

# ИЗМЕНЕНИЕ ЗНАЧЕНИЯ ПАРАМЕТРА

Для изменения значения параметра функция должна знать адрес памяти параметра. Чтобы сообщить функции адрес параметра, ваши программы должны использовать оператор адреса C++ (&). Следующий вызов функции иллюстрирует, как программа будет использовать оператор адреса для передачи адресов переменных *big* и *small* в функцию *change\_values*:

```
change_values ( &big, &small ); --> Передача параметров по адресу
```

Внутри функции вы должны сообщить C++ , что программа будет передавать параметры с помощью адреса. Для этого вы объявляете *переменные-указатели*, предваряя имя каждой переменной звездочкой, как показано ниже:

```
void change_values ( int. *big, int. *small ) ---> Указатель на тип int
```

Переменная-указатель представляет собой переменную, которая содержит адрес памяти. Внутри функции вы должны сообщить C++ , что функция работает с адресом параметра. Для этого вы предваряете имя параметра звездочкой, как показано ниже:

```
*big = 1001;  
*small = 1001;
```

## ИЗМЕНЕНИЕ ЗНАЧЕНИЯ ПАРАМЕТРА

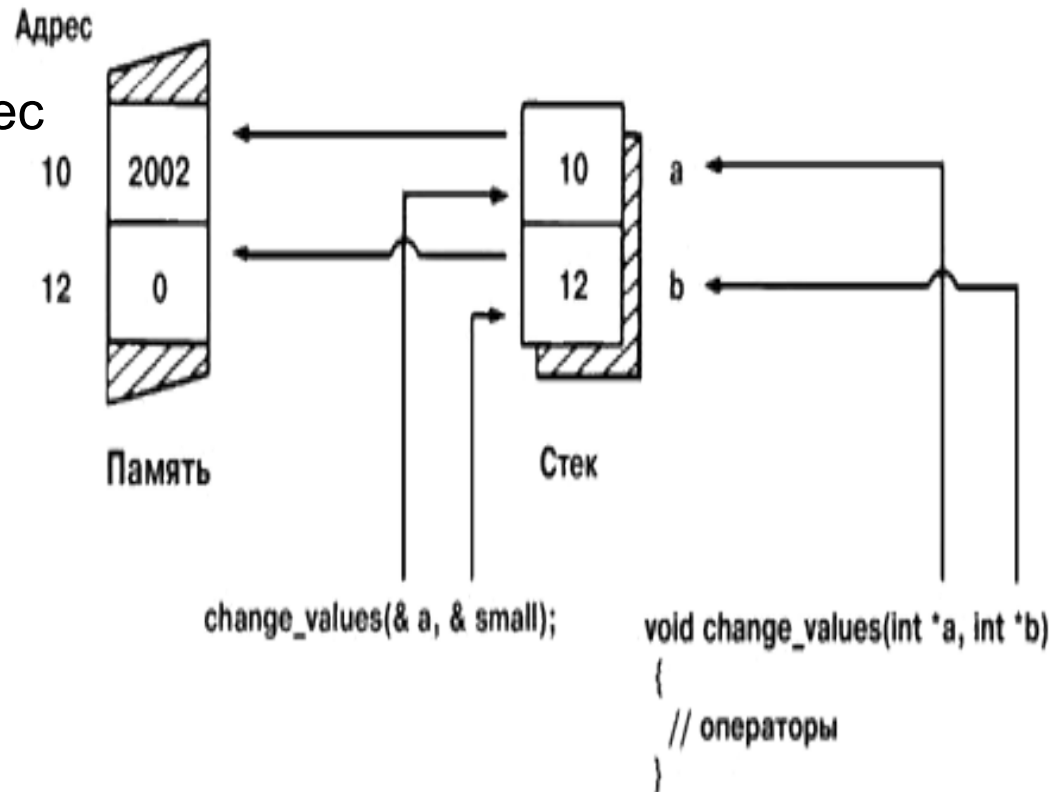
- ◆ `#include <iostream.h>`
- ◆ `void change_values (int *a, int *b)`
- ◆ 

```
{
    *a = 1001;
    *b = 1001;
    cout << "Значения в функции display_values" << "
равны " << *a << " и " << *b << endl;
}
```
- ◆ `void main(void)`
- ◆ 

```
{
    int big = 2002, small = 0;
    cout << "Значения перед функцией " << big << " и "
<< small << endl;
    change_values(&big, &small);
    cout << "Значения после функции " << big << " и "
<< small << endl;
}
```

