Практическая работа 9

Понятие функции. Рекурсивные функции

Цель работы:

1. Изучить теоретический материал по теме применение функций C++
2. Решить задачи по теме обработка двумерных динамических массивов с использованием функций.
3. Решить задачу с использованием рекурсии.

Теоретические сведения

С увеличением объема программы становится невозможным удерживать в памяти все детали. Естественным способом снижения сложности любой задачи является разделение ее на части.

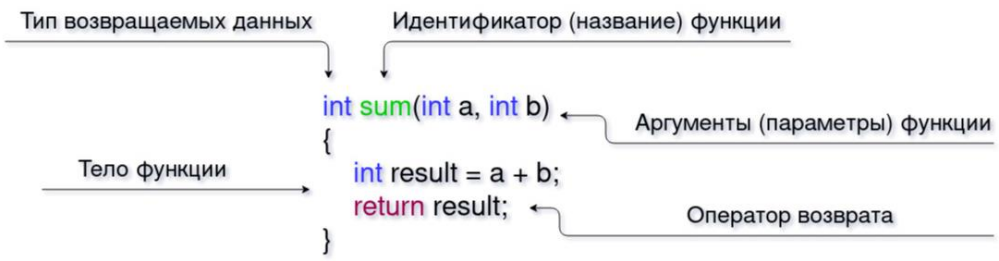
В языке C++ задача может быть разделена на более простые с помощью функций, после чего программу можно рассматривать в более укрупненном виде – на уровне взаимодействия функций.

Использование функций является первым шагом к повышению степени абстракции программы и ведет к ее упрощению. Разделение программы на функции позволяет избежать избыточности кода, поскольку функция создается один раз, а вызывать ее для выполнения можно многократно из разных точек программы.

C++ функция – это фундаментальное понятие. Каждая программа обязательно должна содержать одну главную функцию (с предопределенным именем main), которая обеспечивает создание точки входа в откомпилированную программу, и произвольное количество других функций, выполнение которых инициируется прямо или опосредованно вызовами из главной функции.

Функция – это именованная последовательность определений и операторов, выполняющая какое-либо законченное действие.

Определение функции – это ее текст, записанный на языке C++. В нем определяются тип, имя, параметры (заголовок функции), операторы тела и тип возвращаемого результата. Структура каждой функции идентична структуре главной функции.



Параметры функции являются важной частью ее определения – с помощью параметров функции могут обрабатывать различные данные при неизменности ее тела.

Формат определения функции:

```
[s][тип] имя ([спецификация_формальных_параметров]) {
тело_функции }
```

где s – спецификатор класса памяти функции;

тип – определяет тип результата, возвращаемого функцией с помощью оператора return. Тип может быть любым простым типом или указателем на любой допустимый тип, включая массив, структуру, класс, файл или функцию.

```
Если функция не возвращает значения, вместо типа в заголовке
указывается служебное слово void.
```

Если в заголовке функции тип не указан, по умолчанию принимается тип int;

имя – это либо main (для главной функции), либо идентификатор, с помощью которого функция вызывается для выполнения;

тело_функции – это всегда блок или составной оператор (даже если они не содержат операторов);

спецификация_формальных_параметров – определяет типы и имена формальных параметров, с помощью которых выполняется обмен данными между вызываемой и вызывающей функциями.

Спецификация каждого параметра состоит из типа и имени, между собой спецификации разделяются запятыми. **Отсутствие имени типа у параметра является синтаксической ошибкой.**

У функции может не быть параметров: в этом случае спецификация формальных параметров – это пусто или void.

Примечание. Символами [ ] отмечены необязательные элементы определения функции.

Формальными параметрами могут быть только идентификаторы (переменной, массива, указателя и т. д**.), но не выражения с ними**,

Примеры заголовков функций:

```
int f1() // 1
void f2(int i,char *s) // 2
```

В строке 1 определен заголовок функции с именем f1, которая в качестве результата своей работы возвращает значение целого типа. Параметров у функции нет.

В строке 2 определен заголовок функции с именем f2, у которой два параметра и нет возвращаемого значения.

При обращении к функции первым параметром должно быть значение целого типа (например, арифметическое выражение, имя переменной или константы целого типа), вторым – адрес значения символьного типа (например, адрес переменной типа char или строки).

Таким образом, определение формальных параметров в заголовке функции сообщает компилятору: – о типах значений, которые будут передаваться функции при обращении к ней; – об именах, под которыми эти значения будут известны внутри функции – все действия по обработке передаваемых значений будут выполняться с использованием имен формальных параметров.

