

ОСНОВЫ ТЕОРИИ И ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Пышкин Евгений Валерьевич

к.т.н., доцент

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ

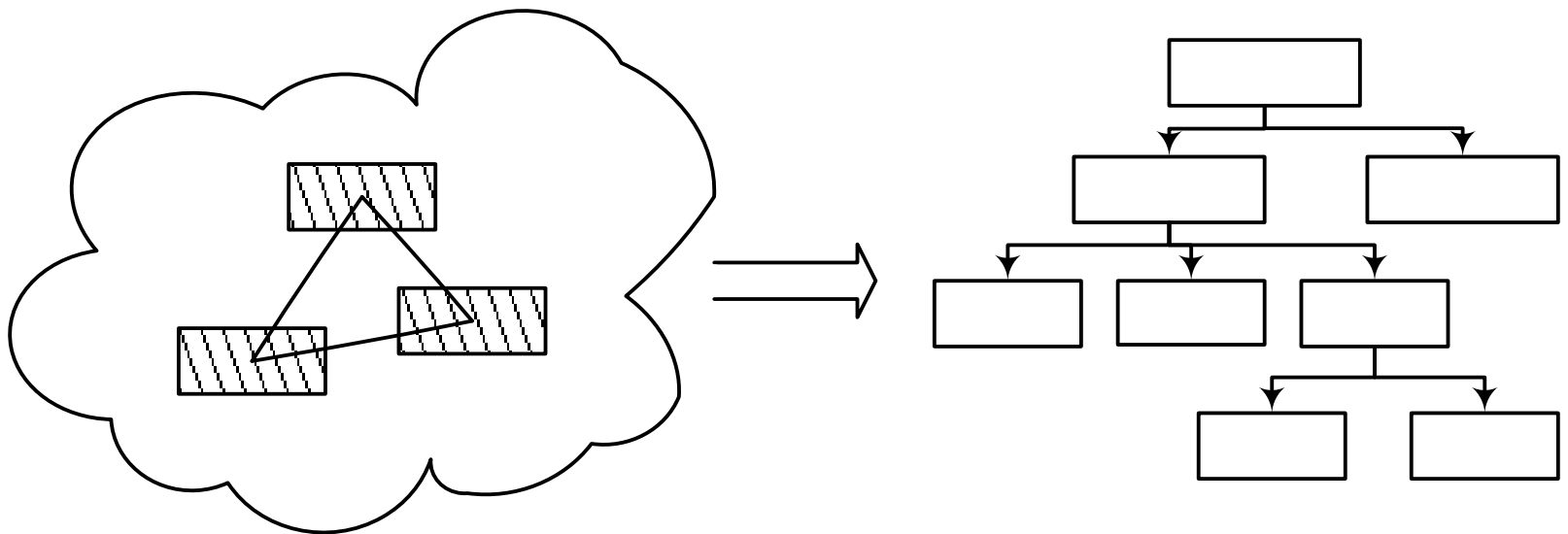
Блок 4. Функции, массивы, строки...

Функции в языке C++

- Стратегии проектирования
- Понятие функции
- Понятие об аргументах и параметрах
- Механизмы связывания аргументов и параметров: первоначальные сведения
- Когда требуется функциональное обособление

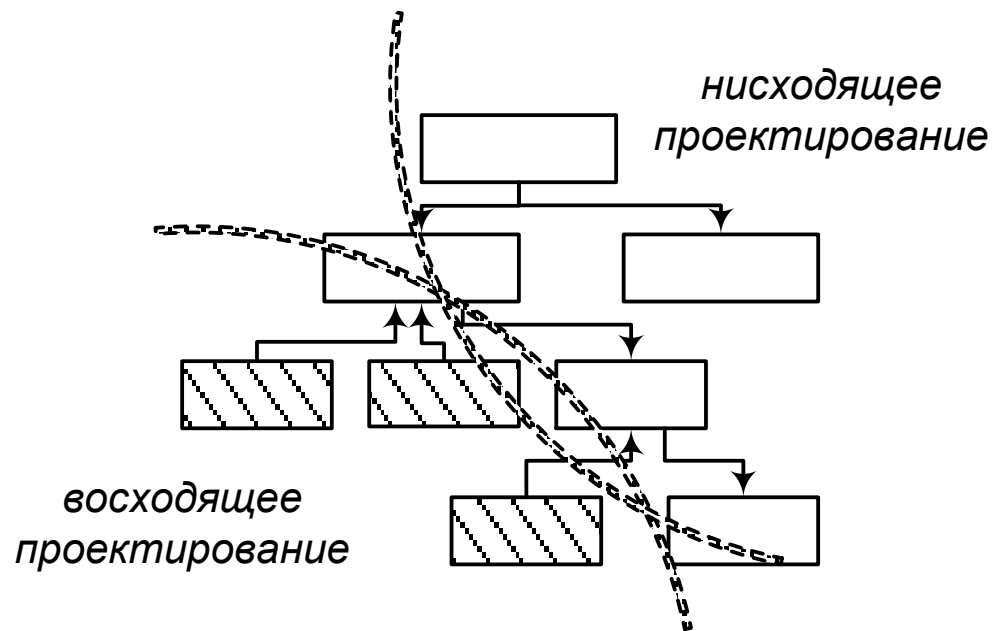
Стратегии проектирования

- Нисходящая функциональная декомпозиция



Стратегии проектирования

- Восходящее проектирование и сочетание стратегий



Процедуры и функции

- **Процедура** – обособленный именованный блок кода, выполняющий определенные действия, и взаимодействующий с другими функциональными блоками посредством связывания аргументов и параметров
- **Функция** – именованный блок, выполняющий определенные действия, и взаимодействующий с другими функциональными блоками посредством связывания аргументов и параметров, а также формирующий возвращаемое значение
 - ▣ *В традиции C и C++ используют термин «функция, не возвращающая значения» в качестве эквивалента термина «процедура»*

