

# Работа с динамическими данными в C++

## Лекция 1

# Переменные

**Переменная** - это ячейка в памяти компьютерного устройства, которая имеет имя и хранит некоторое значение.

Все данные в памяти ЭВМ представляют собой последовательность 0 и 1, поэтому для интерпретации данных компилятору важно знать их **тип**.

Тип данных определяет размер ячейки в памяти, необходимый для их хранения.

# Инициализация переменных

**Инициализация переменной** – присваивание «первоначального» значения переменной при объявлении, подготовка переменной к «работе».

Одновременно в памяти хранится только одно значение переменной, но его можно «перезаписать» т.е. изменить на другое подходящего типа.

# Основные способы инициализации переменных - 1

- «**Инициализация по умолчанию**». Если не присвоить переменной значение при объявлении, то в ней будет находиться «мусор из памяти» - та последовательность 0 и 1, которая размещалась по этому адресу ранее.

```
int a;  
cout<<" a = "<<a<<endl;
```

# Основные способы инициализации переменных - 2

- «**Копирующая инициализация**». Присвоение переменной значения при объявлении с помощью операции присваивания (=).

```
int a = 10;  
cout<<" a = "<<a<<endl;
```

# Основные способы инициализации переменных - 3

- «**Прямая инициализация**». Присваивание значения с помощью круглых скобок.

```
int a (10);  
cout<<" a = "<<a<<endl;
```

# Основные способы инициализации переменных - 4

- «**Uniform-инициализация**» или агрегатная инициализация . Используется в том числе со списками значений, подразумевает проверку некорректного присваивания (проверяются типы).

```
int a {10};  
cout<<" a = "<<a<<endl;
```

```
double mas[]={1.1, 2.2, 3.3};
```

## Переменные различаются:

- По области видимости, области действия, времени жизни (локальные и глобальные).
- По расположению в сегментах памяти (локальные хранятся в стеке, глобальные - в сегменте данных, динамические - в heap).
- По автоматизации работы с памятью (автоматические и динамические).



# Классификация переменных - 1

- **Локальные переменные** видимы и существуют внутри той программной единицы, где объявлены (в теле функции, в теле цикла или условного оператора, в любых парных фигурных скобках).
- **Глобальные переменные** видимы всюду из программы, существуют все время работы программы, объявляются вне функций.

## Классификация переменных - 2

- **Автоматические переменные** создаются и уничтожаются автоматически.
- **Динамические переменные** создаются и уничтожаются программистом вручную.

Данные, на которые указывают динамические переменные, хранятся в специальном сегменте памяти – в heap.

## Классификация переменных - 3

- **Динамические переменные** в C++ объявляются как указатели на некоторый адрес в памяти.

**Указатель в языке C++** – это переменная, значением которой является адрес другой переменной.

# Работа с автоматической переменной

```
double d {123.4567};  
cout<<"\t d = "<<d<<endl  
    <<"\t address d: "<<&d<<endl  
    <<"\t size of d: "<<sizeof(d)<<" byte"  
    <<endl;
```

# Работа с автоматической переменной

```
double d {123.4567};  
cout<<"\t d = "<<d<<endl  
  <<"\t address d: "<<&d<<endl  
  <<"\t size of d: "<<sizeof(d)<<" byte"  
  <<endl;
```

Получение адреса  
автоматической  
переменной

Обращение к  
значению  
автоматической  
переменной

# Работа с автоматической переменной

```
double d {123.4567};  
cout<<"\t d = "<<d<<endl  
  <<"\t address d: "<<&d<<endl  
  <<"\t size of d: "<<sizeof(d)<<" byte"  
  <<endl;
```

Получение адреса  
автоматической  
переменной

Обращение к  
значению  
автоматической  
переменной

```
d = 123.457  
address d: 0x28ff00  
size of d: 8 byte
```

# Работа с динамической переменной - 1

```
double *pd1=&d;  
cout<<"\t pd1 = "<<*pd1<<endl  
    <<"\t address pd1: "<<pd1<<endl  
    <<"\t size of data pd1: "<<sizeof(*pd1)<<" byte"  
    <<endl  
    <<"\t size of address pd1: "<<sizeof(pd1)<<" byte"  
    <<endl;
```

