Функции

Лекция 6

Функция - это

именованный блок программы, созданный для решения одной небольшой задачи.

Наилучшим способом создания и поддержки больших программ является их конструирование из маленьких фрагментов, каждый из которых более управляем, чем сложная программа

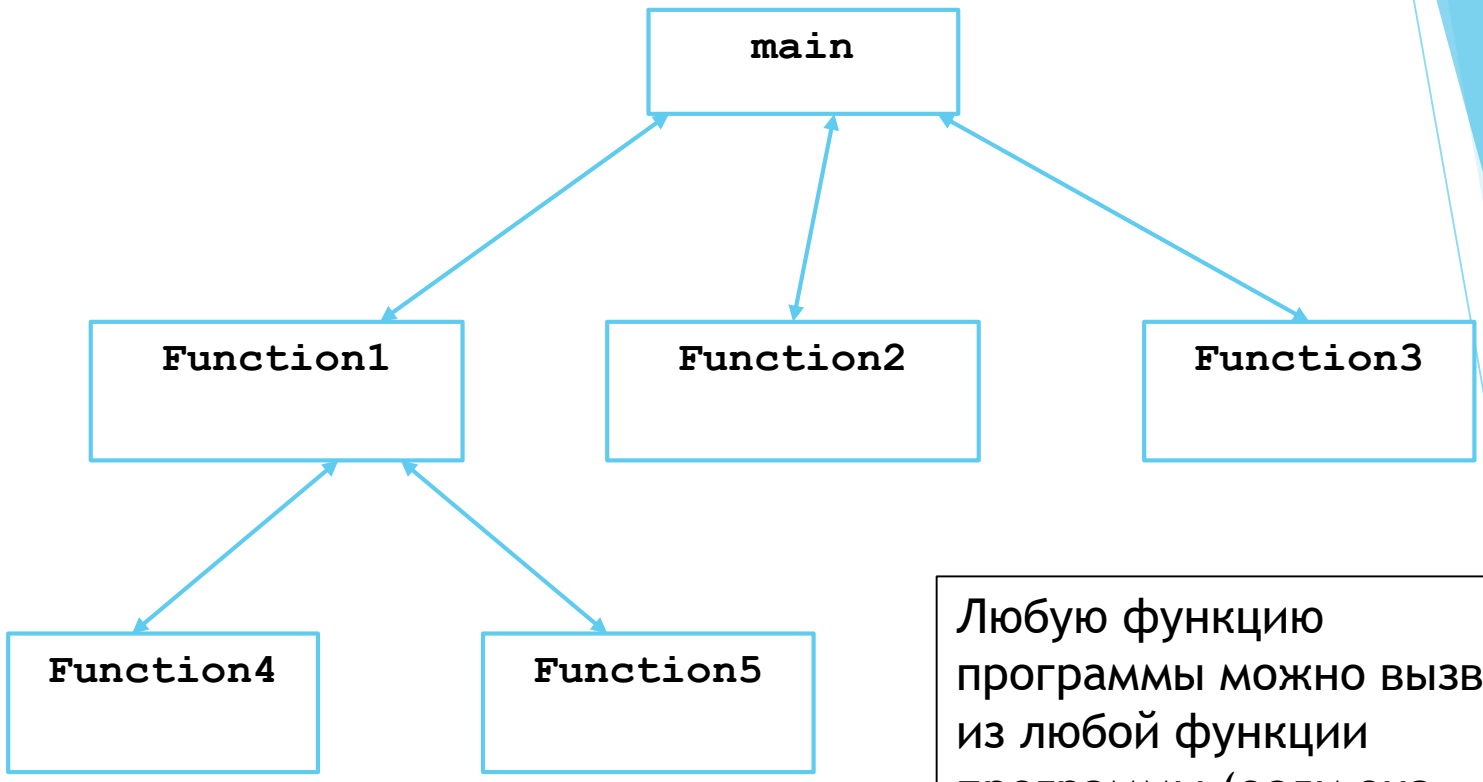
Разделяй и властвуй!

(Царь Филипп,
отец Александра Македонского)

Функции

- Повторяющиеся фрагменты программы можно оформить в виде функций.
- Таким же образом (в виде функций) можно оформить логически целостные фрагменты программы, даже если они не повторяются.
- Тогда в тексте основной программы (вызывающей функции), вместо помещённого в функцию фрагмента, вставляется инструкция «Вызов функции». При выполнении такой инструкции работает вызванная функция.
- После этого продолжается исполнение основной программы, начиная с инструкции, следующей за командой «Вызов функции».