Функции: пример определения

```
/*
 * Определение, является ли заданный год високосным
 * Функция возвращает 1, если год високосный, иначе - 0
 */
int isLeapYear( int year )
{
    if( year % 4 != 0 )    return 0;
    if( year % 100 != 0 ) return 1;
    if( year % 400 != 0 ) return 0;
    return 1;
}
```

Функции: пример вызова

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

// Процедура проверки корректности календарной даты
void CheckDate( int day, int month, int year )
{
    int daysLimit[ 12 ] = { 31,29,31,30,31,30,
                           31,31,30,31,30,31 };

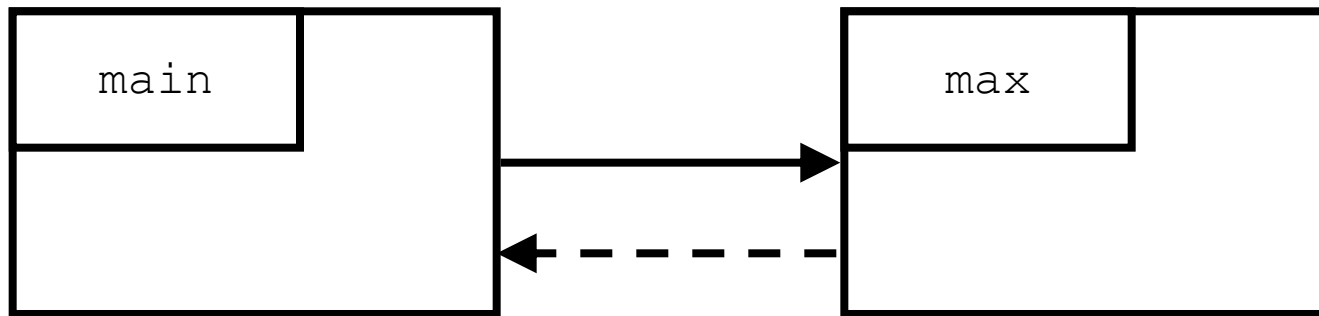
    bool isCorrect = true;
    if( (month < 1) || (month > 12) )
        isCorrect = false;
    else if( (day < 1) || (day > daysLimit[ month-1 ]) )
        isCorrect = false;
    else if( (year<1) || (year > 5000) )
        isCorrect = false;
    else if( (month==2) && (day==29) && (!isLeapYear( year )) )
        isCorrect = false;
    if( correct ) cout << "Date is correct" << endl;
    else          cout << "Date is incorrect" << endl;
}
```


Связывание аргументов и параметров: основные механизмы

- Связывание по значению
- Связывание с косвенной адресацией
- Связывание по ссылке

Пример: вычисление максимума двух чисел

- Постановка задачи



Связывание по значению

□ Определение функции

```
double max1( double arg1, double arg2 )  
{  
    if( arg1 > arg2 ) return arg1;  
    return arg2;  
}
```

Связывание по значению

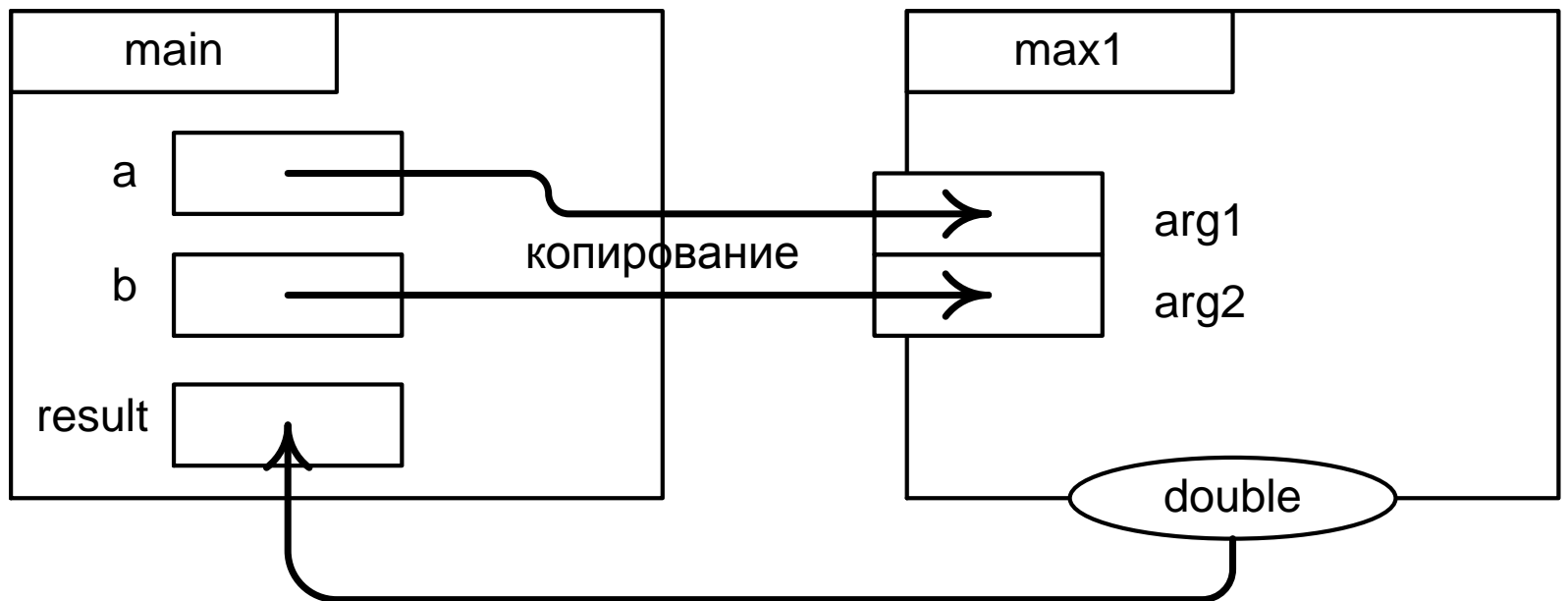
- Вызов функции из внешнего модуля

