функцию,
//принимающую координаты двух точек через полярные координаты
double len(polar a, polar b)
{
   cout << "Считаем расстояние через полярные координаты" << endl;
   return sqrt(pow(a.r, 2) + pow(b.r, 2) - 2*a.r*b.r*cos(a.pi - b.pi));
}
//а затем принимающую координаты двух точек через декартовы координаты
double len(cartesian x, cartesian y)
{
   cout << "Считаем расстояние через декартовы координаты" << endl;
   return sqrt(pow(y.x - x.x, 2)+pow(y.y - x.y, 2));
}
//будем считать, что при передаче четырех параметров
//передаются декартовы координаты двух точек
double len(double x1, double y1, double x2, double y2)
{
   cout << "Считаем расстояние через декартовы \
      координаты с 4-мя параметрами" << endl;
   return sqrt(pow(x2 - x1, 2)+pow(y2 - y1, 2));
}

void main(int argc, char* argv[])
{
   setlocale(LC_ALL, "Russian");

   const double PI = 3.14159;

   cartesian a = {3, 0},
             b = {1, 1};

   polar c = {1.41, PI/4},
         d = {3.1, 0.95};

   double x1 = 1.4, y1 = 2.5,
          x2 = 2.1, y2 = 3.7;

   cout << len(a, b) << endl;
   cout << len(c, d) << endl;
   cout << len(x1, y1, x2, y2) << endl;
}
```

Результат работы:

```
Считаем расстояние через декартовы координаты
2.23607
Считаем расстояние через полярные координаты
1.7246
Считаем расстояние через декартовы координаты с 4-мя параметрами
1.38924
```

2.2 Шаблоны функций

Цель введения шаблонов функций – автоматизация создания функций, которые могут обрабатывать разнотипные данные. В отличие от механизма перегрузки, когда для каждой сигнатуры определяется своя функция, шаблон семейства функций определяется один раз. Шаблон располагается перед main.

```
template <class ttype>

ttype имя_функции (список_формальных_параметров)
{
   тело функции
}
```

Здесь ttype – любой корректный идентификатор, который автоматически заменяется компилятором на любой стандартный тип.

Пример 12.2. Шаблон функции для нахождения максимального элемента массива

```
#include "stdafx.h"
#include <iostream>
#include <math.h>

using namespace std;

template <class array_type>
array_type max(array_type *a, const int N)
{
   array_type m = a[0];

   for (int i = 1; i < N; i++)
       if (a[i] > m)
       {
           m = a[i];
       }
   return m;
}

int main(int argc, char* argv[])
{
   setlocale(LC_ALL, "Russian");

   double a[] = {2.5, 8.3, 6};
   int b[] = {3, 5, -1, 2};
   char c[] = {'A', 'b', 'Z', 'r'};

   cout << max(a, sizeof(a)/sizeof(a[0])) << endl;
   cout << max(b, sizeof(b)/sizeof(b[0])) << endl;
```

```
    cout << max(c, sizeof(c)/sizeof(c[0])) << endl;

    return 0;
}
```

При использовании шаблонов уже нет необходимости готовить заранее все варианты функций с перегруженным именем. Компилятор автоматически, анализируя вызовы функций в тексте программы, формирует необходимые определения именно для таких типов параметров, которые использованы в обращениях. Дальнейшая обработка выполняется так же, как и для перегруженных функций.

Основные свойства параметров шаблона:

1. Имена параметров шаблона должны быть уникальными всем определении шаблона.

2. Список параметров шаблона функции не может быть пустым, так как при этом теряется возможность параметризации и шаблон функции становиться обычным определением конкретной функции.

3. В списке параметров шаблона функции может быть несколько параметров. Каждый из них должен начинаться со служебного слова **class**.

Допустимый заголовок шаблона:

```
template <class type1, class type2>
```

Соответственно, неверен заголовок:

```
template <class type1, type2, type3>
```

4. Недопустимо использовать в заголовке шаблона параметры с одинаковыми именами, т.е. ошибочен такой заголовок:

```
template <class type1, class type1, class type1>
```

2.3 Функции с переменным количеством параметров

В C++ допустимы функции, у которых количество параметров при компиляции определения функции не определено. Кроме того, могут быть неизвестными и типы параметров. Количество и типы параметров становятся известными только в момент вызова функции, когда явно задан список фактических параметров. При определении и описании таких функций спецификация формальных параметров заканчивается многоточием:

```
тип имя (список_явных_параметров, ...);
```

Каждая функция с переменным списком параметров должна иметь один из двух механизмов определения их количества и типов.

Первый подход предполагает добавление в конец списка реально использованных, необязательных, фактических параметров специального параметра-индикатора с уникальным значением которое будет сигнализировать об окончании списка. В теле функции параметры перебираются, и их значение сравнивается с заранее известным концевым признаком.

Второй подход предусматривает передачу в функцию значения реального количества используемых фактических параметров, которые передаются с помощью одного из обязательных параметров.

В обоих случаях переход от одного фактического параметра к другому выполняется с помощью указателей (с использованием адресной арифметики).

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

using namespace std;

long summa(int k, ...)
{
   int *pik = &k;

   long total = 0;

   for (; k; k--)
       total += *(++pik);

   return total;
}

int main(int argc, char* argv[])
{
   setlocale(LC_ALL, "Russian");

   cout << "\n summa(2, 6, 4) = " << summa(2, 6, 4);
   cout << "\n summa(6, 1, 2, 3, 4, 5, 6) =" <<
      summa(6, 1, 2, 3, 4, 5, 6);

   return 0;
}
```

Результат

```
summa(2, 6, 4)=10
summa(6, 1, 2, 3, 4, 5, 6)=21
```

Особенность этой программы, что указатель `pik` может работать только с целочисленными фактическими параметрами.

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

using namespace std;

double prod(double arg, ...)
{
   double result = 1.0;

   double *prt = &arg;

   if (*prt == 0.0)
       return 0.0;

   for( ; *prt; prt++)
       result *= *prt;

   return result;
}
```

```
int main(int argc, char* argv[])
{
   setlocale(LC_ALL, "Russian");

   cout <<"\n prod(2e0,4e0,3e0,0e0) = " << prod(2e0,4e0,3e0,0e0);
   cout <<"\n prod(1.5,2.0,3.0,0.0) = " << prod(1.5,2.0,3.0,0.0);
   cout <<"\n prod(1.4,3.0,0.0,16.0,84.3,0.0) = " <<
       prod(1.4,3.0,0.0,16.0,84.3,0.0);
   cout << "\n prod(0e0) =" << prod(0e0);

   return 0;
}
```

Результат

```
prod(2e0,4e0,3e0,0e0) = 24
prod(1.5,2.0,3.0,0.0) = 9
prod(1.4,3.0,0.0,16.0,84.3,0.0) = 4.2
prod(0e0) = 0
```

Признаком окончания списка фактических параметров служит параметр с нулевым значением. Чтобы функция с переменным количеством параметров воспринимала параметры различных типов, то для однотипных параметров необходимо передавать информацию функции с помощью обязательного дополнительного параметра.

2.4 Пример решения задачи

Пример Нахождение расстояния между точками

Задание. Написать программу, которая находит расстояние между двумя точками. Координаты могут задаваться как в декартовой, так и в полярной системе координат. Предусмотреть функции перевода из декартовой системы в полярную и обратно.

Алгоритм решения:

- Задать или ввести с клавиатуры координаты точек.
- Для пары точек определить расстояние по одной из формул:
  - для декартовой системы: $l = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$
  - для полярной системы: $l = \sqrt{\rho_1^2 + \rho_0^2 - 2\rho_1\rho_0 \cdot \cos(\varphi_1 - \varphi_0)}$
- Вывести результаты на экран.

Определение требуемых типов данных.

Для удобства хранения информации о координате точки в одной из систем координат удобно воспользоваться структурами, хранящими пары координат одной и той же точки. Для простоты назовем их cartesian для декартовой, и polar для полярной системы координат.

Разбиение на подзадачи.

В результате описанного алгоритма и задания определяем спецификации нужных нам функций. Для удобства использования воспользуемся перегрузкой:

1. double len(cartesian a, cartesian b) – принимает две координаты в декартовой системе и возвращает длину.

2. double len(polar a, polar b) – принимает две координаты в полярной системе и возвращает длину.

3. double len(cartesian a, polar b) – принимает первую координату в декартовой системе, а вторую в полярной, и возвращает длину.

4. double len(polar a, cartesian b) – принимает первую координату в полярной системе, а вторую в декартовой, и возвращает длину.

5. polar cartesian_to_polar(cartesian x) – переводит координаты точки из в декартовой системы в полярную.

6. cartesian polar_to_cartesian(polar x) – переводит координаты точки из в полярной системы в декартову.

Разбиение на модули.

Исходя из логики работы программы, а также, возможного дальнейшего применения полученных результатов естественным будет разбить ее на два файла: PRG-Lab13.cpp – содержащий точку входа в программу, а также основные вызовы функции и взаимодействие с пользователем; points.cpp – содержащий реализации вышеперечисленных функций; а также points.h – содержащий интерфейсы данных функций и типы данных, и служащий для подключения возможностей работы с точками в различных системах координат.

Файл points.h