Функциональная декомпозиция



Любую функцию программы можно вызвать из любой функции программы (если она доступна)

Хорошо спроектированная функция

- Полностью выполняет четко поставленную задачу
- Не берет на себя слишком много работы
- Не связана с другими функциями бесцельно
- Хранит данные максимально сжато
- Помогает распознать и разделить структуру программы
- Помогает избавиться от излишков, которые иначе присутствовали бы в программе

Краткое объявление функции - declaration

```
[<sc-specifier>]<type-specifier>name([type-specifier arg1,...);
```

- **<sc-specifier>** - спецификатор класса памяти (необязательный параметр)
- **<type-specifier>** - тип возвращаемого значения. Если функция не возвращает значения, то тип возвращаемого значения должен быть **void**
- **name** - имя функции
- **type-specifier arg1** - формальный параметр функции. Для каждого параметра указывается тип параметра и имя параметра. В объявлении функции имя параметра может быть пропущено (но такой подход не приветствуется)

Полное объявление функции - definition

```
[<sc-specifier>]<type-specifier>name([type-specifier arg1,...)
{
    function's body
}
```

- Заголовок функции совпадает с кратким объявлением функции
- После заголовка функции идет тело функции
- Полное объявление функции в программе должна встречаться только один раз
- Краткое объявление функции в программе может встречаться любое количество раз (либо не встречаться вообще)
- Краткое объявление функции может быть указано внутри другой функции
- Полное объявление функции внутри другой функции не допускается

Рекомендации по именованию функций

- Слово `compute` может быть использовано в функциях, вычисляющих что-либо
- Слово `find` может быть использовано в методах, осуществляющих какой-либо поиск
- Слово `initialize` может быть использовано там, где объект или сущность инициализируется
- Названия функций должны быть глаголами, быть записанными в смешанном регистре и начинаться с нижнего

Определение функции

```
int calculateSquare (int);
```

прототип (заголовок) функции

```
int main() {
    for (int x = 1; x<=10; x++)
        std::cout << calculateSquare (x) << " ";
    std::cout << std::endl;
    return 0;
}
```

вызов функции

```
int calculateSquare (int y)
{
    return y*y;
}
```

полное определение функции

Задача. Определить максимальное из трех чисел

```
int max (int, int, int);
```

```
int main()
{
    int a,b,c;
    std::cout << "Enter 3 integers: ";
    std::cin >> a >> b >> c;
    std::cout << "Maximum is: "
              << max(a, b, c) << std::endl;
    return 0;
}
```

```
int max (int x, int y, int z)
{
    int result = x;
    if (y > result)
        result = y;
    if (z > result)
        result = z;
    return result;
}
```

Формальные параметры - указываются в описании функции

```
int max (int x, int y, int z)
```

Фактические параметры передаются при вызове функции

```
max(a,b,c)
```

Что будет напечатано?

```
int fa() { std::cout << "fa" << std::endl; return 1; }
int fb() { std::cout << "fb" << std::endl; return 2; }
int fc() { std::cout << "fc" << std::endl; return 3; }
```

```
void fun( int a, int b, int c)
{
   std::cout << a << b << c << std::endl;
}
```

```
int main()
{
    fun(fa(),fb(),fc());
    return 0;
}
```

Способы передачи параметров в функции

- вызов по значению
- вызов по ссылке с аргументами указателями
- вызов по ссылке с аргументами ссылками

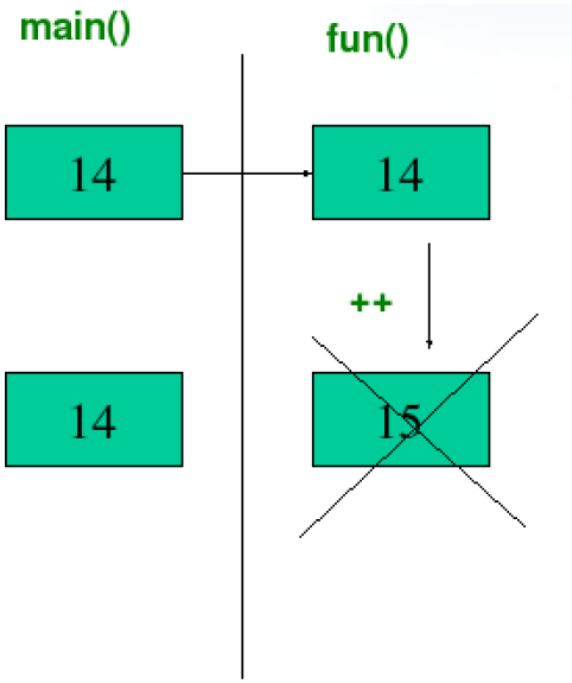
Передача параметра по значению

- создается копия аргумента
- изменения копии не влияет на значение оригинала в операторе вызова
- один из недостатков вызова по значению состоит в том, что если передается большой элемент данных, это может привести к значительным накладным расходам на вызов функции