Все переменные, определенные внутри функции, и ее формальные параметры являются локальными. Область их действия – тело функции. Выполнение функции продолжится до тех пор, пока не будут выполнены все операторы тела функции или не встретится оператор return – оператор возврата управления в точку вызова.

Необходимость использования этого оператора в функции определяется следующим образом:

```
1. Если функция не имеет возвращаемого значения, то в ней
допускается не использовать оператор return. В последних версиях
Visual Studio наличие return в процедуре вызывает ошибку.
```

Выполнение такой процедуры- функции завершится после выполнения всех операторов ее тела, поскольку в этом случае компилятор добавит в конец тела функции инструкции, соответствующие ее завершению автоматически.

Функции (процедуры), не возвращающие значения, нельзя использовать в выражениях. Однако это не означает, что у таких функций нет связей с "внешним миром" – результатом их выполнения могут быть значения, возвращаемые через аппарат параметров.

2. Если функция должна возвращать результат, то в ее теле необходимо использовать оператор return в виде return (выражение);

Выражение определяет значение возвращаемого функцией результата – выражение вычисляется, преобразуется (при необходимости) к типу возвращаемого значения и передается в точку вызова функции. Тип выражения должен совпадать (быть совместимым) с типом, указываемым в заголовке функции. Круглые скобки, ограничивающие выражение,

необязательны, однако возможны такие варианты программ, в которых скобки нужны, иначе выполнение программы прекращается без выдачи каких-либо сообщений. Функции, возвращающие значения, можно использовать в выражениях правой части оператора присваивания и во всех выражениях, где допустимо значение результата функции, в том числе в списках фактических параметров других функций.

Примеры определений функций:

1. Функция поиска более короткой строки

```
char * minstr(char *s1,char *s2) {
int i=0;
while(s1[i]!='\0' && s2[i]!='\0') i++;
if (s1[i] == '\0')
 return s1; else return s2; }
```

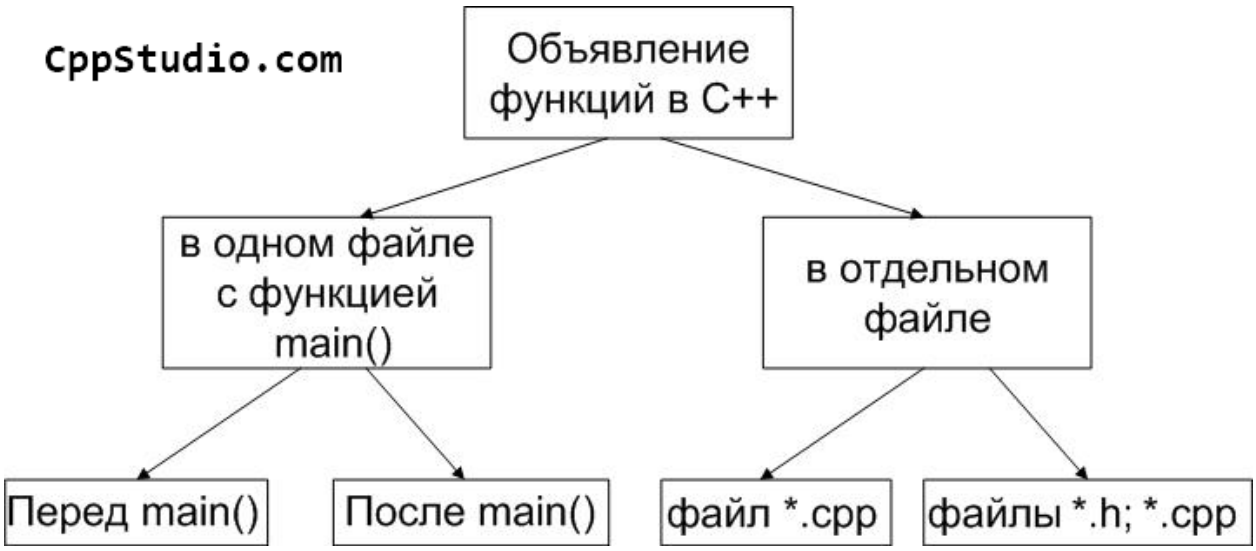
2. Функция нахождения максимального значения.

```
int max(int x, int y){
int m;
If x >= y
m =x; else m=y;}
return m;
```

Функции языка **C++ не могут быть вложенными** – внутри тела одной функции нельзя определить другую функцию.

Способы объявления функции

Описание функции (или ее прототип) делает возможным доступ к ней, помещая ее в область видимости под названием "файл программы". В тексте программы, состоящей из нескольких функций, до первого обращения к ней должно быть размещено либо ее определение, либо ее прототип. Такое размещение текстов функций и их прототипов позволяет компилятору выполнять контроль соответствия типов параметров и функции.