```
double max1( double, double );
void main()
{
    double a, b;
    double result;
    // ...
    result = max1( a, b );

    // ...
    exit( 0 );
}
```

Связывание по значению

□ Механизм

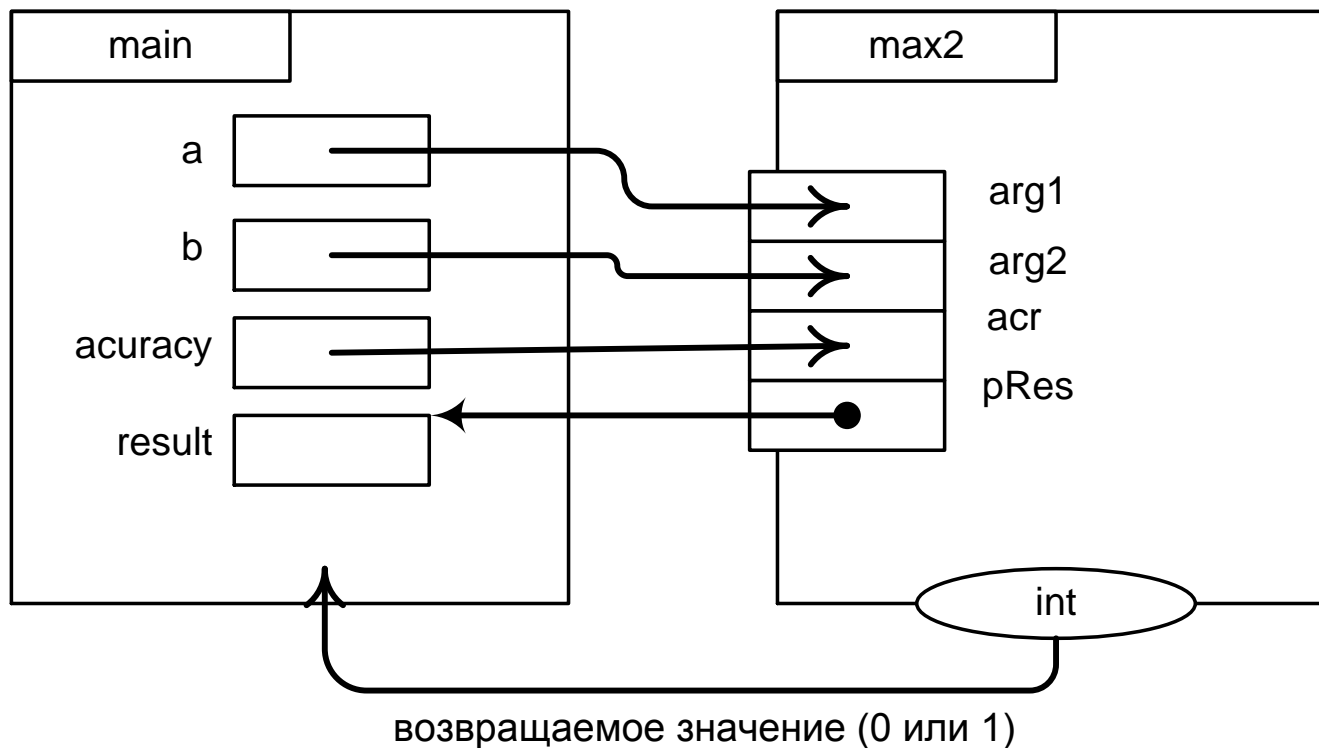


Пример: вычисление максимума двух чисел

- Изменение постановки задачи
 - Важен случай совпадения аргументов функции
- Фактически, функция должна сформировать два результата
 - есть ли среди двух значений наибольшее
 - если есть, то какое оно

Связывание с косвенной адресацией

□ Механизм



Связывание с косвенной адресацией

□ Определение функции