Передача параметра по значению

```
void fun(short value)
{
    value++;
}
```

```
int main()
{
    short value=14;
    fun(value);
    std::cout << value;
    return 0;
}
```



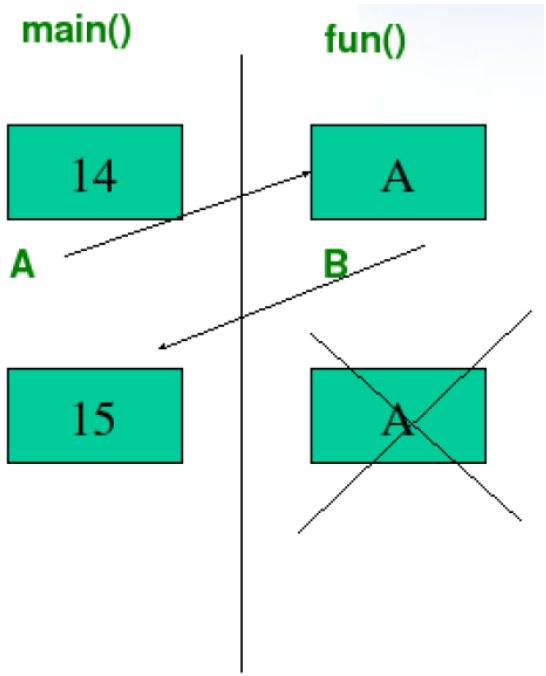
Передача параметра через указатель

- в качестве аргумента передается адрес параметра
- доступ к значению осуществляется через операцию разыменования
- изменения значения сохраняются при выходе из функции
- передача параметров через указатель – один из способов вернуть из функции больше одного значения одновременно

Передача параметра через указатель

```
void fun(short *value)
{
     (*value)++;
}
```

```
int main()
{
     short value=14;
     fun(&value);
     std::cout << value;
     return 0;
}
```



Передача параметров через указатель

```
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

Для изменения значения указатель нужно разыменовать

```
int main()
{
    int a = 5, b = 7;
    std::cout << "a=" << a << " " << "b=" << b << std::endl;
    swap(&a, &b);
    std::cout << "a=" << a << " " << "b=" << b << std::endl;
    return 0;
}
```

При вызове функции передаются адреса

Ссылка - это

второе имя переменной, псевдоним

```
//объявление целой переменной count
int count = 1;
//создание cRef как псевдонима для count
int &cRef = count;
//count = 2
++cRef;
```

Ни один оператор не выполняет действия над ссылкой

Инициализация ссылки

Ссылка должна быть инициализирована в момент объявления

```
int count = 1;
int &cRef = count;
int &cRef1;                 //ошибка
int &cRef2 = 2;             //ошибка
short &cRef3 = count;       //ошибка
```

Константные ссылки

```
const int &cRef = 1;


void fun(const int & ref);
int main()
{
    short m = 15;
    int n = 5;
    fun(10);
    fun(n);
    fun(m);
}
```

Константные ссылки могут используются как именованные константы и инициализироваться конкретными значениями

Передача параметров по ссылке

```
void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

Для изменения значения просто используется имя ссылки

```
int main()
{
    int a = 5, b = 7;
    std::cout << "a=" << a << " " << "b=" << b << std::endl;
    swap(a, b);
    std::cout << "a=" << a << " " << "b=" << b << std::endl;
    return 0;
}
```

При вызове функции передаются имена переменных

Ссылка как возвращаемое значение

```
int & fun1(int a)
{
    int n = 0;
    n += a;
    return n;
}
```

Плохо! Возвращается ссылка, копия переменной n не создается, при выходе из функции n уничтожается, поведение программы не определено

```
int fun1(int a)
{
    int n = 0;
    n += a;
    return n;
}
```

Хорошо! Создается копия возвращаемого значения, переменная n уничтожается

Передача массивов в функции

- Массив всегда передается как адрес первого элемента
- Все изменения значений элементов массива, выполненные в функции, при выходе из функции сохраняются

Передача массивов в функции. 1 вариант

```
void print(float av[], int N)
{
    for(int j = 0; j < N; j++)
        std::cout << av[j] << std::endl;
}
```

Передача массива как массива

```
int main()
{
   const int N = 20;
   float average[N] = {0};
   //…
   print(average, N);
   return 0;
}
```

В вызове функции указывается имя массива

Передача массивов в функции. 2 вариант

```
void print(float *av, int N)
{
    for(int j = 0; j < N; j++)
        std::cout << av[j] << std::endl;
}
```