На **рисунке 3** показаны 4 способа объявления функций в языке программирования C++.

Рассмотрим структуры объявления функций в одном файле, с главной функцией. Функции можно объявлять в двух областях, до начала функции main(), после функции main(). До сих пор мы объявляли функции в одном файле, перед функцией main() — это самый простой из способов. Второй способ это объявление перед функцией main прототипа функции

```
 #include <iostream>
using namespace std;
char * minstr(char *,char *); // 1
int main() {
 cout << minstr("22","4444") << endl; // 2
 return 0; }
char * minstr(char *s1,char *s2) { // 3
 int i=0; while(s1[i]!='\0' && s2[i]!='\0') i++; if (s1[i] ==
'\0') return s1; else return s2; }
```

Функция minstr (ее определение начинается со строки 3) определяет более короткую из двух строк, являющихся ее параметрами, и возвращает указатель на нее. Перед оператором вызова функции minstr (строка 2) необходимо поместить ее прототип (строка 1), поскольку определение этой функции расположено в файле программы после оператора ее вызова.

Параметры функций

Описание механизма передачи параметров – один из наиболее важных аспектов программирования, знать который необходимо программисту любой квалификации. Он позволяет понять процесс взаимодействия и взаимосвязи функций, из которых состоит программа. Неверная передача параметров – один из источников серьезных ошибок.

Формальные и фактические параметры используются для связи функций по данным. Параметры, перечисленные в заголовке определения функции, называются формальными; записанные в операторе вызова функции – фактическими.

При разработке функции определяющими являются параметры, расположенные в списке формальных параметров. Именно их имена используются в теле функции. Фактические параметры передаются в функцию при ее вызове, при этом формальные параметры заменяются на фактические для обработки конкретных значений, содержащихся в них. Между формальными и фактическими параметрами должно соблюдаться взаимно-однозначное соответствие:

- количество формальных и фактических параметров должно быть одинаковым;
- соответствие между формальными и фактическими параметрами устанавливается по их взаимному расположению в списках: первому формальному параметру соответствует первый фактический, второму – второй и т. д.;
- типы формальных и соответствующих им фактических параметров должны быть одинаковыми (или совместимыми). Таким образом, внутри функции значения переданных ей фактических параметров становятся известными и обрабатываются с использованием имен соответствующих формальных параметров.

На имена формальных и фактических параметров ограничение по соответствию не накладывается – у них разные области действия. По этой причине имена **формальных и фактических параметров могут как совпадать, так и нет.** Сопоставление формальных и фактических параметров выполняется динамически при каждом вызове функции. При передаче параметров нарушение соответствия по количеству или типам параметров может привести к серьезным ошибкам, особенно если это несоответствие влечет за собой отличия в размерах памяти для переменных, так как контроля нарушения границ переменных, в том числе и массивов, в C++ нет.

Перед использованием параметров компилятор преобразует их следующим образом: все формальные параметры будут иметь тип не меньше int, параметры типа float преобразуются в тип double. По этой причине определение параметра с типом short эквивалентно определению параметра с типом int, с типом float – типу double. Для того чтобы компилятор не выполнял

эти преобразования, нужно определять целые переменные не меньше типа int, вещественные – не меньше double.

Способы передачи параметров

1. по значению
2. по адресу

1. способ. Так как формальный параметр – это просто переменная, определенная в заголовке функции и никак не связанная со значением фактического параметра, то такая замена реализуется копированием значения фактического параметра в участок памяти, который выделен для формального в момент вызова функции.

Такой способ называется передачей параметра по значению. Поскольку формальный параметр является локальной переменной вызываемой функции, то после ее завершения выделенная память освобождается, поэтому изменение значения формального параметра не оказывает никакого влияния на соответствующий фактический параметр. Передача параметра по значению реализует так называемые "входные" параметры

Данный способ имеет два преимущества:

1. Является самым надежным способом передачи параметров. Поскольку в вызываемую функцию передается только копия фактического параметра, она не может изменить значение фактического параметра – все изменения формального параметра действуют только на копию, а не на сам фактический параметр.

2. Фактическими параметрами могут быть и константы, и переменные, и выражения.

Однако преимущества этого способа передачи параметров являются одновременно и его недостатком, так как результатом работы функции может быть изменение значения фактического параметра, например: функция возвращает только одно значение, однако в случае сложного результата необходимо вернуть в точку вызова несколько значений – в этом случае параметры должны быть не "входными", а "выходными"; целью выполнения функции может быть модификация передаваемых ей данных, например сортировка значений массива – нет никакого смысла сортировать его копию; передаваемый параметр может быть настолько объемным, что копировать его неэффективно. Например, если передавать по значению структуру большого объема, то может просто не хватить памяти, чтобы сделать копию, или же затраты на копирование будут слишком велики.