```
// Функция возвращает 1, если наибольшее значение имеется
//                                0, если исходные аргументы равны
int max2( double arg1, // Исходный параметр #1
          double arg2, // Исходный параметр #2
          double acr,  // Точность вычислений
          double *pRes // Адрес результата вычислений
        )
{
    if( fabs( arg1 - arg2 ) <= fabs( acr ) ) return 0;
    if( arg1 > arg2 ) *pRes = arg1;
    else              *pRes = arg2;
    return 1;
}
```


Связывание с косвенной адресацией

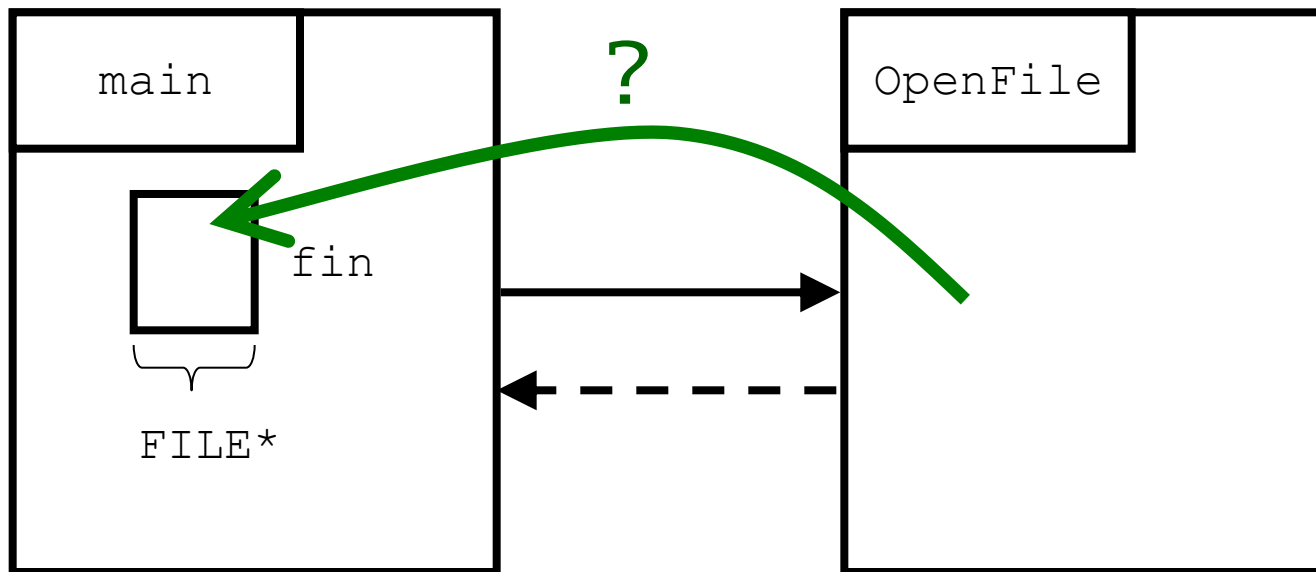
□ Вызов из внешнего модуля

```
int max2( double, double, double, double* );
void main()
{
    double a, b;
    double accuracy = 1e-9;
    double result;
    // ...

    if( max2( a, b, accuracy, &result ) ) {
        printf( "Max( %le, %le ) = %le\n", a, b, result );
    }
    // ...
}
```

Связывание с косвенной адресацией

- А если результатом работы является указатель?



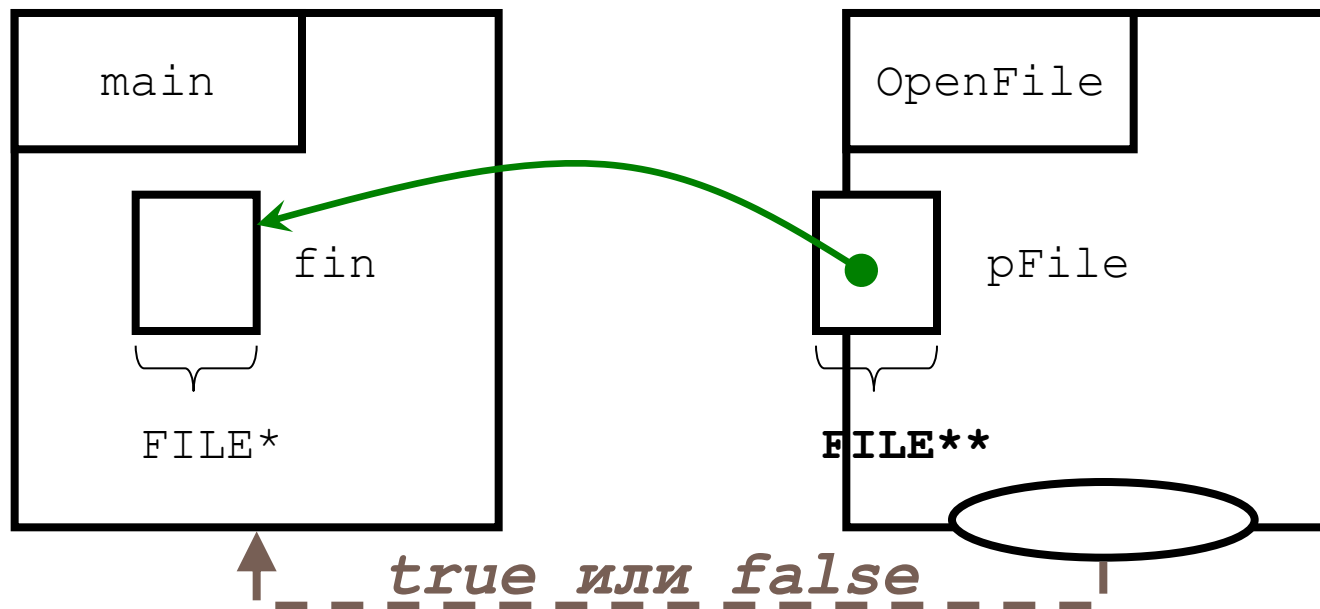
Связывание с косвенной адресацией

- Аргументом является адрес этого указателя

```
void main()
{
    FILE *fin;
    if( !OpenFile( &fin, "input.txt", "rt" ) ) exit( 1 );
    // ...
}
```

Связывание с косвенной адресацией

- Параметр: указатель на указатель



Связывание с косвенной адресацией

- Определение функции с параметром FILE**

```
bool OpenFile( FILE **pFile, char *filename, char *mode )
{
    *pFile = fopen( filename, mode );
    if( *pFile == NULL )
    {
        printf( "Can't open file \"%s\" ", filename );