```
// points.h : Содержит определения основных типов и прототипы функций.
//
#ifndef POINTS_H
#define POINTS_H

//Объявления типов
//Точка в декартовой системе координат
struct cartesian{
   double x, y;
};

//Точка в полярной системе координат
struct polar{
   double r, pi;
};

//Прототипы функций
double len(polar a, polar b);
double len(cartesian a, cartesian b);
double len(cartesian a, polar b);
double len(polar a, cartesian b);

polar cartesian_to_polar(cartesian x);
cartesian polar_to_cartesian(polar x);
#endif
```

Файл points.cpp

Примечание. Обратите внимание на повторное использование вызовов уже существующих функций. Данная техника позволяет добиться быстрого и удобного внесения корректив (хотя и за счет несущественной потери в производительности). Например, при наличии ошибки, скажем, при неверно запрограммированной формуле, ее достаточно исправить лишь в одном месте, а не искать по всему тексту модуля.

```cpp
// points.cpp : Содержит определения функций
//

#include "stdafx.h"
#include "points.h"
#include <math.h>


double len(polar a, polar b)
{
   return sqrt( pow(a.r, 2) + pow(b.r, 2) - 2*a.r*b.r*cos(a.pi - b.pi) );
}

double len(cartesian a, cartesian b)
{
   return sqrt( pow(a.x - b.x, 2) + pow(a.y - b.y, 2) );
}

double len(cartesian a, polar b)
{
   cartesian c;

   c = polar_to_cartesian(b);

   return len(a, c);
}

double len(polar a, cartesian b)
{
   return len(b, a);
}

polar cartesian_to_polar(cartesian x)
{
   polar y;

   y.r = sqrt(pow(x.x, 2) + pow(x.x, 2));

   if (y.r == 0)
       y.pi = 0;
   else
       y.pi = asin(x.y/y.r);

   return y;
}

cartesian polar_to_cartesian(polar x)
{
   cartesian y;

   y.x = x.r*cos(x.pi);
   y.y = x.r*sin(x.pi);

   return y;

}
```

Файл PRG-Lab12.cpp

```
// PRG-Lab12.cpp : Основной файл проекта, содержит точку входа в
программу
//

#include "stdafx.h"
#include <iostream>
#include "points.h"

using namespace std;

void main(int argc, char* argv[])
{
   setlocale(LC_ALL, "Russian");

   const double PI = 3.14159;

   cartesian a = {3, 0},
             b = {1, 1};

   polar c = {1.41, PI/4},
         d = {3.1, 0.95};