2 Способ. Существует другой способ передачи параметров – передача по адресу (ссылке). При использовании этого способа в вызываемую функцию передается адрес фактического параметра. Формальный параметр в этом случае должен быть указателем соответствующего типа данных.

Использование внутри вызываемой функции операции разыменования формального параметра-указателя позволяет получить доступ к адресуемому объекту – фактическому параметру вызывающей функции, а значит, становится возможным изменение значения фактического параметра посредством изменения формального. Помимо этого исключаются затраты памяти на создание копии фактического параметра, что имеет значение при передаче структур данных большого объема

```
#include <iostream>
using namespace std;

/*ФУНКЦИЯ ЗАПОЛНЕНИЯ ДВУМЕРНОГО МАССИВА*/
void fill_array(int** arr, const int N, const int M) {   //указатель на указатель
адреса где хранится будущий массив и число ячеек массива
    int count = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            arr[i][j] = rand() % 10;
        }
    }
}

/*ФУНКЦИЯ ВЫВОДА ДВУМЕРНОГО МАССИВА НА ЭКРАН*/
void print_array(int** arr, const int N, const int M) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            cout << arr[i][j] << '\t';
        }   cout << '\n';
    }
}

int main() {
    int row = 0, col = 0;    //
    /*ЗАДАЁМ ЧИСЛО строе и столбцов ВО ВРЕМЯ РАБОТЫ ПРОГРАММЫ*/
    cout << "input row: ";
    cin >> row;
    cout << "input col: ";
    cin >> col;

    /*Выделение памяти*/
    int** p = new int* [row];   //Создание строк
    for (int i = 0; i < row; i++) {
        p[i] = new int[col];  //Создание колонок для каждой строки
    }
    //================================

    fill_array(p, row, col);      //Обращение к функции заполнения двумерного
массива
    print_array(p, row, col);     //Обращение к функции вывода двумерного массива

//================================

    for (int i = 0; i < row; i++) {
        delete[]p[i];        //освобождение памяти
    }
    delete[]p;   //Отбор у программы выделенной под указательную переменную памяти
      }
```

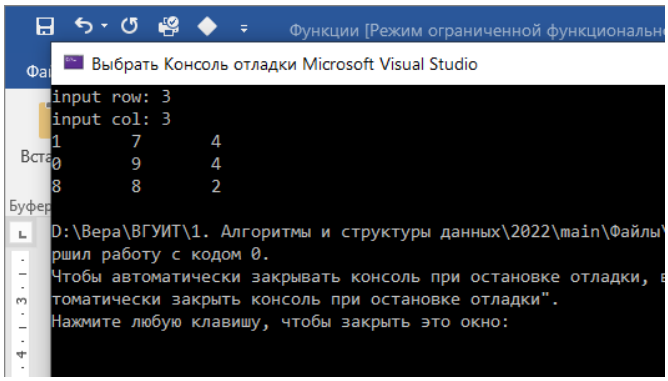


Рис. 1 -Результат работы программы

2. Рекурсивные функции

Рекурсивной называется функция, тело которой содержит вызов самой себя.

Другими словами, рекурсия – определение части функции через саму себя. При этом рекурсивная функция может вызывать саму себя как непосредственно (в своем теле), так и косвенно (через другую функцию). Многие задачи программирования могут быть решены с помощью рекурсивных функций. К таким относят, например, задачи нахождения факториала n! и чисел Фибоначчи.

Любую задачу можно свести к рекурсивной, если для нее выполняются следующие условия:

− исходную задачу можно разделить на ряд более простых подзадач меньшей размерности;

− каждая из подзадач имеет алгоритм решения, схожий с алгоритмом исходной задачи, поэтому каждую из подзадач рассматривать как самостоятельную задачу, алгоритмически неотличимую от исходной;

− все множество подзадач должно содержать хотя бы одно базовое решение (базовым называется решение, вычисляемое явно, а не через разделение на подзадачи).

Ключевым моментом для рекурсии является возможность использования одной и той же функции для вычисления каждой подзадачи независимо от их количества. Разделение задачи на подзадачи и вызов для каждой из них рекурсивной функции называется шагом рекурсии. На каждом шаге рекурсии выполняется сведение задачи к более простой до тех пор, пока не будет получено базовое решение. В качестве примера рассмотрим задачу вычисления факториала числа n!.