Передача массива как указателя

```
int main()
{
   const int N = 20;
   float average[N] = {0};
   //…
   print(average, N);
   return 0;
}
```

В вызове функции указывается имя массива

Передача многомерных массивов в функции. 1 вариант

```
const int N = 10;
const int M = 7;
void set_temp(int t[][M])
{
        for(int i=0; i<N; i++)
                for(int j = 0; j<M; j++)
                        t[i][j] = rand()%41-30;
}
```

Вторая и последующие скобки всегда заполнены

```
int main()
{
    int temperature[N][M];
    set_temp(temperature);
}
```

Размер массива задан внешними константами

В вызове функции указывается имя массива

Передача многомерных массивов в функции. 2 вариант

```
const int N = 10;
const int M = 7;
void set_temp(int (*t)[M])
{
        for(int i=0; i<N; i++)
                for(int j = 0; j<M; j++)
                        t[i][j] = rand()%41-30;
}
```

Передается указатель на массив

```
int main()
{
    int temperature[N][M];
    set_temp(temperature);
}
```

В вызове функции указывается имя массива

Возврат массива из функции

```
const int SIZE = 5;
int * f()
{
    int arr[SIZE] = { 0 };
    for (int i = 0; i < SIZE; i++)
        arr[i] = rand() % SIZE;
    return arr;
}
```

Память под массив выделяется внутри функции
При выходе из функции эта память будет освобождена

Массив возвращается как адрес первого элемента

```
int main()
{
    srand(time(0));
    int *arr = f();
    for (int i = 0; i < SIZE; i++)
        std::cout << arr[i] << std::endl;
    return 0;
}
```

Из функции получаем адрес первого элемента

При попытке работы с таким массивом поведение программы не определено

Возврат массива из функции

```
const int SIZE = 5;
int * f()
{
   int * arr = new int [SIZE];
   for (int i = 0; i < SIZE; i++)
      arr[i] = rand() % SIZE;
   return arr;
}
```

Память под массив выделяется в динамической памяти. При выходе из функции все буде сохранено.

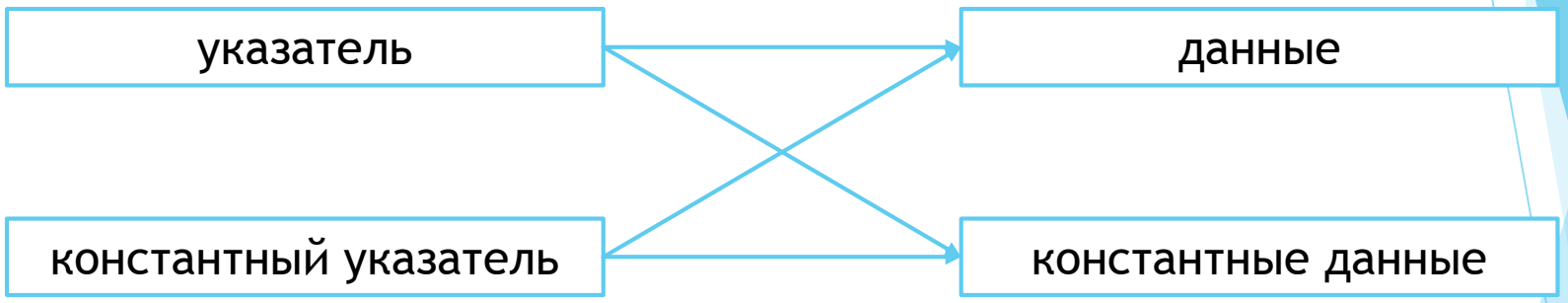
Массив возвращается как адрес первого элемента

```
int main()
{
   srand(time(0));
   int *arr = f();
   for (int i = 0; i < SIZE; i++)
      std::cout << arr[i] << std::endl;
   delete[] arr;
   return 0;
}
```

Работа с таким массивом возможна в любой функции, достаточно передать имя массива

В конце работы необходимо память принудительно освободить

Спецификатор `const` с указателями



Неконстантный указатель на неконстантные данные

Можно:

- изменять данные, на которые ссылается указатель, через операцию разыменования указателя
- изменять сам указатель

```
int a = 5, b = 7;
int * ptr = &a;
*ptr = 15;
a = 25;
ptr = &b;
```

Изменяем данные

Изменяем указатель

Неконстантный указатель на константные данные

Можно:

- изменять сам указатель

Нельзя:

- изменять данные, на которые ссылается указатель, через операцию разыменования указателя

```
int a = 5, b = 7;
const int * ptr = &a;
*ptr = 15;
a = 25;
ptr = &b;
```