# Работа с динамической переменной - 1

```
double *pd1=&d;  
cout<<"\t pd1 = "<<*pd1<<endl  
  <<"\t address pd1: "<<pd1<<endl  
  <<"\t size of data pd1: "<<sizeof(*pd1)<<" byte"  
  <<endl  
  <<"\t size of address pd1: "<<sizeof(pd1)<<" byte"  
  <<endl;
```

Обращение к  
значению, на которое  
указывает  
динамическая  
переменная.  
\* - операция  
разыменования

Работа с адресом



# Работа с динамической переменной - 1

```
double *pd1=&d;  
cout<<"\t pd1 = "<<*pd1<<endl  
    <<"\t address pd1: "<<pd1<<endl  
    <<"\t size of data pd1: "<<sizeof(*pd1)<<" byte"  
    <<endl  
    <<"\t size of address pd1: "<<sizeof(pd1)<<" byte"  
    <<endl;
```

Обращение к  
значению, на которое  
указывает  
динамическая  
переменная.  
\* - операция  
разыменования

Работа с адресом

```
pd1 = 123.457  
address pd1: 0x28ff00  
size of data pd1: 8 byte  
size of address pd1: 4 byte
```

## Работа с динамической переменной - 2

```
pd1=new double;  
*pd1=-9.876;  
cout<<"\t pd1 = "<<*pd1<<endl  
    <<"\t address pd1: "<<pd1<<endl  
    <<"\t size of data pd1: "<<sizeof(*pd1)<<" byte"  
    <<endl  
    <<"\t size of address pd1: "<<sizeof(pd1)<<" byte"  
    <<endl;  
delete pd1;  
pd1=NULL;
```

## Работа с динамической переменной - 2

```
pd1=new double;
```

```
*pd1=-9.876;
```

```
cout<<"\t pd1 = "<<*pd1<<endl
```

```
<<"\t address pd1: "<<pd1<<endl
```

```
<<"\t size of data pd1: "<<sizeof(*pd1)<<" byte"
```

```
<<endl
```

```
<<"\t size of address pd1: "<<sizeof(pd1)<<" byte"
```

```
<<endl;
```

```
delete pd1;
```

```
pd1=NULL;
```

Выделение памяти вручную

Освобождение памяти  
вручную

## Работа с динамической переменной - 2

```
pd1=new double;  
*pd1=-9.876;  
cout<<"\t pd1 = "<<*pd1<<endl  
    <<"\t address pd1: "<<pd1<<endl  
    <<"\t size of data pd1: "<<sizeof(*pd1)<<" byte"  
    <<endl  
    <<"\t size of address pd1: "<<sizeof(pd1)<<" byte"  
    <<endl;  
delete pd1;  
pd1=NULL;
```

```
pd1 = -9.876  
address pd1: 0xc0ada8  
size of data pd1: 8 byte  
size of address pd1: 4 byte
```

## Работа с динамической переменной - 3

```
double *pd1=&d;
```

```
pd1=new double;
```

```
//....
```

```
delete pd1;
```

```
pd1=NULL;
```

## Работа с динамической переменной - 3

Инициализация адреса.  
Присвоению адресу адреса (**адресация**).

**double** `*pd1=&d;`

`pd1=new double;`

`// . . . .`

**delete** `pd1;`

`pd1=NULL;`

# Работа с динамической переменной - 3

Инициализация адреса.  
Присвоению адресу адреса (**адресация**).

```
double *pd1=&d;
```

```
pd1=new double;
```

```
// ...
```

```
delete pd1;
```

```
pd1=NULL;
```

Используется адрес уже  
хранящихся в памяти данных

Используется адрес ячейки,  
выделенной в heap

# Работа с динамической переменной - 3

Инициализация адреса.  
Присвоению адресу адреса (**адресация**).

```
double *pd1=&d;
```

```
pd1=new double;
```

```
// ...
```

```
delete pd1;
```

```
pd1=NULL;
```

Используется адрес уже  
хранящихся в памяти данных

Используется адрес ячейки,  
выделенной в heap

Деинициализация  
указателя



# Выделение и освобождение памяти в C++

- **Выделение памяти** – «изъятие» ячейки памяти из сегмента.
- При выделении памяти ячейка помечется как «занятая» и не может использоваться для хранения других данных вплоть до выполнения процедуры **«освобождения»**.
- По мере создания в программе новых объектов количество доступной памяти уменьшается, поэтому необходимо следить за своевременным освобождением ранее выделенной памяти.

