

Тема 2. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ФУНКЦИЙ

Цель занятия: овладеть навыками алгоритмизации и программирования задач с использованием функций пользователя; получить практические навыки написания функций с умалчиваемыми значениями параметров, подставляемых (inline) функций, функций с переменным числом параметров, перегруженных функций; использования шаблона функции.

Методические указания

При подготовке к занятию необходимо изучить: правила записи и способы передачи параметров различных типов функций и способы обращения к ним; правила записи и порядок выполнения программ, использующих такие функции.

Теоретические сведения

Функции с начальными (умалчиваемыми) значениями параметров

В определении функции может содержаться начальное (умалчиваемое) значение параметра. Это значение используется, если при вызове функции соответствующий параметр опущен. Все параметры, описанные справа от такого параметра также должны быть умалчиваемыми.

Задание 2.1. Создайте функцию *print*:

```
void print(double value=10, char*name2="Иванов ", char*name1="Студент: ", char*name3="Средний бал ")
{cout<<"\n"<<name1 <<"\t"<<name2<<"\t"<<name3<<"\t"<<value;}
```

Организуйте вызовы функции так, чтобы получить следующие выводы:

- Студент: Иванов Средний бал 10
- Студент: Иванов Средний бал 8
- Студент: Сидоров Средний бал 6
- аспирант Колесник Средний бал 11
- количество слов в статье 2000

Подставляемые (inline) функции

Функция, определенная с использованием служебного слова *inline*, называется подставляемой или встраиваемой.

Пример 2.1. Подставляемая функция возвращает расстояние от точки с координатами (x1,y1) до точки с координатами (x2,y2) (по умолчанию центр координат).

```
inline float Line(float x1, float y1, float x2=0, float y2=0)
{return sqrt(pow(x1-x2,2)+pow(y1-y2,2));}
```

Спецификатор *inline* определяет для функции так называемое внутреннее связывание (компилятор вместо вызова функции подставляет команды ее кода). *При этом может увеличиваться размер программы, но исключаются затраты на передачу управления к вызываемой функции и возврата из нее.* Подставляемыми не могут быть:

- рекурсивные функции;
- функции, у которых вызов размещается до ее определения;
- функции, которые вызываются более одного раза в выражении;
- функции, содержащие циклы, переключатели и операторы переходов;
- функции, которые имеют слишком большой размер, чтобы сделать подстановку.

Функции с переменным числом параметров

В C++ допустимы функции, у которых при компиляции не фиксируется число параметров, и может быть неизвестен тип этих параметров. Количество и тип параметров становится известным только в момент вызова, когда явно задан список фактических параметров.

Каждая функция с переменным числом параметров должна иметь хотя бы один обязательный параметр. Определение функции с переменным числом параметров:

тип имя (явные параметры, ...)

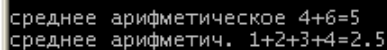
{ тело функции }

После списка обязательных параметров ставится запятая, а затем многоточие, которое показывает, что дальнейший контроль соответствия количества и типов параметров при обработке вызова функции производить не нужно. Существует два механизма определения количества и типов параметров:

- 1) известно количество параметров, которое передается как обязательный параметр;
- 2) известен признак конца списка параметров.

Пример 2.2. Найти среднее арифметическое последовательности чисел, если известно количество параметров (подход 1).

```
#include <iostream>
#include <windows.h>
using namespace std;
float sum(int k, ...)
{
    int *p=&k;          //настроили указатель на параметр k
    double s=0,n=k;
    for(;k!=0;k--)
        s+=*(++p);
    return s/n;
}
void main()
{
    SetConsoleOutputCP(1251);
    cout<<"\nсреднее арифметическое 4+6="<<sum(2,4,6);    /* среднее арифметич. 4+6*/
    cout<<"\nсреднее арифметич. 1+2+3+4="<<sum(4,1,2,3,4);/*среднее арифметич. 1+2+3+4*/
}
```



В **примере 2.2** для доступа к списку параметров используется указатель **p* типа *int*. Он устанавливается на начало списка параметров в памяти, а затем перемещается по адресам фактических параметров (*++p*).

Пример 2.3. Найти среднее арифметическое последовательности чисел, если известен признак конца списка параметров (подход 2).

```
#include <iostream>
#include <conio.h>
using namespace std;
double sum(int k, ...)
{
    int *p=&k;          //настроили указатель на параметр k
    double s=0, i;
    for( i=1;*p!=0;i++)    //пока нет конца списка
        s+=*(p++);
    return s/(i-1);
}
void main()
{
    cout<<"\n4+6="<<sum(4,6,0);    /*находит среднее арифметическое 4+6*/
    cout<<"\n1+2++3+4="<<sum(1,2,3,4,0);    /*находит среднее арифметич. 1+2+3+4*/
}
```

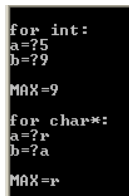
Задание 2.2. Найти произведение отрицательных элементов последовательности чисел, если известно количество параметров (используя подход 1 и 2).