Нельзя изменять данные через указатель

Можно изменить значение переменной

Можно изменить указатель

Константный указатель на неконстантные данные

Можно:

- изменять данные, на которые ссылается указатель, через операцию разыменования указателя

Нельзя:

- изменять сам указатель

```
int a = 5, b = 7;
int * const ptr = &a;
*ptr = 15;
a = 25;
ptr = &b;
```

Можно изменять данные через указатель

Нельзя изменить указатель

Константный указатель на константные данные

Нельзя:

- изменять данные, на которые ссылается указатель, через операцию разыменования указателя
- изменять сам указатель

```
int a = 5, b = 7;
const int * const ptr = &a;
*ptr = 15;
a = 25;
ptr = &b;
```

Нельзя изменять данные через указатель

Можно изменить значение переменной

Нельзя изменить указатель

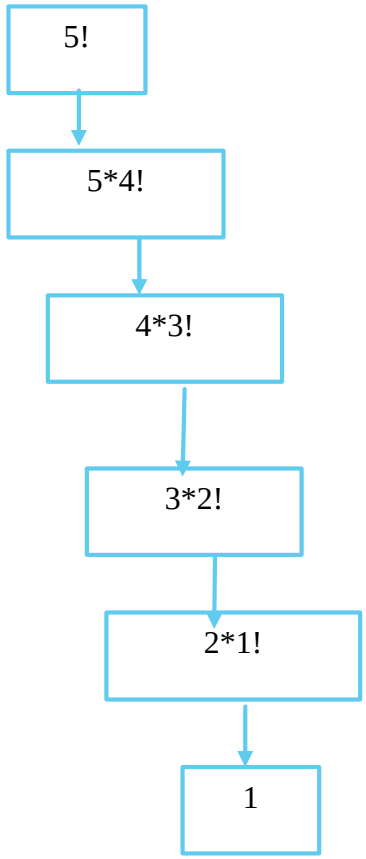
Использование констант с указателями

- Константный указатель – это имя массива
- Указатель на константные данные используется при передаче массивов в функции. Такой способ передачи не позволит изменить значение элементов массива в функции

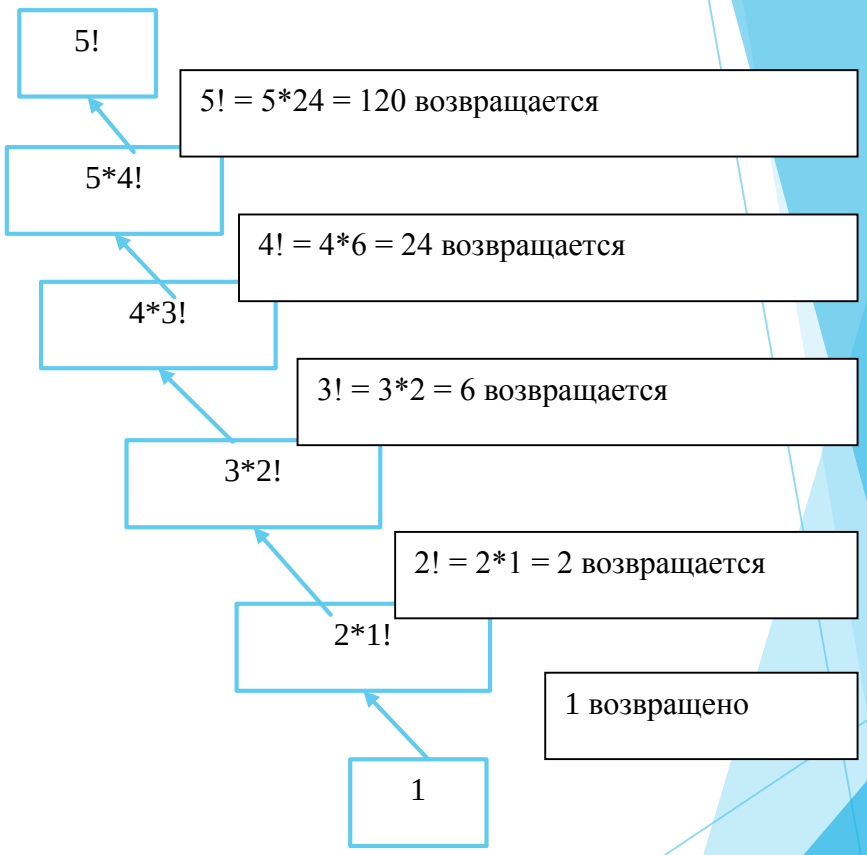
Рекурсия

Рекурсивная функция – это функция, которая вызывает сама себя либо непосредственно, либо косвенно с помощью другой функции