**Программа всегда должна  
полностью освободить всю  
память, которая  
потребовалась для ее  
работы!**

# Способы выделения памяти в C++

- **Статическое выделение памяти** выполняется для статических и глобальных переменных. Память выделяется один раз (при запуске программы) и сохраняется на протяжении работы всей программы.
-

# Способы выделения памяти в C++

- **Статическое выделение памяти** выполняется для статических и глобальных переменных. Память выделяется один раз (при запуске программы) и сохраняется на протяжении работы всей программы.
- **Автоматическое выделение памяти** выполняется для параметров функции и локальных переменных. Память выделяется при входе в блок, в котором находятся эти переменные, и удаляется при выходе из него.
-

# Способы выделения памяти в C++

- **Статическое выделение памяти** выполняется для статических и глобальных переменных. Память выделяется один раз (при запуске программы) и сохраняется на протяжении работы всей программы.
- **Автоматическое выделение памяти** выполняется для параметров функции и локальных переменных. Память выделяется при входе в блок, в котором находятся эти переменные, и удаляется при выходе из него.
- **Динамическое выделение памяти** — способ выделения оперативной памяти компьютера для объектов в программе, при котором выделение памяти под объект осуществляется во время выполнения программы.

# Утечка памяти

**Утечка памяти** (англ. *memory leak*) — это неконтролируемое уменьшение свободной оперативной или виртуальной памяти компьютера.

Причиной утечек являются ошибки в программном коде, связанные с прямым управлением распределения памяти.

# Использование памяти при работе функций

# Параметры функции

- **Формальные параметры** - параметры, находящиеся в скобках, при объявлении прототипа функции и при ее определении.
- **Фактические параметры** - параметры, подставляемые на место формальных при вызове функции.



# Пример №1 - 1

```
void Func (double a)
{
    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<a<<endl
        <<"\t address a: "<<&a<<endl
        <<"\t size of a: "<<sizeof(a)<<" byte"
        <<endl;
}
```

# Пример №1 - 1

Формальный параметр.  
Передача данных в функцию по  
значению

```
void Func (double a)
{
    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<a<<endl
    <<"\t address a: "<<&a<<endl
    <<"\t size of a: "<<sizeof(a)<<" byte"
    <<endl;
}
```

Память под формальный параметр **a** выделяется в  
стеке в момент вызова функции.

Автоматически выполняется инициализация  
формального параметра данными из  
фактического параметра.

# Пример №1 - 2

```
double d {123.4567};
```

```
Func(d);
```

```
cout<<" --- In main --- "<<endl;
```

```
cout<<"\t d = "<<d<<endl
```

```
    <<"\t address d: "<<&d<<endl
```

```
    <<"\t size of d: "<<sizeof(d)<<" byte"
```

```
    <<endl;
```

## Пример №1 - 2

```
double d {123.4567};
```

```
Func(d);
```

```
cout<<"--- In main --- "<<endl;
```

```
cout<<"\t d = "<<d<<endl
```

```
<<"\t address d: "<<&d<<endl
```

```
<<"\t size of d: "<<sizeof(d)<<" byte"
```

```
<<endl;
```

Фактические параметры должны совпадать по количеству, порядку следования и по типу с фактическими параметрами.

Имена могут различаться.