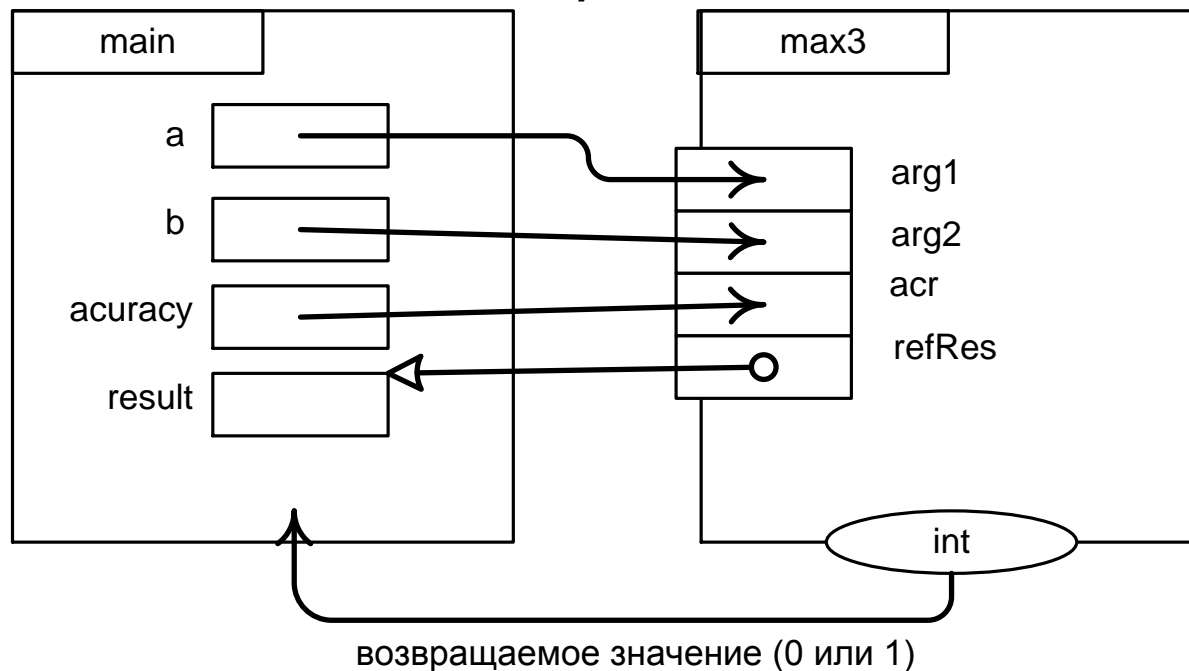
        // ...
        return false;
    }
    return true;
}
```

Связывание по ссылке

- Иногда: альтернатива использованию указателя
- Иногда: наиболее подходящее решение, позволяющее избежать создания копий аргументов
- В некоторых языках: единственный механизм, позволяющий сослаться на внешний объект данных

Связывание по ссылке

- Механизм применительно к задаче нахождения максимума



Связывание по ссылке

□ Реализация: определение функции max3

```
int max3( double  arg1, // Исходный параметр #1
          double  arg2, // Исходный параметр #2
          double  acr,  // Точность вычислений
          double& refRes // Ссылка на переменную, хранящую
                        // результат вычислений
        )
{
    if( fabs( arg1 - arg2 ) < fabs( acr ) ) return 0;
    if( arg1 > arg2 )    refRes = arg1;
    else                 refRes = arg2;
    return 1;
}
```


Связывание по ссылке

□ Реализация: вызов функции max3

```
int max3( double, double, double, double& );
void main()
{
    double a, b;
    double accuracy = 1e-9;
    double result;
    // ...

    if( max3( a, b, accuracy, result ) ) {
        printf( "Max( %le, %le ) = %le\n", a, b, result );
    }
    // ...
}
```

Механизмы связывания: рассмотрим позднее

- Передача массивов в качестве аргументов функций
- Передача структурных объектов в качестве аргументов функций
- Определение общих статических данных на уровне модуля
- Методы классов

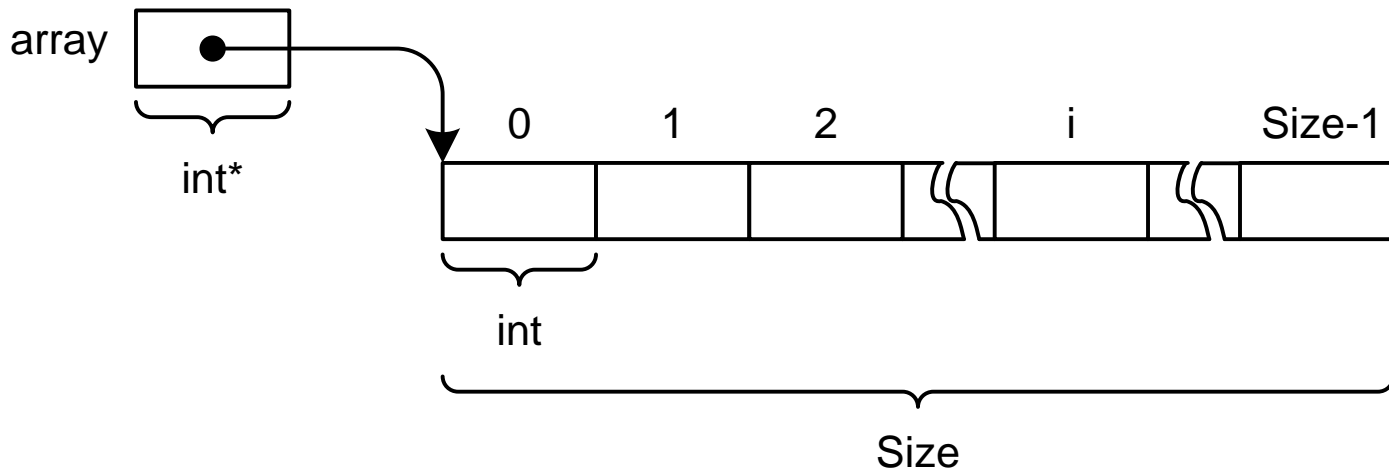
Процедуры и функции: разумные причины создания

- Снижение сложности
- Формирование понятной промежуточной абстракции
- Предотвращение дублирования кода
- Соккрытие очередности действий
- Повышение быстродействия
 - оптимизация кода выполняется в одном месте, а не в нескольких

Проектирование и использование составных типов

- Массивы и их организация в C/C++
- Размещение массивов в динамической памяти
- Структуры и структурные объекты
- Массивы структур
- Указатели: некоторые подробности

Массивы и их организация в C/C++