Дерево рекурсии для нахождения факториала



Процесс рекурсивных вызовов



Значения, возвращаемые после рекурсивного вызова

Рекурсивное вычисление факториала

```
unsigned long factorial (unsigned long number)
{
    if (number <=1)
        return 1;
    else
        return number * factorial (number-1);
}
```

Базовое условие - выход из рекурсии

Шаг рекурсии

Рекурсивный вызов функции

Формула для вычисления чисел Фибоначчи

$$\begin{cases} fib = 1,\ n = 1,\ 2 \\ fib = fib(n - 1) + fib(n - 2),\ n > 2 \end{cases}$$

Рекурсивное вычисление чисел Фибоначчи

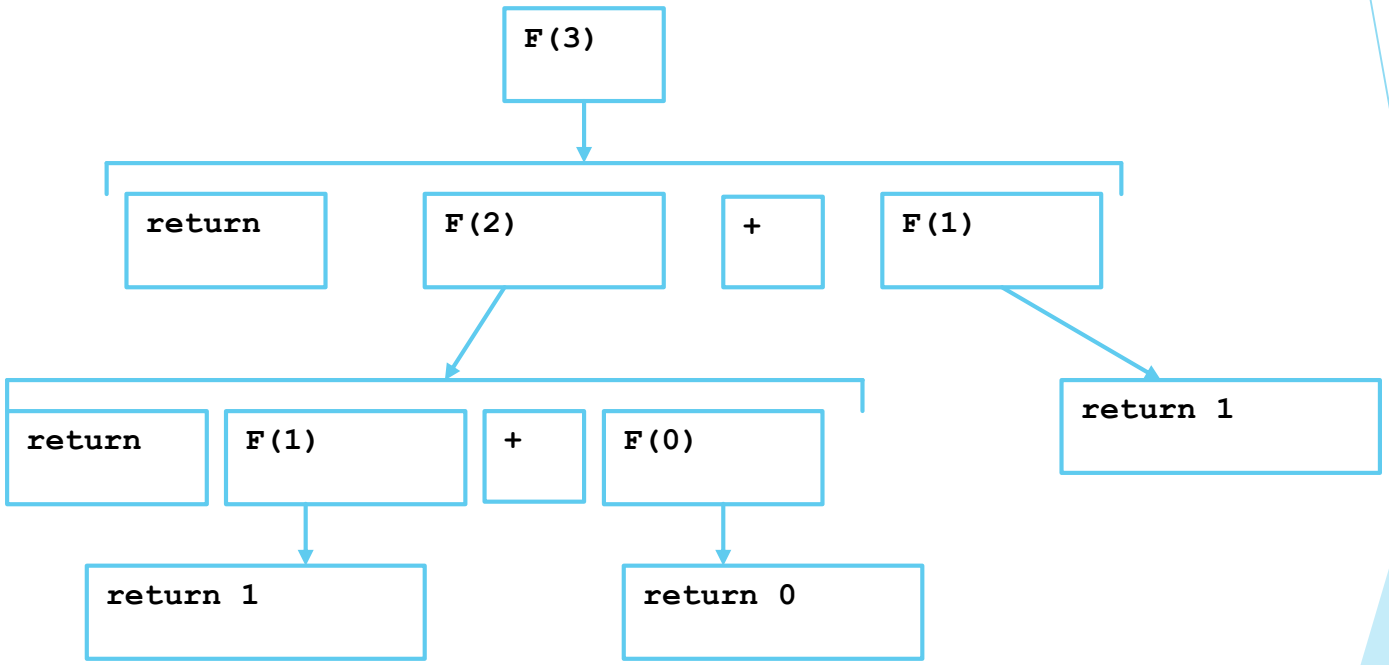
```
unsigned long fib (unsigned long n)
{
   if(n == 1 || n == 2)
      return 1;
   else
      return fib (n-1) + fib (n-2);
}
```

Базовое условие - выход из рекурсии

Шаг рекурсии

Рекурсивный вызов функции

Дерево рекурсии для вычисления чисел Фибоначчи



Рекурсивное вычисление чисел Фибоначчи

```
int count = 0;
unsigned long fib (unsigned long n)
{
    count++;
    if(n == 1 || n == 2)
        return 1;
    return fib (n-1) + fib (n-2);
}
```

```
int main()
{
    std::cout << fib(10) << std::endl;
    std::cout << fib(20) << std::endl;
    std::cout << fib(30) << std::endl;
    return 0;
}
```

```
count = 109
count = 13529
count = 1664079
```

Рекурсивная функция печати массива

```
void print(float* arr, int n)
{
   if(0 == n)
      return;
   print (arr+1, n-1);
   std::cout << arr[0] << ' ';
}
```