Перегрузка функций

Цель перегрузки состоит в том, чтобы функция с одним именем по-разному выполнялась и возвращала разные значения при обращении к ней с различными типами и различным числом фактических параметров. Для обеспечения перегрузки необходимо для каждой перегруженной функции определить возвращаемые значения и передаваемые параметры так, чтобы каждая перегруженная функция отличалась от другой функции с тем же именем. **Компилятор определяет, какую функцию выбрать по типу фактических параметров.**

Пример 2.4. Поиск максимального значения

```
#include <iostream>
#include <conio.h>
using namespace std;
int max(int a,int b)
{
    if(a>b)return a;
    else return b;
}
char*max(char*a,char*b)
{
    if(strcmp(a,b)>0) return a;
    else return b;
}
void main()
{
    int a1,b1;
    char s1[20];
    char s2[20];
    cout<<"\nfor int:\n";
    cout<<"a=?";cin>>a1;
    cout<<"b=?";cin>>b1;
    cout<<"\nMAX="<<max(a1,b1)<<"\n";
    cout<<"\nfor char*:\n";
    cout<<"a=?";cin>>s1;
    cout<<"b=?";cin>>s2;
    cout<<"\nMAX="<<max(s1,s2)<<"\n";
}
```



```
for int:
a=?5
b=?9
MAX=9
for char*:
a=?r
b=?a
MAX=r
```

Задание 2.3. Создайте перегруженную функцию *sum*, которая находит сумму двух целых чисел, трех вещественных чисел и сумму кодов четырех символов.

Шаблоны функций

Шаблоны вводятся для того, чтобы автоматизировать создание функций, обрабатывающих разнотипные данные. При перегрузке функции для каждого используемого типа определяется своя функция. Шаблон функции определяется один раз, но определение параметризуется, т.е. тип данных передается как параметр шаблона.

Формат шаблона:

```
template <class имя_типа [,class имя_типа]>
заголовок_функции
{тело функции}
```

Таким образом, шаблон семейства функций состоит из 2 частей – заголовка шаблона: **template<список параметров шаблона>** и обыкновенного определения функции, в котором вместо типа возвращаемого значения и/или типа параметров, записывается имя типа, определенное в заголовке шаблона.

Пример 2.5. Шаблон функции, которая находит абсолютное значение числа любого типа.

```
template<class type>           //type – имя параметризуемого типа
type abs(type x)
{
    if(x<0) return -x;
    else return x;
}
```

Шаблон служит для автоматического формирования конкретных описаний функций по тем вызовам, которые компилятор обнаруживает в программе. Например, если в программе вызов функции осуществляется как `abs(-1.5)`, то компилятор сформирует определение функции `double abs(double x){...}`.

Пример 2.6. Найти максимальный элемент массива и заменить его на 0. Выполнить для целого и вещественного массивов. Для вывода массива и поиска максимального использовать шаблоны функции.

```
#include <iostream>
#include <conio.h>
#include <windows.h>
using namespace std;
template<class Data>
void print(Data a[], int n)
{
    for(int i=0;i<n;i++)
        cout<<a[i]<<" ";
    cout<<"\n";
}
template<class Data>
Data &rmax(int n, Data a[])
{
    int im=0;
    for(int i=0;i<n;i++)
        if(a[im]<a[i])im=i;
    return a[im];    //возвращает ссылку на максимальный элемент в массиве
}
void main()
{
    SetConsoleOutputCP(1251);
    int n=5,i;
    int x[]={ 10,20,30,15,5};
    cout<<"ИСХОДНЫЙ МАССИВ\n";
    print(x,n);
    cout<<"rmax(n,x)="<<rmax(n,x)<<"\n";
    rmax(n,x)=0; //заменяет максимальный элемент массива на 0
    cout<<"МАССИВ ПОСЛЕ ЗАМЕНЫ\n";
    print(x,n);
    float y[]={ 10.4,20.2,30.6,15.5};
    cout<<"\nИСХОДНЫЙ МАССИВ\n";
    print(y,4);
    cout<<"rmax(n,y)="<<rmax(n,y)<<"\n";
    rmax(4,y)=0; //заменяет максимальный элемент массива на 0
    cout<<"МАССИВ ПОСЛЕ ЗАМЕНЫ\n";
    print(y,4);
}
```

```
ИСХОДНЫЙ МАССИВ
10 20 30 15 5
rmax(n,x)=30
МАССИВ ПОСЛЕ ЗАМЕНЫ
10 20 0 15 5

ИСХОДНЫЙ МАССИВ
10.4 20.2 30.6 15.5
rmax(n,y)=30.6
МАССИВ ПОСЛЕ ЗАМЕНЫ
10.4 20.2 0 15.5
```

Основные свойства параметров шаблона функций:

1. Имена параметров должны быть уникальными во всем определении шаблона.
2. Список параметров шаблона не может быть пустым.
3. В списке параметров шаблона может быть несколько параметров, каждый из них начинается со слова *class*.

Задание 2.4. Транспонировать матрицу. Выполнить для целой и символьной матриц. Для ввода-вывода матриц и транспонирования использовать шаблоны функции.

Методические указания

При подготовке к занятию необходимо изучить: правила записи и способы передачи параметров различных типов функций и способы обращения к ним; правила записи и порядок выполнения программ, использующих такие функции.

Аудиторные и домашние задания

Контрольные вопросы

1. Правила использования функций с начальными (умалчиваемыми) значениями параметров.
2. Правила использования подставляемых (*inline*) функций.
3. Определение функции с переменным числом параметров.
4. Какова цель перегрузки функций?
5. Для чего вводятся шаблоны функций?
6. Основные свойства параметров шаблона функций?