```
#define Size 100
void main()
{
    int array[ Size ];
    // ...
}
```

```
const int Size = 100;
void main()
{
    int array[ Size ];
    // ...
}
```

Массивы и их организация в C/C++

```
int ixFirst = 0;
int ixLast = Size - 1;

// Корректные обращения
array[ ixFirst ] = 10; // Эквивалентный код:
                        // *(array + ixFirst) = 10;
array[ ixLast ] = 10; // Эквивалентный код:
                        // *(array + ixLast) = 10;
```

Массивы и их организация в C/C++



```
// Некорректные обращения (выход за границу массива  
// НЕ контролируется транслятором!)  
array[ -1 ] = 10;    // Эквивалентный код:  
                  // *(array - 1) = 10;  
array[ Size ] = 10; // Эквивалентный код:  
                  // *(array + Size) = 10;
```

Массивы переменной длины (C99)

- Размер массива определяется неконстантным выражением
- Размер должен быть положительным целым
- После создания размер не может быть изменен
- Не может быть `static` или `extern`

```
int Reservation = 10;

void f( int customSize )
{
    int realSize = customSize + Reservation;
    int myArray[ realSize ];
    // ...
}
```


Размещение массивов в динамической памяти (C)

```
#include <stdlib.h>
```

```
int *array = NULL;    // Инициализация здесь не
                      // обязательна, но во
                      // многих случаях полезна

int size;

array = (int *) malloc( size * sizeof( int ) );
// Здесь возможна ошибка выделения памяти

if( array == NULL )
{
    // Обработка ошибки
    // ...
}
```

Размещение массивов в динамической памяти (С)

- Освобождение занятой памяти

```
free( array );  
array = NULL;    // Не обязательно, но во многих  
                // случаях полезно
```

```
// Более строгий вариант  
if( array != NULL )  
{  
    free( array );  
    array = NULL;  
}
```

Размещение массивов в динамической памяти (C++, до 1998)

```
int *array;
int size;
array = new int[ size ]; // Здесь возможна ошибка
                          // выделения памяти

if( array == NULL )
{
    // Обработка ошибки
    // ...
}
```

```
delete [] array;
```

Размещение массивов в динамической памяти (Standard C++)

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

```
int *array;
int size;
try
{
    array = new int[ size ]; // Возможно исключение
    // Операции над сформированным массивом
    // ...
}
catch( bad_alloc& )
{
    // Обработка ошибки
    // ...
}
```

Размещение массивов в динамической памяти (nothrow)

```
int *array;
int size;
array = new (nothrow) int[ size ];
if( array == 0 )
{
    // Обработка ошибки
    // ...
}
```

*Демонстрационный пример
(ООП) листинг 6.13*

Копирование массивов

```
void CopyArray( int *dest, int size, int *src )
{
    for( int ix = 0; ix < size; ix++ )
    {
        dest[ ix ] = src[ ix ];
    }
}
```