   //вызовы перегруженных функций
   //для точек в различных системах координат
   //декартова - декартова
   cout << len(a, b) << endl;
   //полярная - полярная
   cout << len(c, d) << endl;
   //декартова - полярная
   cout << len(a, d) << endl;
   //полярная - декартова
   cout << len(c, b) << endl;
}
```

Таким образом мы получили собственную мини библиотеку, которую можно расширять в дальнейшем и использовать в других проектах.

3 Контрольные вопросы

1. Что такое перегрузка функции?
2. Для чего применяется перегрузка функций?
3. Как определяется, какая из версий перегружаемых функций будет вызвана?
4. Каким образом описывается шаблон семейства функций?
5. Как описываются функции с переменным количеством параметром?

4 Задание

1. Создать новый проект.
2. Добавить к проекту файл исходным кодом (.h и .cpp).
3. Вынести в отдельный модуль функции и типы данных.

4. Написать программу в соответствии с вариантом задания из пункта 5.1 и 5.2, задав начальные значения при объявлении переменных.
5. Отладить и протестировать программу.
6. Создать новый проект.
7. Подключить модуль из предыдущего задания в новый проект.
8. Написать программу в соответствии с вариантом задания из пункта 5, введя начальные значения переменных с клавиатуры.
9. Отладить и протестировать программу.

5 Варианты заданий

5.1 Перегрузка функций

1. Определить функцию, возвращающую максимальное из нескольких чисел. Выполнить перегрузку функции для следующих типов параметров:
    1.1. Два параметра типа int.
    1.2. Три параметра типа int.
    1.3. Два параметра типа float.
    1.4. Три параметра типа double.
2. Определить функцию, возвращающую количество дней до конца месяца. Выполнить перегрузку функции для следующих типов параметров:
    2.1. Структура «дата» (год, месяц, день).
    2.2. Три целочисленных параметра: год, месяц, день.
    2.3. Два целочисленных параметра: месяц, день (считать передаваемые числа датой текущего года).
3. Определить функцию, возвращающую НОК нескольких чисел. Выполнить перегрузку функции для следующих типов параметров:
    3.1. Два параметра типа int.
    3.2. Два параметра типа long.
    3.3. Два параметра типа float.
    3.4. Два параметра типа double.
4. Определить функцию, находящую максимальный элемент массива. Выполнить перегрузку функции для следующих типов параметров:
    4.1. Одномерный массив типа int размерностью N.
    4.2. Одномерный массив типа float размерностью N.
    4.3. Одномерный массив типа double размерностью N.
5. Определить функцию, возвращающую минимальное из нескольких чисел. Выполнить перегрузку функции для следующих типов параметров:
    5.1. Три параметра типа int.
    5.2. Четыре параметра типа int.
    5.3. Три параметра типа float.
    5.4. Два параметра типа double.
6. Определить функцию, возвращающую количество недель с начала года. Выполнить перегрузку функции для следующих типов параметров:
    6.1. Структура «дата» (год, месяц, день).
    6.2. Три целочисленных параметра: год, месяц, день.
    6.3. Два целочисленных параметра: месяц, день (считать передаваемые числа датой текущего года).
7. Определить функцию, возвращающую количество недель до конца года. Выполнить перегрузку функции для следующих типов параметров:
    7.1. Структура «дата» (год, месяц, день).
    7.2. Три целочисленных параметра: год, месяц, день.

7.3. Два целочисленных параметра: месяц, день (считать передаваемые числа датой текущего года).

8. Определить функцию, возвращающую НОД нескольких чисел. Выполнить перегрузку функции для следующих типов параметров:
   8.1. Два параметра типа int.
   8.2. Два параметра типа long.
   8.3. Два параметра типа float.
   8.4. Два параметра типа double.
9. Определить функцию, возвращающую количество минут до окончания суток. Выполнить перегрузку функции для следующих типов параметров:
   9.1. Структура «время» (часы, минуты, секунды).
   9.2. Три целочисленных параметра: часы, минуты, секунды.
   9.3. Два целочисленных параметра: часы, минуты.
10. Определить функцию, возвращающую предыдущую минуту. Выполнить перегрузку функции для следующих типов параметров:
   10.1. Структура «время» (часы, минуты, секунды).
   10.2. Три целочисленных параметра: часы, минуты, секунды.
   10.3. Два целочисленных параметра: часы, минуты.
11. Определить функцию, находящую сумму элементов массива. Выполнить перегрузку функции для следующих типов параметров:
   11.1. Одномерный массив типа int размерностью N.
   11.2. Одномерный массив типа float размерностью N.