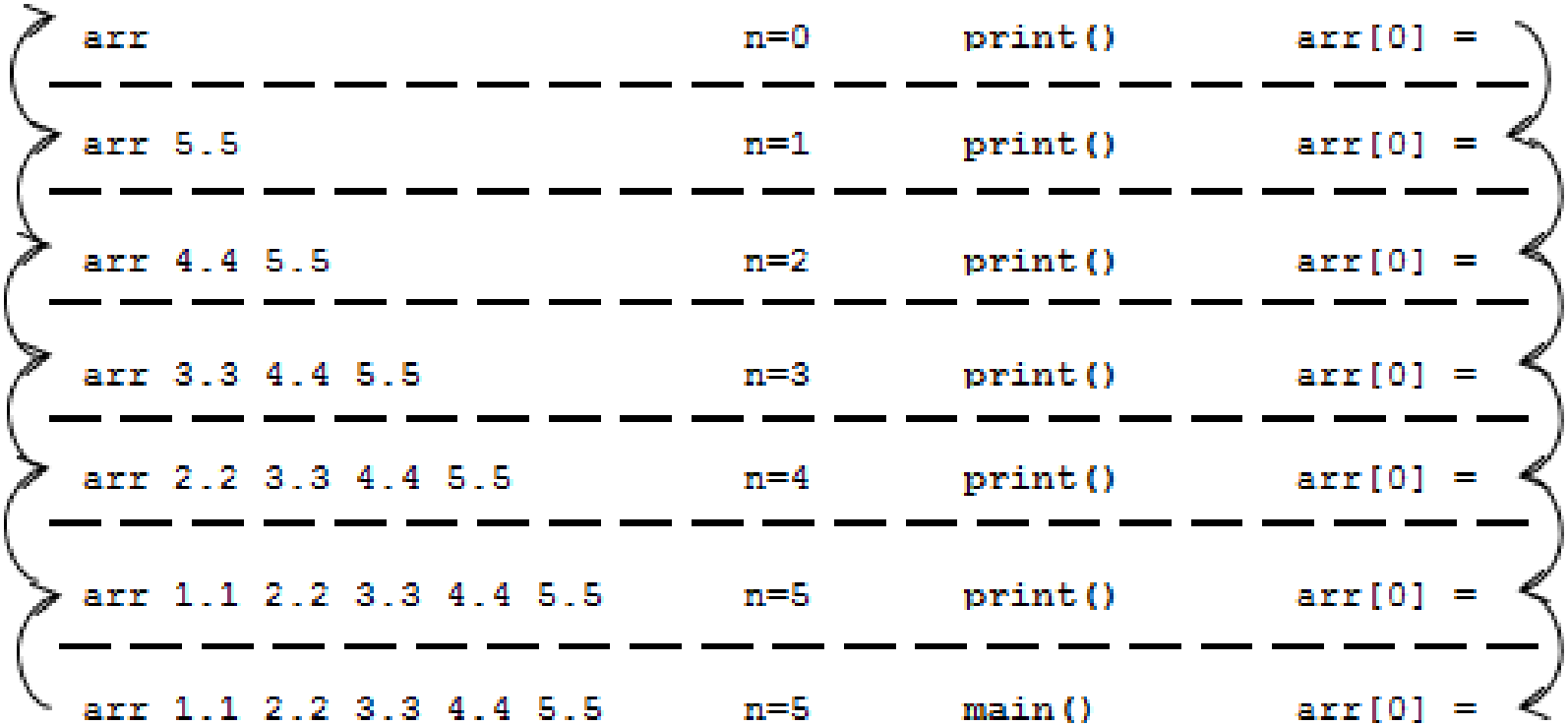
Базовое условие - выход из рекурсии

Шаг рекурсии

Рекурсивный вызов функции

Функция имеет тип **void**, но без оператора **return** организация рекурсии была бы невозможна

Стек вызовов функции



Перегрузка функций

- позволяет определить несколько функций с одним и тем же именем
- функции должны иметь разные наборы параметров (по меньшей мере, разные типы параметров)
- при вызове перегруженной функции компилятор C++ определяет соответствующую функцию путем анализа количества, типов и порядка следования аргументов в вызове

Перегрузка функций

```
int square (int x)
{
    return x*x;
}
```

```
double square (double y)
{
    return y*y;
}
```

```
int main()
{
    std::cout << "Integer square is " <<  square(7);
    std::cout << "Double square is " << square(7.1);
    return 0;
}
```