```
void CopyArray( int *dest, int size, int *src )
{
    dest = src;
}
```

Обработка массивов: задача

- Написать программу, генерирующую первую тысячу простых чисел
 - Будем записывать полученные значения в массив
 - Проверять простоту числа будем, выясняя его делимость на ранее найденные простые числа
 - Обеспечим просмотр наименьшего числа «кандидатов а делители»
 - Результат работы программы напечатаем в файл

Демонстрационный пример `ArrayOfSimpleValues`

Обработка массивов: еще одна задача

- В исходном файле находится набор целых чисел. Размер исходного набора заранее неизвестен, но не превосходит 1000. Написать программу, считывающую исходные данные в массив, упорядочивающую массив по возрастанию и печатающую в файл результатов преобразованный массив

Демонстрационный пример 001_ArraySort

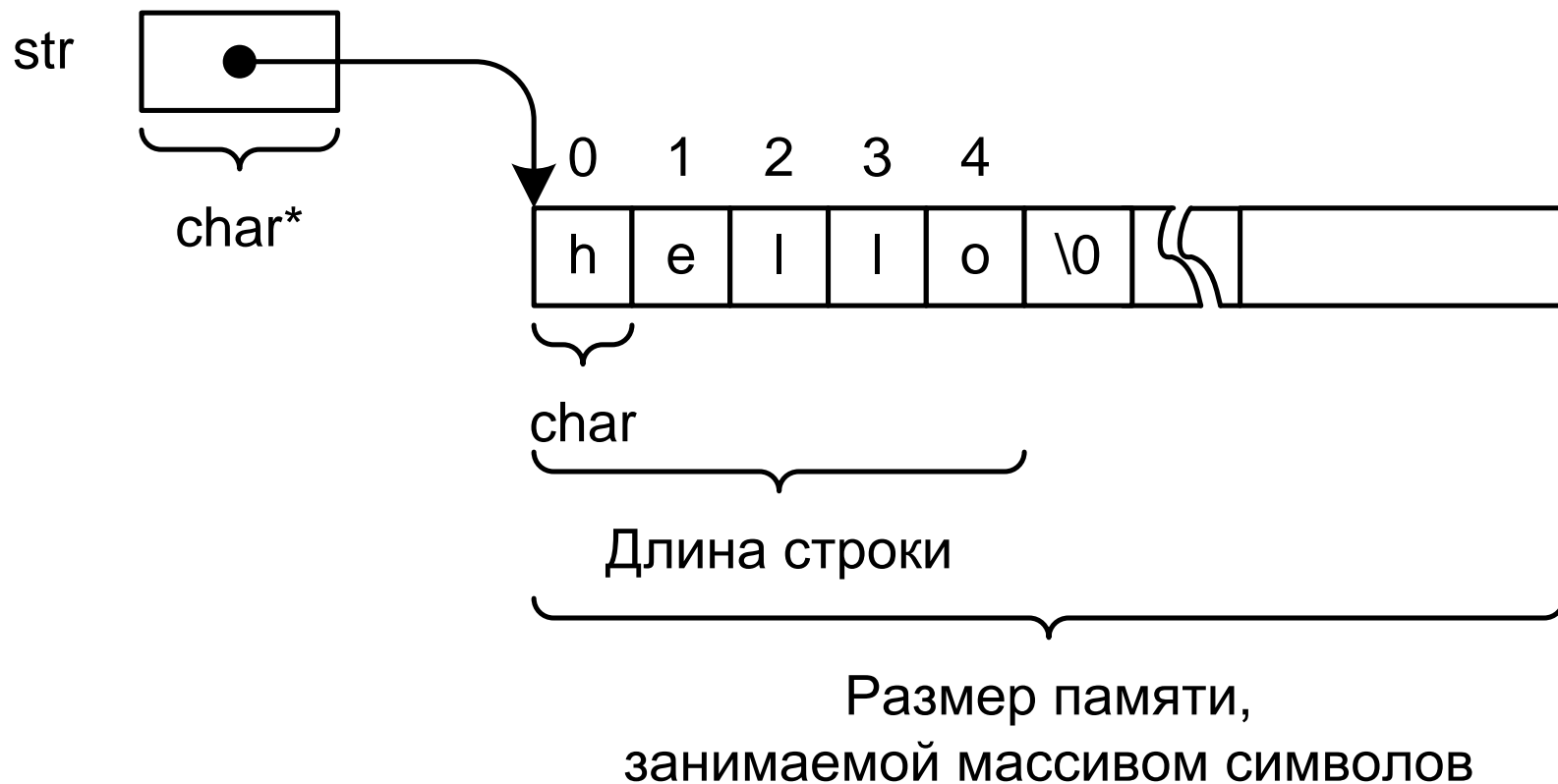
Обработка массивов: задача*

- В исходном файле находится набор целых чисел. Размер исходного набора **заранее неизвестен**. Написать программу, считывающую исходные данные в массив, упорядочивающую массив по возрастанию и печатающую в файл результатов преобразованный массив

Демонстрационный пример 002_ArraySort

Демонстрационный пример 003_ArraySort

Строки символов ANSI C



Строки символов ANSI C

```
// STRingLength
// Вычисление длины строки
int StrLength ( const char *s )
{
    char *pChar = s;
    int    length = 0;

    while( *pChar != '\0' )
    {
        length++;
        pChar++;
    }

    return length;
}
```

Строки символов ANSI C

```
// STRingLength (optimized)
// Вычисление длины строки (оптимизированная версия)
int StrLength ( const char *s )
{
    int    length = 0;

    while( *s++ ) length++;

    return length;
}
```

Строки символов ANSI C

```
// STRingCOPY
// Копирование строк
char *StrCopy( char *dst, const char *src )
{
    const char *pCharSrc = src;
    char      *pCharDst = dst;

    while( *pCharDst = *pCharSrc )
    {
        pCharSrc++;
        pCharDst++;
    }

    return dst;
}
```

Строки символов ANSI C

```
// STRingCOPY (optimized)
// Копирование строк (оптимизированная версия)
char *StrCopy( char *dst, const char *src )
{
    char *dstCopy = dst;

    while( *dst++ = *src++ );

    return dstCopy;
}
```

Строки символов ANSI C

- Заголовочный файл `string.h`

- Функции

 - `strcmp`

 - `strlen`

 - `strcpy`

 - `strcat`

 - `strchr`

 - ...

Строки символов ANSI C

□ Формирование строки ANSI C

```
#include <stdio.h>
#include <ctype.h>

// GET WORD from file
// Чтение слова из файла
int GetWord( FILE *f, char *word, int limit )
{
    char ch = fgetc( f );
    int ixLit = 0;

    // Пропуск ведущих пробелов
    while( isspace( ch ) ) ch = fgetc( f );
    if( ch == EOF ) return EOF;
```


Строки символов ANSI C

```
// Чтение слова из букв
while( !isspace( ch ) && ch != EOF )
{
    if( !isalpha( ch ) ) return 1;

    if( ixLit < limit ) word[ ixLit++ ] = ch;

    ch = fgetc( f );
}
word[ ixLit ] = '\\0';

return 0;
}
```

Самостоятельное изучение

- Практика использования функций обработки строк
- Операция «запятая»
- Операции над указателями (арифметические, операции отношения)

Области видимости и классы памяти

- Глобальная область видимости
- Область видимости модуля
- Область видимости функции
- Область видимости блока
- Область видимости прототипа

*Демонстрационный пример
001_Scope*

Средства препроцессора языка С и их использование в коде на С++

- Макроопределения и макроподстановки
- Директивы условной компиляции
- Управление работой компилятора
- Классификация Страуструпа
 - ▣ полезные
 - ▣ бесполезные
 - ▣ опасные
- Код на С++: ограниченное использование препроцессора С

Макроопределения и макроподстановки