Фактический параметр

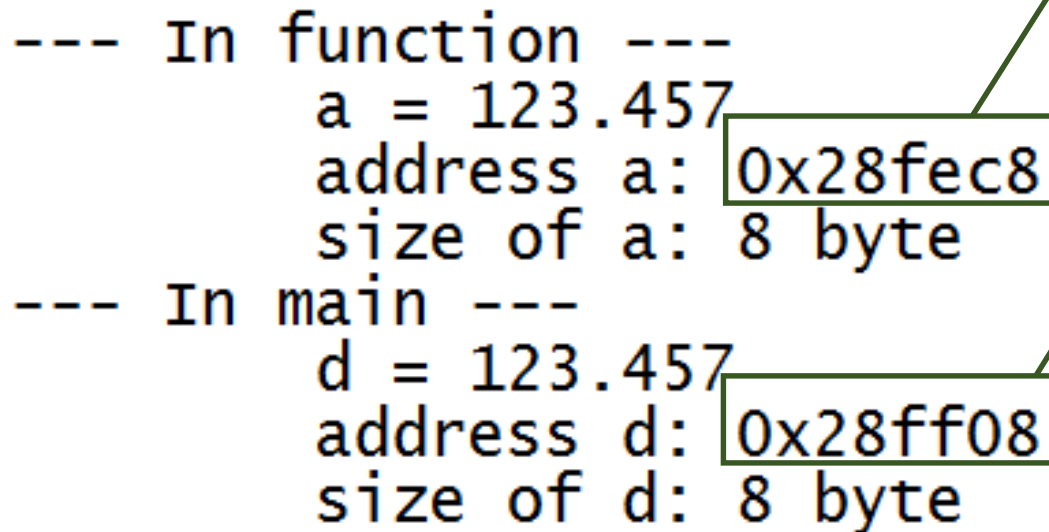
# Пример №1 - 3

```
--- In function ---  
    a = 123.457  
    address a: 0x28fec8  
    size of a: 8 byte  
--- In main ---  
    d = 123.457  
    address d: 0x28ff08  
    size of d: 8 byte
```

# Пример №1 - 3

Фактический и формальный параметры  
хранятся в разных ячейках в памяти

```
--- In function ---  
    a = 123.457  
    address a: 0x28fec8  
    size of a: 8 byte  
--- In main ---  
    d = 123.457  
    address d: 0x28ff08  
    size of d: 8 byte
```



# Пример №2 - 1

```
void Func1 (double a)
{
    a=10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<a<<endl
        <<"\t address a: "<<&a<<endl
        <<"\t size of a: "<<sizeof(a)<<" byte"
        <<endl;
}

double d {123.4567};

Func1(d);

cout<<" --- In main --- "<<endl;

cout<<"\t d = "<<d<<endl
    <<"\t address d: "<<&d<<endl
    <<"\t size of d: "<<sizeof(d)<<" byte"
    <<endl;
```

# Пример №2 - 1

```
void Func1 (double a)
{
```

```
    a=10;
```

```
    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<a<<endl
        <<"\t address a: "<<&a<<endl
        <<"\t size of a: "<<sizeof(a)<<" byte"
        <<endl;
```

```
}
```

```
double d {123.4567};
```

```
Func1(d);
```

```
cout<<" --- In main --- "<<endl;
```

```
cout<<"\t d = "<<d<<endl
    <<"\t address d: "<<&d<<endl
    <<"\t size of d: "<<sizeof(d)<<" byte"
    <<endl;
```

Изменится только формальный параметр.  
При завершении работы функции память из  
под формального параметра будет  
освобождена автоматически, а значение  
потеряно.



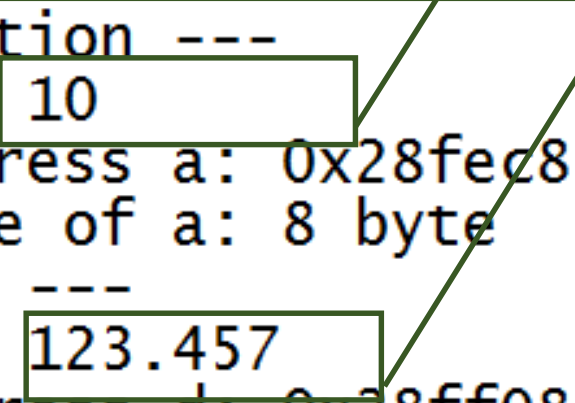
## Пример №2 - 2

```
--- In function ---  
    a = 10  
    address a: 0x28fec8  
    size of a: 8 byte  
--- In main ---  
    d = 123.457  
    address d: 0x28ff08  
    size of d: 8 byte
```

## Пример №2 - 2

Возврата значения не происходит

```
--- In function ---  
    a = 10  
    address a: 0x28fec8  
    size of a: 8 byte  
--- In main ---  
    d = 123.457  
    address d: 0x28ff08  
    size of d: 8 byte
```