## ИЗМЕНЕНИЕ ЗНАЧЕНИЯ ПАРАМЕТРА

- функция имеет доступ к ячейке памяти каждой переменной. Если вы передаете параметры по адресу, С++ помещает адрес каждой переменной в стек



# Изменение значений параметров в функциях

- ◆ Для изменения значения параметра в функции, функция должна знать адрес параметра в памяти. Следовательно, ваша программа должна передать адрес параметра с помощью оператора адреса C++ :
- ◆ **`some_function(&some_variable);`**
- ◆ Внутри функции вы должны сообщить C++ , что функция будет работать с адресом памяти (указателем). Для этого при объявлении вы предваряете имя параметра звездочкой:
- ◆ **`void some_function(int *some_variable);`**
- ◆ Далее внутри функции вы должны употреблять звездочку перед именем переменной:
- ◆ **`*some_variable = 1001;`  
`cout << *some_variable;`**

# Изменение значений параметров в функциях

- ◆ Если ваша программа передает указатели на параметры, параметры могут быть любого типа, например `int`, `float` или `char`. Функция, которая использует указатели, объявляет переменные соответствующего типа, предваряя имя каждой переменной звездочкой, подтверждающей, что такая переменная является указателем.

# Изменение значений параметров в функциях

- ◆ `#include <iostream.h>`
- ◆ `void swap_values(float *a, float *b)`
- ◆ 

```
{  
    float temp;  
    temp = *a;  
    *a = *b;  
    *b = temp;  
}
```
- ◆ `void main(void)`
- ◆ 

```
{  
    float big = 10000.0;  
    float small = 0.00001;  
    swap_values(&big, &small);  
    cout << "Big содержит " << big << endl;  
    cout << "Small содержит " << small << endl;  
}
```

# Локальные переменные и область видимости

- ◆ *Локальная переменная* представляет собой переменную, определенную внутри функции.
- ◆ *void some\_function(void)*
- ◆ 

```
{  
    int count;  
    float result;  
}
```

# Локальные переменные

- ◆ используем функцию со *und\_speaker*, которая заставляет играть встроенный компьютерный динамик столько раз, сколько указано параметром *beeps*. Внутри функции со *und\_speaker* локальная переменная *counter* хранит количество звуков, издаваемых динамиком:
- ◆ `#include <iostream.h>`
- ◆ `void sound_beeps(int beeps)`
- ◆ 

```
{  
    for (int counter = 1; counter <= beeps; counter++) cout  
    << '\a';  
}
```
- ◆ `void main(void)`
- ◆ 

```
{  
    sound_beeps(2);  
    sound_beeps(3);  
}
```



# ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ

- ◆ `int`  
`some_global_variable; --`  
*-> Объявление глобальной переменной*
- ◆ `void main(void)`
- ◆ `{`  
`// Здесь должны быть операторы`  
`программы`  
`}`

# ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ

- ◆ используем глобальную переменную именем *number*. Каждая функция в программе может использовать (или изменять) значение глобальной переменной. В данном случае каждая функция выводит текущее значение этой переменной, а затем увеличивает это значение на единицу:
- ◆ `#include <iostream.h>`
- ◆ `int number = 1001;`
- ◆ `void first_change(void)`
- ◆ 

```
(  
    cout << "значение number в first_cbange " << number << endl;  
    number++;  
)
```
- ◆ `void second_change(void)`
- ◆ 

```
{  
    cout << "значение number в second_change " << number << endl;  
    number++;  
}
```
- ◆ `void main(void)`
- ◆ 

```
{  
    cout << "значение number в main " << number << endl;  
    number++;  
    first_change ();  
    second_change();  
}
```

# Локальные переменные и область видимости

- ◆ Если необходимо обратиться к глобальной переменной, чье имя конфликтует с именем локальной переменной. В таких случаях ваши программы могут использовать *глобальный оператор разрешения C++ (::)*, если вы хотите использовать глобальную переменную.
- ◆ `number = 1001; // обращение к локальной переменной`
- ◆ `::number = 2002; // Обращение к глобальной переменной`

# Локальные переменные и область ВИДИМОСТИ

- ◆ `#include <iostream.h> int number = 1001; //`  
Глобальная переменная
- ◆ `void show_numbers(int number)`
- ◆ `{`  
    `cout << "Локальная переменная number" <<`  
    `" содержит " << number << endl;`  
    `cout << "Глобальная переменная number" <<`  
    `" содержит " << ::number << endl;`  
    `}`
- ◆ `void main(void)`  
    `{`  
        `int some_value = 2002;`  
        `show_numbers(some_value) ;`  
    `}`