Известно, что факториал определяется следующим образом:

n! = 1 · 2 ·3 n.

Данную задачу можно решить итеративно – с помощью цикла. В то же время, выражение для вычисления факториала можно записать рекурсивно: n! = n·(n-1)! Таким образом, решение исходной задачи n! представлено через решение задачи (n-1)!, которая является более простой с вычислительной

точки зрения, поскольку она стала короче на одну операцию. Так как алгоритм вычисления n! не зависит от значения n, то можно выполнить рекурсивный переход (n-1)! Пока на дойдем до 1!=1

В качестве базы для вычисления факториала используется соотношение 1!= 1.

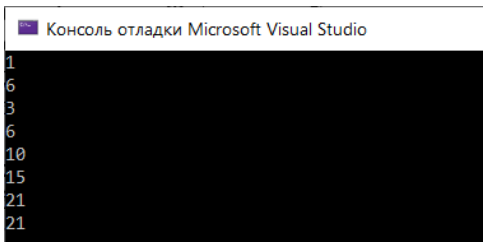
```
#include <iostream>
using namespace std;
int fact(int n) {
	if (n == 1)
		return 1;
	else {
		int k = n * fact(n - 1);
		cout << k << endl;
		return k;
	}
}
int main() {
	int n;
	cin >> n;
	int x = fact(n);
	return 0;
```

$$S = N1 + (N1 + 1) + (N1 + 2) + \ldots + N2$$

Представим эту функция как рекурсивную в качестве базовой функции здесь выступает нижняя граница интервала.

$$S(N1, N2) = N2 + S(N1, N2 - 1).$$

```
#include <iostream>
using namespace std;
int sum(int n, int n1) {
	if (n == n1)
		return n1;
	else {
		int k = n + sum(n - 1, n1);
		cout << k << endl;
		return k;
	}
                                        }
int main() {
	int n1, n2;
	cin >> n1 >> n2;
	int s = 0;
	cout << sum(n2, n1) << endl;
	return 0;
	}
```



Задания 1.

Функции. Динамические двумерные массивы.

Из практической работы №7. Задание 2 (двумерные массивы) реализовать графический алгоритм и программу на с++ с использованием функций. Считать матрицу динамической. Количество элементов строки столбцов вводить с клавиатуры в главной программе.

В программе д.б. реализовано следующие функции

1. Функция заполнения матрицы с клавиатуры; (пользователь должен иметь возможность выбирать способ ввода матрицы)
2. Функция заполнения матрицы случайным образом;
3. Функция вывода матрицы,
4. Функция решения задания по варианту.

Задание 2 Рекурсия

С использованием рекурсии решить следующие задачи:

1. Напишите рекурсивную функцию для вычисления суммы заданных положительных целых чисел a и b без прямого использования оператора +.
2. Напишите функцию без итерационного оператора для поиска максимального значения в массиве целых чисел.
3. Даны два целых числа A и B (каждое в отдельной строке). Выведите все числа от A до B включительно, в порядке возрастания, если A < B, или в порядке убывания в противном случае.
4. Дано натуральное число N. Выведите слово YES, если число N является точной степенью двойки, или слово NO в противном случае. Операцией возведения в степень пользоваться нельзя!
5. Дано натуральное число N. Выведите слово YES, если число N является точной степенью пятерки, или слово NO в противном случае. Операцией возведения в степень пользоваться нельзя!
6. Дано натуральное число N. Вычислите сумму его цифр. При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется).
7. Дано натуральное число N. Выведите все его цифры по одной, в обратном порядке, разделяя их пробелами или новыми строками. При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется). Разрешена только рекурсия и целочисленная арифметика.
8. Дано натуральное число n>1. Проверьте, является ли оно простым. Программа должна вывести слово YES, если число простое и NO, если число составное. Алгоритм должен иметь сложность O(logn). Указание. Понятно, что задача сама по себе нерекурсивна, т.к. проверка числа n на простоту никак не сводится к проверке на простоту меньших чисел. Поэтому нужно сделать еще один параметр рекурсии: делитель числа, и именно по этому параметру и делать рекурсию.

9. Дано натуральное число n>1. Выведите все простые множители этого числа в порядке неубывания с учетом кратности. Алгоритм должен иметь сложность O(logn).

10. Дано число n, десятичная запись которого не содержит нулей. Получите число, записанное теми же цифрами, но в противоположном порядке.

При решении этой задачи нельзя использовать циклы, строки, списки, массивы, разрешается только рекурсия и целочисленная арифметика. Функция должна возвращать целое число, являющееся результатом работы программы, выводить число по одной цифре нельзя.