Функции (продолжение)

Лекция 8

Вопрос 1

Передача массивов в качестве параметров

Общие положения

- При использовании в качестве параметра *массива* в функцию передается указатель на его первый элемент, иными словами, массив всегда передается по адресу.
- При этом информация о количестве элементов массива теряется, и следует передавать его размерность через отдельный параметр.

Пример

Функция нахождения суммы элементов массива

```
#include <iostream.h>
 …
int sum(const int* mas, const int n);
int const n = 5;

int main() {
    setlocale(LC_ALL,"");
    int marks[n] = {3, 4, 5, 4, 4};
    cout << "Сумма элементов массива: " << sum(marks, n);
    return 0;
}
```

Пример (продолжение)

```
int sum(const int* mas, const int n) {
   // варианты: int sum(int mas[], int n)
   // или int sum(int mas[n], int n)
   // (величина n должна быть константой)
   int s = 0;
   for (int i = 0; i<n; i++)
    s += mas[i];
   return s;
}
```

Многомерные массивы

- При передаче *многомерных массивов* все размерности, если они не известны на этапе компиляции, должны передаваться в качестве параметров.
- Внутри функции массив интерпретируется как одномерный, а его индекс пересчитывается в программе.

Пример

Нахождение суммы элементов двумерных массивов **a** и **b**

```
#inc1ude <stdio.h>
#include <stdlib.h>
 …
int sum(const int *a, const int nstr, const int
  nstb);
```

Пример (продолжение)

```
int main(){
    setlocale(LC_ALL,"");
    int b[2][2] = {{2, 2}, {4, 3}};
    printf("Сумма элементов b:  %d\n", sum(&b[0][0], 2, 2));
    // имя массива передавать в sum нельзя из-за несоответствия
    // типов
    int i, j, nstr, nstb, *a;
    printf("Введите количество строк и столбцов: \n");
    scanf("%d%d", &nstr, &nstb);
    a = (int *)malloc(nstr * nstb * sizeof(int));
    for (i = 0; i<nstr; i++)
     for (j = 0; j<nstb; j++)
            scanf("%d", &a[i * nstb + j]);
    printf("Сумма элементов a: %d\n", sum(a, nstr, nstb));
    return 0;
}
```

Пример (продолжение)

```
int sum(const int *a, const int nstr, const int
   nstb){
    int i, j, s = 0;
    for (i = 0; i<nstr; i++)
        for (j = 0; j<nstb; j++)
            s += a[i * nstb + j];
    return s;
}
```

Альтернативный пример

```
#include <iostream.h>
int sum(int **a, const int nstr, const int nstb);
int main(){
    int nstr, nstb;
    cin >> nstr >> nstb;
    int **a, i, j;
    // Формирование матрицы a:
    a = new int* [nstr];
    for (i = 0; i<nstr; i++)
     a[i] = new int [nstb];
    for (i = 0; i<nstr; i++)
     for (j = 0; j<nstb; j++)
            cin >> a[i][j];
    cout << sum(a, nstr, nstb);
    return 0;
}
```

Альтернативный пример (продолжение)

```
int sum(int **a, const int nstr, const int
   nstb){
   int i, j, s = 0;
   for (i = 0; i<nstr; i++)
    for (j = 0; j<nstb; j++)
        s += a[i][j];
   return s;
}
```

Вопрос 2

Рекурсивные функции

Определение

- Рекурсивной называется функция, которая вызывает саму себя. Такая рекурсия называется *прямой*.

- Существует еще *косвенная* рекурсия, когда две или более функций вызывают друг друга.

Особенности

- Если функция вызывает себя, в стеке создается копия значений ее параметров, как и при вызове обычной функции, после чего управление передается первому исполняемому оператору функции.
- При повторном вызове этот процесс повторяется.
- Для завершения вычислений каждая рекурсивная функция должна содержать хотя бы одну нерекурсивную ветвь алгоритма, заканчивающуюся оператором возврата.
- При завершении функции соответствующая часть стека освобождается, и управление передается вызывающей функции, выполнение которой продолжается с точки, следующей за рекурсивным вызовом.

Пример

Вычисление факториала числа **n**

```
long fact(long n){
   if (n==0 || n==l) return 1;
   return (n * fact(n - 1));
}
```

Пример

Вычисление факториала числа **n**
(альтернативный вариант)

```
long fact(long n){
  return (n>l) ? n * fact(n - 1) : 1;
}
```

Вопрос 3

Передача имен функций в качестве параметров

Особенности

Функцию можно вызвать через указатель на нее. Для этого объявляется указатель соответствующего типа и ему с помощью операции взятия адреса присваивается адрес функции:

```
void f(int a ){/*...*/ } // определение функции
void (*pf)(int); // указатель на функцию
…
pf = &f;      // указателю присваивается адрес функции
              // (можно написать pf = f;)
pf(10);       // функция f вызывается через указатель pf
              // (можно написать (*pf)(10) )
```

Использование typedef

Для того чтобы сделать программу легко читаемой, при описании указателей на функции используют переименование типов (typedef). Можно объявлять массивы указателей на функции (это может быть полезно, например, при реализации меню):

```
// Описание типа PF как указателя
// на функцию с одним параметром типа int:
typedef void (*PF)(int);
// Описание и инициализация массива указателей:
PF menu[] = {&new, &open, &save};
menu[l](10); // Вызов функции open
```

Здесь new, open и save – имена функций, которые должны быть объявлены ранее.

Передача функции через указатель

Указатели на функции передаются в подпрограмму таким же образом, как и параметры других типов:

```
#include <iostream.h>
typedef void (*PF)(int);
void f1(PF pf){      // функция f1 получает в качестве параметра
// указатель типа PF
    pf(5);// вызов функции, переданной через указатель
}

void f(int i ){cout << i;}

int main(){
    fl(f);
    return 0;
}
```

Тип указателя и тип функции, которая вызывается посредством этого указателя, должны совпадать в точности.

Вопрос 4

Функция main()

Особенности

- Функция, которой передается управление после запуска программы, должна иметь имя **main**.
- Она может возвращать значение в вызвавшую систему и принимать параметры из внешнего окружения.
- Возвращаемое значение должно быть целого типа.

- Стандарт предусматривает два формата функции:

// без параметров:

тип main(){ /* ... */}

// с двумя параметрами:

тип main(int argc, char* argv[]){ /* ... */}

Параметры функции main

- Первый параметр (argc) определяет количество параметров, передаваемых функции, включая имя самой программы.
- Второй параметр (argv) является указателем на массив указателей типа char*.
- Каждый элемент массива содержит указатель на отдельный параметр командной строки, хранящийся в виде строки, оканчивающейся нуль-символом.
- Первый элемент массива (argv[0]) ссылается на полное имя запускаемого на выполнение файла, следующий (argv[1]) указывает на первый параметр, argv[2] – на второй параметр, и т.д.

Возвращаемое значение

- Если функция main() ничего не возвращает, вызвавшая система получит значение, означающее успешное завершение.
- Ненулевое значение означает аварийное завершение.
- Оператор возврата из **main()** можно опускать.

Пример

```
#include <iostream.h>
void main(int argc, char* argv[]){
   for (int i = 0; i<argc; i++) cout << argv[i] << '\n';
}
```

Пусть исполняемый файл программы имеет имя main.exe и вызывается из командной строки:

d:\cpp\main.exe one two three

На экран будет выведено:

D:\CPP\MAIN.EXE

one

two

three