# Перегрузка функций

- ◆ Перегрузка функций позволяет вашим программам определять несколько функций с одним и тем же именем и типом возвращаемого значения.
- ◆ программа перегружает функцию с именем *add\_values*. Первое определение функции складывает два значения типа *int*. Второе определение функции складывает три значения. В процессе компиляции C++ корректно определяет функцию, которую необходимо использовать:

```
◆ #include <iostream.h>
◆ int add_values(int a,int b)
◆ {
   return(a + b);
}
◆ int add_values (int a, int b, int c)
◆ (
   return(a + b + c);
)
◆ void main(void)
◆ {
   cout << "200 + 801 = " << add_values(200, 801) << endl;
   cout << "100 + 201 + 700 = " << add_values(100, 201, 700) << endl;
}
```

# Перегрузка функций

```
◆ #include <iostream.h>
◆ void show_message(void)
◆ {
  cout << "Стандартное сообщение: " << "Учимся программировать на
  C++" << endl;
}
◆ void show_message(char *message)
◆ {
  cout << message << endl;
}
◆ void show_message(char *first, char *second)
◆ {
  cout << first << endl;
  cout << second << endl;
}
◆ void main(void)
◆ {
  show_message();
  show_message("Учимся программировать на языке C++!");
  show_message("В C++ нет предрассудков!", "Перегрузка - это круто!");
}
```

- ◆ *Перегрузка функций C++ позволяет вашим программам определять несколько функций с одним и тем же именем. Перегруженные функции должны возвращать значения одинакового типа\*, но могут отличаться количеством и типом параметров. До появления перегрузки функций в C++ программисты языка C должны были создавать несколько функций с почти одинаковыми именами*

# Использование ссылок в C++

- ◆ *Ссылка* C++ позволяет создать псевдоним (или второе имя) для переменных в вашей программе. Для объявления ссылки внутри программы укажите знак амперсанда (&) непосредственно после типа параметра. Объявляя ссылку, вы должны сразу же присвоить ей переменную, для которой эта ссылка будет псевдонимом
- ◆
- ◆ `int& alias_name = variable; //---> Объявление ссылки`
- ◆
- ◆ После объявления ссылки ваша программа может использовать или переменную , или ссылку:
- ◆ 

```
alias_name = 1001;  
variable = 1001;
```

# Использование ссылок в C++

- ◆ создаем ссылку с именем *alias\_name* и присваивает псевдониму переменную *number*. Далее программа использует как ссылку, так и переменную:

- ◆ `#include <iostream.h>`
- ◆ `void main(void)`
- ◆ 

```
{
    int number = 501;
    int& alias_name = number; // Создать ссылку
    cout << "Переменная number содержит " << number <<
endl;
    cout << "Псевдоним для number содержит " << alias_name
<< endl;
    alias_name = alias_name + 500;
    cout << "Переменная number содержит " << number <<
endl;
    cout << "Псевдоним для number содержит " << alias_name
<< endl;
}
```



# Использование ссылок в C++

- ◆ **Объявление ссылки**
- ◆ **Ссылка C++ представляет собой псевдоним (второе имя), которое ваши программы могут использовать для обращения к переменной. Для объявления ссылки поставьте амперсанд (&) сразу же после типа переменной, а затем укажите имя ссылки, за которым следует знак равенства и имя переменной, для которой ссылка является псевдонимом: `float& salary_alias = salary;`**

# ИСПОЛЬЗОВАНИЕ ССЫЛОК В КАЧЕСТВЕ ПАРАМЕТРОВ

- ◆ присваиваем ссылку с именем *number\_alias* переменной *number*. Программа передает ссылку на переменную в функцию *change\_value*, которая присваивает переменной значение 1001:
- ◆ 

```
#include <iostream.h>
```
- ◆ 

```
void change_value(int &alias)
```
- ◆ 

```
{
```
- ◆ 

```
    alias = 1001;
```
- ◆ 

```
}
```
- ◆ 

```
void main(void)
```
- ◆ 

```
{
```
- ◆ 

```
    int number;
```
- ◆ 

```
    int& number_alias = number;
```
- ◆ 

```
    change_value(number_alias);
```
- ◆ 

```
    out << "Переменная number содержит " << number << endl;
```
- ◆ 

```
}
```

# ИСПОЛЬЗОВАНИЕ ССЫЛОК В КАЧЕСТВЕ ПАРАМЕТРОВ

- ◆ используем ссылки на значения с плавающей точкой для упрощения функции:

- ◆ 

```
#include <iostream.h>
```
- ◆ 


```
void swap_values(float& a, float& b)
{
    float temp;
    temp = a;
    a = b;
    b = temp;
}
```
- ◆ 

```
void main(void)
```
- ◆ 

```
{
    float big = 10000.0;
    float small = 0.00001;
    float& big_alias = big;
    float& small_alias = small;
    swap_values(big_alias, small_alias);
    cout << "Big содержит " << big << endl;
    cout << "Small содержит " << small << endl;
}
```

# ПРАВИЛА РАБОТЫ СО ССЫЛКАМИ

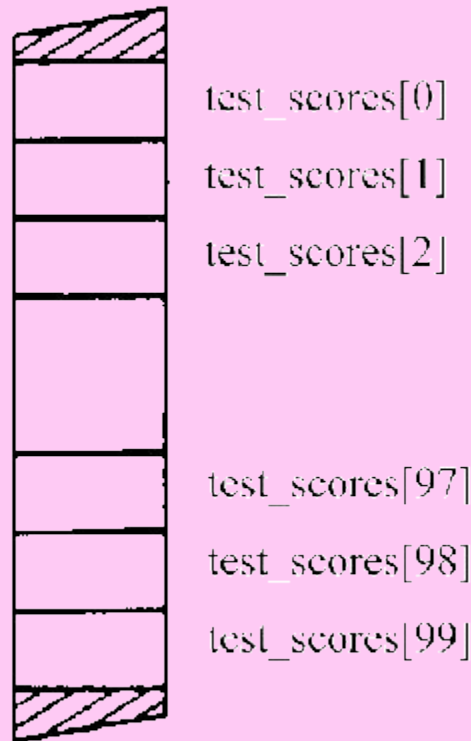
Вы не можете получить адрес ссылки, используя оператор адреса C++.

- Вы не можете присвоить ссылке указатель.
  - Вы не можете сравнить значения ссылок, используя операторы сравнения C++.
  - Вы не можете выполнить арифметические операции над ссылкой, например добавить смещение.
  - Вы не можете изменить ссылку.
- 

# Хранение значений в массивах

- ◆ *Массив* представляет собой переменную, способную хранить одно или несколько значений.
- ◆ создаем массив с именем *test\_scores*, который может вмещать 100 целых значений для тестовых очков:
- ◆
- ◆ `_____` *Тип массива*
- ◆ `int test_scores [100]; //----->` *Размер массива*
- ◆ ***float part\_cost[50];***  
***int employee\_age[100];***  
***float stock\_prices[25];***

# массива



- ♦ для обращения к первому элементу массива *test\_scores* вы должны использовать значение индекса 0. Для обращения ко второму элементу используйте индекс 1. Подобно этому, для обращения к третьему элементу используйте индекс 2. Как показано на рис. 16.1, первый элемент массива всегда имеет индекс 0, а значение индекса последнего элемента на единицу меньше размера массива:

# Использование индекса для обращения к элементам массива

```
□ #include <iostream.h>
□ void main(void)
□ {
    int values[5]; // Объявление массива
    values[0] = 100;
    values[1] = 200;
    values[2] = 300;
    values[3] = 400;
    values [4] = 500;
    cout << "Массив содержит следующие значения" << endl;
    cout << values [0] << ' ' << values [1] << ' ' << values [2] << ' ' << values
[4] << endl;
}
```

```
□ #include <iostream.h>
□ void main (void)
□ {
    int values[5]; // Объявление массива int i;
    values[0] = 100;
    values[1] = 200;
    values[2] = 300;
    values[3] = 400;
    values[4] = 500;
    cout << "Массив содержит следующие значения" << endl;
    for (i = 0; i < 5; i++) cout << values [i] << ' ';
}
```

# ИНИЦИАЛИЗАЦИЯ МАССИВА ПРИ ОБЪЯВЛЕНИИ

- `int values[5] = { 100, 200, 300, 400, 500 };`
- `float salaries[3] = { 25000.00, 35000.00, 50000.00 };`
- `int values[5] = { 100, 200, 300 };`
- `int numbers[] = { 1, 2, 3, 4 };`



# ПЕРЕДАЧА МАССИВОВ В ФУНКЦИИ

- ◆ Программы будут передавать массивы в функции точно так же, как и любые другие переменные. Функция может инициализировать массив, прибавить к массиву значения или вывести элементы массива на экран. Когда вы передаете массив в функцию, вы должны указать тип массива. Нет необходимости указывать размер массива. Вместо этого вы передаете параметр
- ◆ *number\_of\_elements*, который содержит количество элементов в массиве:
- ◆ `void some_function(int array[], int number_of_elements);`