```
#define MAX_SIZE 1000
```

```
#define SUCCEEDED(x) \
    (x) != 0xCC
```

```
#define Cube (x) ((x)*(x)*(x))
```

```
#define GetEntry(key, index) { \
    index=hash(key); \
    index=min(index, MAX_INDEX); \
    index=max(index, MIN_INDEX); \
}
```

**Обратите
внимание на **

**Обратите
внимание на ()**

**Обратите
внимание на {}**

Директивы условной компиляции

```
// SomeHeader.h
#ifndef _SOMEHEADER_H_
#define _SOMEHEADER_H_
    // Содержимое файла
    // ...
#endif
```

*Для отсутствия
повторного
подключения*

```
#define TARGET ...
#if TARGET == _WINDOWS
    // Код для Windows
#elif TARGET == _LINUX
    // Код для Linux
#elif TARGET == _FREEBSD
    // Код для Free BSD
#endif
```

*Для различных
вариантов кода*

Управление работой компилятора

```
#if !defined(__cplusplus)
#error C++ compiler required.
#endif
```

*Для остановки
компиляции при
ошибках
совместимости*

```
#include "FULLNAME"  
#include <FULL-NAME>
```

- "" Сначала – в текущем и вложенных, затем – в каталогах определяемых /I, далее – в каталогах, определяемых переменной окружения **INCLUDE**
- <> Сначала – в стандартных каталогах (ключ компилятора /I), далее – в каталогах, определяемых переменной окружения **INCLUDE**

*Для подключения
заголовочных
файлов*

Управление работой компилятора

```
#pragma ...
```

```
#pragma optimize( "", off )
```

- Поддержка настроек, уникальных для целевой платформы и компилятора
- Примеры:
 - ▣ управление памятью
 - ▣ управление вызовами функций
 - ▣ управление размещением функций и данных в объектном коде
 - ▣ управление локализацией
 - ▣ ...

*Для управления
специфическими
настройками
компилятора*

*Пример:
отключение
оптимизации
кода*

Альтернативы макросам в C++

- Типизированные константы (`const`)
- Встраиваемые функции (`inline`)
- Шаблоны функций и типов (`template`)
- Перечислимые типы (`enum`)
- Директива `typedef` для замены одного типа другим в некотором контексте

Структурные типы

- Определение типов в связи с моделированием объектов предметной области
- Определение структурных типов: различия реализаций С и С++
- Использование структурных типов

Структуры: задача

- В исходном файле находится набор данных о координатах точек на плоскости. Написать программу, подсчитывающую, сколько треугольников можно построить, используя тройки точек в качестве вершин треугольника.

*Демонстрационный пример
001_TriCounter*

Другие важные для практики типы

- Объединения
- Поля битов

Объединения

- использование одной и той же области памяти различным образом
 - ▣ для более удобного доступа к различным участкам этой области
 - ▣ для преобразования типа (редко)

*Демонстрационный пример
001_Uword*

Поля битов

- структуры могут содержать поля, занимающие меньше места, чем объект интегрального типа
- позволяют упростить манипулирование отдельными битами кодового представления

Поля битов

- Синтаксис определяется дополнением к определению структур

[спецификатор-типа] [имя] : конст-выражение



Интегральный тип!

	char
	short
unsigned +	int
	long
	enum



0..max

*max – не больше, чем
может вместить
базовый тип*

Поля битов

□ Простой пример

```
struct TestBitField
{
    unsigned short a : 4;
    unsigned short b : 5;
    unsigned short c : 6;

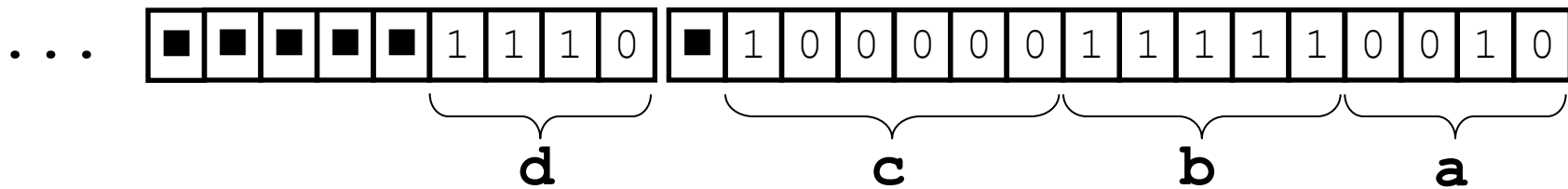
    unsigned short d : 4;
};

TestBitField x;
```


Поля битов

- Порядок следования битов в пределах представления, определяемого базовым типом

```
x.a = 2;  
x.b = 31;  
x.c = 32;  
x.d = 14;
```



Демонстрационный пример 001_bitFields

Поля битов

- Если очередной элемент поля битов не может уместиться в пределах памяти, отводимой под базовый тип, размещение начинается со следующего слова памяти базового типа (выравнивание)
- Возможность явного выравнивания (безымянное поле битов, в том числе поле битов длины 0)

Поля битов

- Наглядная работа с «упакованной» информацией

```
enum ColorType {
    White, Red, Yellow, Green, Blue, Violet, Cyan, Black
};

struct VideoMemTextSymbol {
    unsigned int ch : 8;

    ColorType color : 3;
    unsigned int intense : 1;
    ColorType bkGrndColor : 3;
    unsigned int blinking : 1;
};
```

*Демонстрационный пример
002_BitFields*

Поля битов: ограничения

- Не разрешены указатели на поля битов
- Не разрешается использовать операцию определения адреса (&) над отдельным полем битов
- Нельзя определить массив полей битов
- Функции не могут возвращать значение отдельного поля битов