   11.3. Одномерный массив типа double размерностью N.
12. Определить функцию, проверяющую верна ли дата. Выполнить перегрузку функции для следующих типов параметров:
   12.1. Структура «дата» (год, месяц, день).
   12.2. Три целочисленных параметра: год, месяц, день.
   12.3. Два целочисленных параметра: месяц, день (считать передаваемые числа датой текущего года).
13. Определить функцию, возвращающую расстояние между точками числа. Выполнить перегрузку функции для следующих типов параметров:
   13.1. Два параметра типа структура «точка» (координаты x, y).
   13.2. Четыре параметра типа float.
   13.3. Четыре параметра типа double.
14. Определить функцию, возвращающую следующую минуту. Выполнить перегрузку функции для следующих типов параметров:
   14.1. Структура «время» (часы, минуты, секунды).
   14.2. Три целочисленных параметра: часы, минуты, секунды.
   14.3. Два целочисленных параметра: часы, минуты.
15. Определить функцию, находящую произведение ненулевых элементов массива. Выполнить перегрузку функции для следующих типов параметров:
   15.1. Одномерный массив типа int размерностью N.
   15.2. Одномерный массив типа float размерностью N.
   15.3. Одномерный массив типа double размерностью N.
16. Определить функцию, возвращающую прошедшее время в минутах (считать, что разница между передаваемыми значениями не превышает 24 часа). Выполнить перегрузку функции для следующих типов параметров:
   16.1. Два параметра типа структура «время» (часы, минуты, секунды).
   16.2. Шесть целочисленных параметра: часы, минуты, секунды.
   16.3. Четыре целочисленных параметра: часы, минуты.
17. Определить функцию, возвращающую минимальное из нескольких чисел. Выполнить перегрузку функции для следующих типов параметров:

17.1. Два параметра типа int.
17.2. Три параметра типа int.
17.3. Два параметра типа float.
17.4. Три параметра типа float.
17.5. Три параметра типа double.

18. Определить функцию, возвращающую количество дней с начала года. Выполнить перегрузку функции для следующих типов параметров:
18.1. Структура «дата» (год, месяц, день).
18.2. Три целочисленных параметра: год, месяц, день.
18.3. Два целочисленных параметра: месяц, день (считать передаваемые числа датой текущего года).

19. Определить функцию, возвращающую n-ю степень числа x. Выполнить перегрузку функции для следующих типов параметров:
19.1. Два параметра: x и n – оба типа int.
19.2. Два параметра: x и n – оба типа float.
19.3. Два параметра: x – типа float, и n – типа int.

20. Определить функцию, находящую минимальный элемент массива. Выполнить перегрузку функции для следующих типов параметров:
20.1. Одномерный массив типа int размерностью N.
20.2. Одномерный массив типа float размерностью N.
20.3. Одномерный массив типа double размерностью N.

21. Определить функцию, возвращающую количество минут с начала суток. Выполнить перегрузку функции для следующих типов параметров:
21.1. Структура «время» (часы, минуты, секунды).
21.2. Три целочисленных параметра: часы, минуты, секунды.
21.3. Два целочисленных параметра: часы, минуты.

22. Определить функцию, возвращающую среднеарифметическое нескольких чисел. Выполнить перегрузку функции для следующих типов параметров:
22.1. Два параметра типа int.
22.2. Три параметра типа int.
22.3. Три параметра типа float.
22.4. Два параметра типа double.

23. Определить функцию, возвращающую день недели (иметь в виду, что 1 января 1-го года нашей эры было понедельником). Выполнить перегрузку функции для следующих типов параметров:
23.1. Структура «дата» (год, месяц, день).
23.2. Три целочисленных параметра: год, месяц, день.
23.3. Два целочисленных параметра: месяц, день (считать передаваемые числа датой текущего года).

24. Определить функцию, находящую среднеарифметическое элементов массива. Выполнить перегрузку функции для следующих типов параметров:
24.1. Одномерный массив типа int размерностью N.
24.2. Одномерный массив типа float размерностью N.
24.3. Одномерный массив типа double размерностью N.

25. Определить функцию, возвращающую количество дней до конца года. Выполнить перегрузку функции для следующих типов параметров:
25.1. Структура «дата» (год, месяц, день).
25.2. Три целочисленных параметра: год, месяц, день.
25.3. Два целочисленных параметра: месяц, день (считать передаваемые числа датой текущего года).

5.2 Шаблоны функций

На основе задание 5.1 построить шаблон семейства функций. Вариант выбирать путем добавления 3 к номеру студента в списке группы.