# ПЕРЕДАЧА МАССИВОВ В ФУНКЦИИ

- ◆ передаем массивы в функцию *show\_array*, которая использует цикл *for* для вывода значений массивов:
- ◆ `#include <iostream.h>`
- ◆ `void show_array (int array [] , int number_of_elements)`
- ◆

```
{
    int i;
    for (i = 0; i < number_of_elements; i++) cout << array[i] << '
';
    cout << endl;
}
```
- ◆ `void main(void)`
- ◆

```
{
    int little_numbers[5] = {1,2,3,4,5};
    int big_numbers[3] = { 1000, 2000, 3000 };
    show_array(little_numbers, 5);
    show_array(big_numbers, 3);
}
```

# Пример простой CLR-программы.

- ◆ *Указатель* - это переменная, которая содержит в себе адрес другой переменной. Если мы объявим `int a`, то присвоение `Pa=&a` даст адрес этой переменной (номер ячейки памяти, где она хранится). Что бы было понятно, приведу такую аналогию: сама переменная `a` - это содержимое некоего ящичка с номером. `&a` (или `Pa` после присваивания) - это номер ящичка. В данном случае `Pa` - это уже бумажка с номером ящичка переменной `a`, но уже положенная в другой ящик. Операции взятия адреса соответствует обратная операция - определения содержимого по адресу. Например, мы можем написать `b=*Pa` (после того как написали `Pa=&a`) и в `b` у нас будет содержимое переменной `a`.
- ◆ `<Тип переменной> * <Имя указателя>`
- ◆ `char * P;`
- ◆ `void * ptr;`

# Пример простой CLR-программы

- ◆ Если мы объявили указатель на `char`, то конструкция `*P` у нас будет первым символом массива, а после совершения операции `P++` `*P` возвратит уже второй символ.
- ◆ В `VC++`, начиная с версии `VC++ 2005` появилась система CLR, специально созданная для безопасной и более удобной работы с указателями. В ней указатели подразделяются на следующие типы:
  - ◆ Регулируемые указатели.
  - ◆ Нерегулируемые указатели.
  - ◆ Нерегулируемые указатели функций.

# Пример простой CLR-программы

```
◆ // les16_1.cpp: главный файл проекта.  
◆  
◆ #include "stdafx.h"  
◆ #include <conio.h>  
◆  
◆ ref struct Message {  
◆     System::String ^sender, ^receiver, ^date;  
◆ };  
◆  
◆ using namespace System;  
◆  
◆ int main(array<System::String ^> ^args)  
◆ {  
◆     Message ^M=gcnew Message;  
◆     M->sender="the message to all";  
◆     M->date="15.12.2010";  
◆     Console::WriteLine(M->sender);  
◆     Console::WriteLine(M->date);  
◆     Console::WriteLine(L"Привет, мир!");  
◆     _getch();  
◆     return 0;  
◆ }
```

Начальная страница - Visual C++ 2008, экспресс-выпуск

Файл Правка Вид Сервис Окно Справка

Создать ▶

Открыть ▶

Закрывать

Закрывать решение

Сохранить выбранные элементы Ctrl+S

Сохранить выбранные элементы как...

Сохранить все Ctrl+Shift+S

Параметры страницы...

Печать... Ctrl+P

Последние файлы ▶

Последние проекты ▶

Выход

Проект... Ctrl+Shift+N

Файл... Ctrl+N

Проект из существующего кода...

Последние проекты

- les16\_1
- les4\_3
- les4\_3
- les4\_3
- les4\_3

# Создать проект

Типы проектов:

- Visual C++
  - CLR
  - Win32
  - Общие

Шаблоны:

## Установленные шаблоны Visual Studio

- Библиотека классов
- Пустой проект CLR

- Консольное приложение CLR
- Приложение Windows Forms

## Мои шаблоны

- Найти шаблоны в Интернете...

Проект по созданию консольного приложения

Имя:

Расположение:

Обзор...

Имя решения:

Создать каталог для решения

OK

Отмена

Обозреватель решений - proba



Решение "proba" (проектов: 1)

proba

- Заголовочные файлы
  - resource.h
  - stdafx.h
- Файлы исходного кода
  - AssemblyInfo.cpp
  - proba.cpp
  - stdafx.cpp
- Файлы ресурсов
  - app.ico
  - app.rc
  - ReadMe.txt

proba.cpp

Начальная страница

(Глобальная область)

```
// proba.cpp: главный файл проекта.  
  
#include "stdafx.h"  
  
using namespace System;  
  
int main(array<System::String ^> ^args)  
{  
    Console::WriteLine(L"Здравствуй, мир!");  
    return 0;  
}
```