Параметры по умолчанию

- программист может указать, что параметр является параметром по умолчанию и задать для него значение по умолчанию
- если параметр по умолчанию не указан в вызове функции, то в вызов автоматически передается значение этого параметра по умолчанию
- параметры по умолчанию должны быть самыми правыми (последними) в списке параметров функции
- параметры по умолчанию должны быть указаны при первом упоминании имени функции – обычно в объявлении (прототипе)
- параметры по умолчанию могут быть константами, глобальными переменными или вызовами функций

Параметры по умолчанию

```
int test (double a,     int b);
int test (   int a,     double b);
int test (   int a,     int b,      int c = 3);
int test (   int a,     int b = 4);
int test (   int a = 7);
```

(1)
(2)
(3)
(4)
(5)

```
int main()
{
    test(3.5, 6);
    test(6, 3.5);
    test();
    test(3, 4, 5);
    test(2, 3);
    test(5);
    return 0;
}
```

Вызовы:
(1)
(2)
(5)
(3)
(???)
(???)

Указатель на функцию

- Возможны только две операции с функциями:
  - вызов;
  - взятие адреса.
- Указатель, полученный с помощью последней операции, можно впоследствии использовать для вызова функции
- Указатель на функцию также можно использовать в качестве параметра другой функции. Это аналог делегатов из других языков программирования

Указатель на функцию

```
void error(char* p)
{
    /* ... */
}
```

Определение функции

```
void (*perr)(char*);
```

Объявление указателя на функцию

```
void f()
{
    perr = &error;
    (*perr)("error");
    perr("error");
}
```

Связь указателя perr с функцией error

Вызов функции error через указатель perr
2 варианта

Ошибки при использовании указателей на функцию

```
void (*pf)(char*);
void   f1 (char*);
int    f2 (char*);
void   f3 (int*);
```

```
void f()
{
   pf = &f1;
   pf = &f2;
   pf = &f3;
   pf("asdf");
   (*pf)(1);
   int i = (*pf)("qwer");
}
```

хорошо
не тот тип возвращаемого значения
не тот тип параметра
хорошо
не тот тип параметра
void присваивается int

Пузырьковая сортировка

```
int main()
{
    const int SIZE = 10;
    int arr[SIZE] = { 2, 45, 3, 34, 43, 12, -5, 5, 67, 33 };
    for (int i = 0; i < SIZE; i++)
         std::cout << arr[i] << '\t';
    std::cout << std::endl;
    bubbleSort(arr, SIZE, less);
    for (int i = 0; i < SIZE; i++)
         std::cout << arr[i] << '\t';
    std::cout << std::endl;
    return 0;
}
```

Калькулятор

```
int doSum(int a, int b)
{
    return a + b;
}
```

```
int doSub(int a, int b)
{
    return a - b;
}
```

```
int doMul(int a, int b)
{
    return a * b;
}
```

```
int doDiv(int a, int b)
{
    if (b == 0)
        exit(-1);
    return a / b;
}
```

Калькулятор

```
int main()
{
    int(*menu[4])(int, int);
    int op;
    int a, b;
    menu[0] = doSum;
    menu[1] = doSub;
    menu[2] = doMul;
    menu[3] = doDiv;
    std::cout << "enter a: ";
    std::cin >> a;
    std::cout << "enter b: ";
    std::cin >> b;
    std::cout << "enter operation [0 - add, 1 - sub, 2 - mul, 3 - div]: ";
    std::cin >> op;
    if (op >= 0 && op < 4)
    {
        std::cout << menu[op](a, b) << std::endl;
    }
    return 0;
}
```

Массив указателей на функцию

Вызов нужной функции по индексу

Калькулятор

```
int main()
{
    int(**menu)(int, int) = NULL;
    int op;
    int a, b;
    menu = new (int(*)(int, int)) [4];
    menu[0] = doSum;
    menu[1] = doSub;
    menu[2] = doMul;
    menu[3] = doDiv;
    std::cout << "enter a: ";
    std::cin >> a;
    std::cout << "enter b: ";
    std::cin >> b;
    std::cout << "enter operation [0 - add, 1 - sub, 2 - mul, 3 - div]: ";
    std::cin >> op;
    if (op >= 0 && op < 4)
    {
        std::cout << menu[op](a, b) << std::endl;
    }
    delete [] menu;
    return 0;
}
```

Динамическое создание массива указателей на функцию

Командная строка аргументов

- Позволяет запускать программу с параметрами:

  `ping 192.168.0.4 -t -l 128`

- Реализована как параметры функции **`main`**
- У функции **`main`** появляется два параметра
  - количество аргументов
  - список аргументов
- Первый аргумент списка – имя исполняемого файла

Командная строка аргументов

```
int main(int argc, char* argv[])
{
   int i;
   for(i = 1; i < argc; i++)
       std::cout << argv[i] << (i < argc-1) ? ' ' : '\n';
   return 0;
}
```

Конец