# Пример №3 - 1

```
void Func2 (double& a)
{
    a=10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<a<<endl
        <<"\t address a: "<<&a<<endl
        <<"\t size of a: "<<sizeof(a)<<" byte"
        <<endl;

}

double d {123.4567};

Func2 (d);

cout<<" --- In main --- "<<endl;

cout<<"\t d = "<<d<<endl
    <<"\t address d: "<<&d<<endl
    <<"\t size of d: "<<sizeof(d)<<" byte"
    <<endl;
```

# Пример №3 - 1

Передача по ссылке

```
void Func2 (double& a)
{
    a=10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<a<<endl
    <<"\t address a: "<<&a<<endl
    <<"\t size of a: "<<sizeof(a)<<" byte"
    <<endl;
}

double d {123.4567};

Func2 (d);

cout<<" --- In main --- "<<endl;

cout<<"\t d = "<<d<<endl
<<"\t address d: "<<&d<<endl
<<"\t size of d: "<<sizeof(d)<<" byte"
<<endl;
```

Нет копирования значения. Происходит инициализация ссылки на данные из фактического параметра.  
Ссылка – «псевдоним» существующих данных.

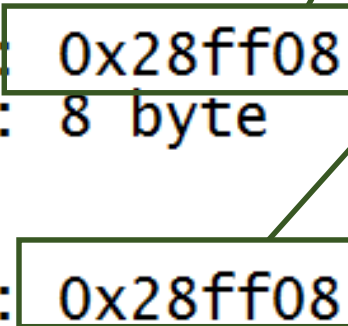
## Пример №3 - 2

```
--- In function ---  
    a = 10  
    address a: 0x28ff08  
    size of a: 8 byte  
--- In main ---  
    d = 10  
    address d: 0x28ff08  
    size of d: 8 byte
```

## Пример №3 - 2

Функция работает в ячейке, где хранится фактический параметр

```
--- In function ---  
    a = 10  
    address a: 0x28ff08  
    size of a: 8 byte  
--- In main ---  
    d = 10  
    address d: 0x28ff08  
    size of d: 8 byte
```



The diagram consists of two rectangular boxes, one above the other. The top box contains the text 'address a: 0x28ff08' and the bottom box contains 'address d: 0x28ff08'. Two arrows originate from these boxes: one from the top box pointing to the text 'Функция работает в ячейке, где хранится фактический параметр' in the box above, and another from the bottom box pointing to the same text. This illustrates that both variables, 'a' in the function and 'd' in the main, point to the same memory address (0x28ff08).

# Пример №4 - 1

```
void Func3 (double* a)
{
    *a=10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<*a<<endl
        <<"\t address a: "<<a<<endl
        <<"\t size of a: "<<sizeof(*a)<<" byte"
        <<endl;
}

double d {123.4567};

Func3 (&d);

cout<<" --- In main --- "<<endl;

cout<<"\t d = "<<d<<endl
    <<"\t address d: "<<&d<<endl
    <<"\t size of d: "<<sizeof(d)<<" byte"
    <<endl;
```

# Пример №4 - 1

```
void Func3 (double* a)
{
```

```
    *a=10;
```

Нужно разыменовывать

```
    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<*a<<endl
        <<"\t address a: "<<a<<endl
        <<"\t size of a: "<<sizeof(*a)<<" byte"
        <<endl;
```

```
    }
    double d {123.4567};
```

```
    Func3 (&d);
```

Передаем адрес

```
    cout<<" --- In main --- "<<endl;

    cout<<"\t d = "<<d<<endl
        <<"\t address d: "<<&d<<endl
        <<"\t size of d: "<<sizeof(d)<<" byte"
        <<endl;
```



# Пример №4 - 1

```
void Func3 (double* a)
```

```
{  
    *a=10;
```

```
    cout<<" --- In function --- "<<endl;
```

```
    cout<<"\t a = "<<*a<<endl
```

```
        <<"\t address a: "<<a<<endl
```

```
        <<"\t size of a: "<<sizeof(*a)<<" byte"
```

```
        <<endl;
```

```
}  
double d {123.4567};
```

```
Func3 (&d);
```

```
cout<<" --- In main --- "<<endl;
```

```
cout<<"\t d = "<<d<<endl
```

```
    <<"\t address d: "<<&d<<endl
```

```
    <<"\t size of d: "<<sizeof(d)<<" byte"
```

```
    <<endl;
```

Передача по адресу  
или по указателю

Нет копирования значения. Происходит адресация к ячейке памяти, где хранится фактический параметр.

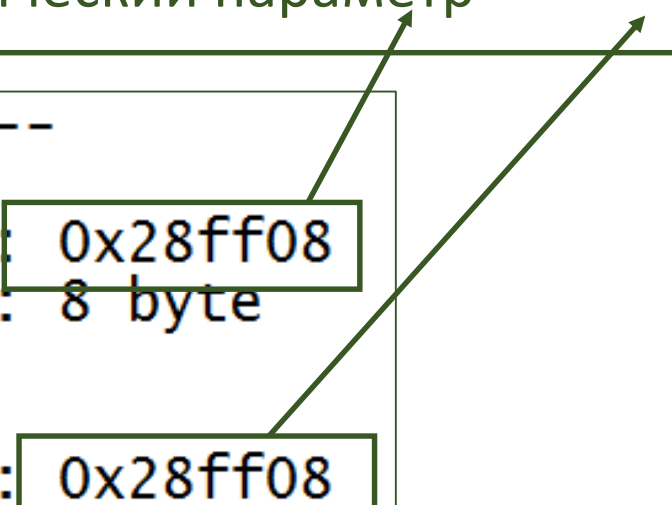
## Пример №4 - 2

```
--- In function ---  
    a = 10  
    address a: 0x28ff08  
    size of a: 8 byte  
--- In main ---  
    d = 10  
    address d: 0x28ff08  
    size of d: 8 byte
```

## Пример №4 - 2

Функция работает в ячейке, где хранится фактический параметр

```
--- In function ---  
    a = 10  
    address a: 0x28ff08  
    size of a: 8 byte  
--- In main ---  
    d = 10  
    address d: 0x28ff08  
    size of d: 8 byte
```



# Выделение памяти в функциях

# Пример №5

```
void Func4 ()
{
    double* a=new double;
    *a = 10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<*a<<endl
        <<"\t address a: "<<a<<endl
        <<"\t size of a: "<<sizeof(*a)<<" byte"
        <<endl;
    delete a;
}
```

# Пример №5

```
void Func4 ()
{
    double* a=new double;
    *a = 10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<*a<<endl
        <<"\t address a: "<<a<<endl
        <<"\t size of a: "<<sizeof(*a)<<" byte"
        <<endl;
    delete a;
}
```

С динамическими  
переменными можно  
работать локально  
в функциях.

# Пример №5

```
void Func4 ()
{
    double* a=new double;
    *a = 10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<*a<<endl
        <<"\t address a: "<<a<<endl
        <<"\t size of a: "<<sizeof(*a)<<" byte"
        <<endl;
    delete a;
}
```

Вызов:

```
Func4 ();
```

С динамическими  
переменными можно  
работать локально  
в функциях.

```
--- In function ---
    a = 10
    address a: 0x78ada8
    size of a: 8 byte
```

# Пример №6 - 1

```
void Func5 (double* a)
{
    a=new double;
    *a = 10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<*a<<endl
        <<"\t address a: "<<a<<endl
        <<"\t size of a: "<<sizeof(*a)<<" byte"
        <<endl;
}
```



# Пример №6 - 1

```
void Func5 (double* a)
{
    a=new double;
    *a = 10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<*a<<endl
        <<"\t address a: "<<a<<endl
        <<"\t size of a: "<<sizeof(*a)<<" byte"
        <<endl;
}
```

Адрес выделенной ячейки в памяти сохранится только в адрес формального параметра.

Неудачная попытка написать функцию, которая выделит память под «внешний» массив.

## Пример №6 - 2

```
double* d;  
Func5(d);  
cout<<" --- In main --- "<<endl;  
cout<<"\t d = "<<*d<<endl  
    <<"\t address d: "<<d<<endl  
    <<"\t size of d: "<<sizeof(*d)<<" byte"  
    <<endl;  
  
delete d;  
d=NULL;
```

## Пример №6 - 2

```
double* d;  
Func5(d);  
cout<<" --- In main --- "<<endl;  
cout<<"\t d = "<<*d<<endl  
    <<"\t address d: "<<d<<endl  
    <<"\t size of d: "<<sizeof(*d)<<" byte"  
    <<endl;  
  
delete d;  
d=NULL;
```

```
--- In function ---  
    a = 10  
    address a: 0x7aada8  
    size of a: 8 byte  
--- In main ---  
    d = -3.57615e+211  
    address d: 0x427a3e  
    size of d: 8 byte
```

Адреса формального и фактического параметров различаются.

# Пример №7 - 1

```
void Func6 (double*& a)
{
    a=new double;
    *a = 10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<*a<<endl
        <<"\t address a: "<<a<<endl
        <<"\t size of a: "<<sizeof(*a)<<" byte"
        <<endl;
}
```

# Пример №7 - 1

Память будет выделена под фактический параметр функции

```
void Func6 (double*& a)
{
    a=new double;
    *a = 10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<*a<<endl
        <<"\t address a: "<<a<<endl
        <<"\t size of a: "<<sizeof(*a)<<" byte"
        <<endl;
}
```

Передача по ссылке

# Пример №7 - 2

```
double* d;  
Func6(d);  
cout<<" --- In main --- "<<endl;  
cout<<"\t d = "<<*d<<endl  
    <<"\t address d: "<<d<<endl  
    <<"\t size of d: "<<sizeof(*d)<<" byte"  
    <<endl;  
  
delete d;  
d=NULL;
```

```
--- In function ---  
    a = 10  
    address a: 0x9cada8  
    size of a: 8 byte  
--- In main ---  
    d = 10  
    address d: 0x9cada8  
    size of d: 8 byte
```

Адреса совпадают

# Пример №8 - 1

```
void Func7 (double** a)
{
    *a=new double;
    *(*a) = 10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<**a<<endl
        <<"\t address a: "<<*a<<endl
        <<"\t size of a: "<<sizeof(**a)<<" byte"
        <<endl;
}
```

# Пример №8 - 1

Память будет выделена под фактический параметр функции

```
void Func7 (double** a)  
{
```

```
    *a=new double;  
    *(*a) = 10;
```

Передача по указателю

```
    cout<<" --- In function --- "<<endl;  
    cout<<"\t a = "<<**a<<endl  
        <<"\t address a: "<<*a<<endl  
        <<"\t size of a: "<<sizeof(**a)<<" byte"  
        <<endl;
```

Разыменовываем



# Пример №8 - 2

```
double* d;  
Func7 (&d) ;  
cout<<" --- In main --- "<<endl;  
cout<<"\t d = "<<*d<<endl  
    <<"\t address d: "<<d<<endl  
    <<"\t size of d: "<<sizeof(*d)<<" byte"  
    <<endl;  
  
delete d;  
d=NULL;
```

```
--- In function ---  
    a = 10  
    address a: 0x83ada8  
    size of a: 8 byte  
--- In main ---  
    d = 10  
    address d: 0x83ada8  
    size of d: 8 byte
```

Адреса совпадают

# Пример №9 - 1

```
double* Func8 ()
{
    double* a=new double;
    *a = 10;

    cout<<" --- In function --- "<<endl;
    cout<<"\t a = "<<*a<<endl
        <<"\t address a: "<<a<<endl
        <<"\t size of a: "<<sizeof(*a)<<" byte"
        <<endl;
    return a;
}
```

# Пример №9 - 1

```
double* Func8 ()  
{  
    double* a=new double;  
    *a = 10;  
  
    cout<<" --- In function --- "<<endl;  
    cout<<"\t a = "<<*a<<endl  
        <<"\t address a: "<<a<<endl  
        <<"\t size of a: "<<sizeof(*a)<<" byte"  
        <<endl;  
    return a;  
}
```

return a;

Возврат адреса  
выделенной памяти

## Пример №9 - 2

```
double* d;  
d=Func8();  
cout<<" --- In main --- "<<endl;  
cout<<"\t d = "<<*d<<endl  
    <<"\t address d: "<<d<<endl  
    <<"\t size of d: "<<sizeof(*d)<<" byte"  
    <<endl;  
  
delete d;  
d=NULL;
```

```
--- In function ---  
    a = 10  
    address a: 0x91ada8  
    size of a: 8 byte  
--- In main ---  
    d = 10  
    address d: 0x91ada8  
    size of d: 8 byte
```

Адреса совпадают