

## **Изменение атрибутов доступа к виртуальной странице**

Изменить атрибуты доступа к области виртуальной памяти можно при помощи вызова функции VirtualProtect, которая имеет следующий прототип:

```
BOOL VirtualProtect(  
    LPVOID lpAddress, // адрес области памяти  
    SIZE_T dwSize, // размер области памяти  
    DWORD  fNewProtect, // флаги новых атрибутов доступа  
    PDWORD lpfOldProtect // указатель на старые атрибуты доступа  
);
```

В случае успешного завершения эта функция возвращает значение, а в случае неудачи — FALSE. Параметры этой функции имеют тот же смысл, что и при распределении виртуальной памяти функцией VirtualAlloc. Единственное отличие состоит в том, что старые атрибуты доступа возвращаются функцией VirtualProtect по адресу lpfOldProtect, который должна установить вызывающая программа.

## **Инициализация и копирование блоков виртуальной памяти**

Чтобы заполнить блок памяти определенным значением, используется функция FillMemory, которая имеет следующий прототип:

```
VOID FillMemory(  
    PVOID Destination, // адрес блока памяти  
    SIZE_T Length, // длина блока  
    BYTE Fill // символ-заполнитель  
);
```

Эта функция заполняет блок памяти, длина в байтах и базовый адрес которого задаются соответственно параметрами Length и Destination, символом, заданным в параметре Fill.

Если блок памяти необходимо заполнить нулями, то для этого можно использовать функцию ZeroMemory, которая имеет следующий прототип:

```
VOID ZeroMemory(  
    PVOID Destination, // адрес блока памяти  
    SIZE_T Length, // длина блока  
);
```

Параметры этой функции имеют то же назначение, что и параметры функции FillMemory, исключая символ-заполнитель.

Для копирования блока виртуальной памяти используется функция CopyMemory, которая имеет следующий прототип:

```
VOID CopyMemory(  
    PVOID Destination, // адрес области назначения  
    CONST VOID *Source, // адрес исходной области  
    SIZE_T Length // длина блока памяти  
);
```

Эта функция копирует блок памяти, длина в байтах и базовый адрес которого задаются соответственно параметрами Length и Source в область памяти по адресу Destination. Результат выполнения функции CopyMemory не-предсказуем, если исходный и результирующий блоки памяти перекрываются.

Для копирования перекрывающихся блоков памяти используется функция MoveMemory, которая имеет следующий прототип:

```
VOID MoveMemory(
```

```
    PVOID Destination, // адрес области назначения
```

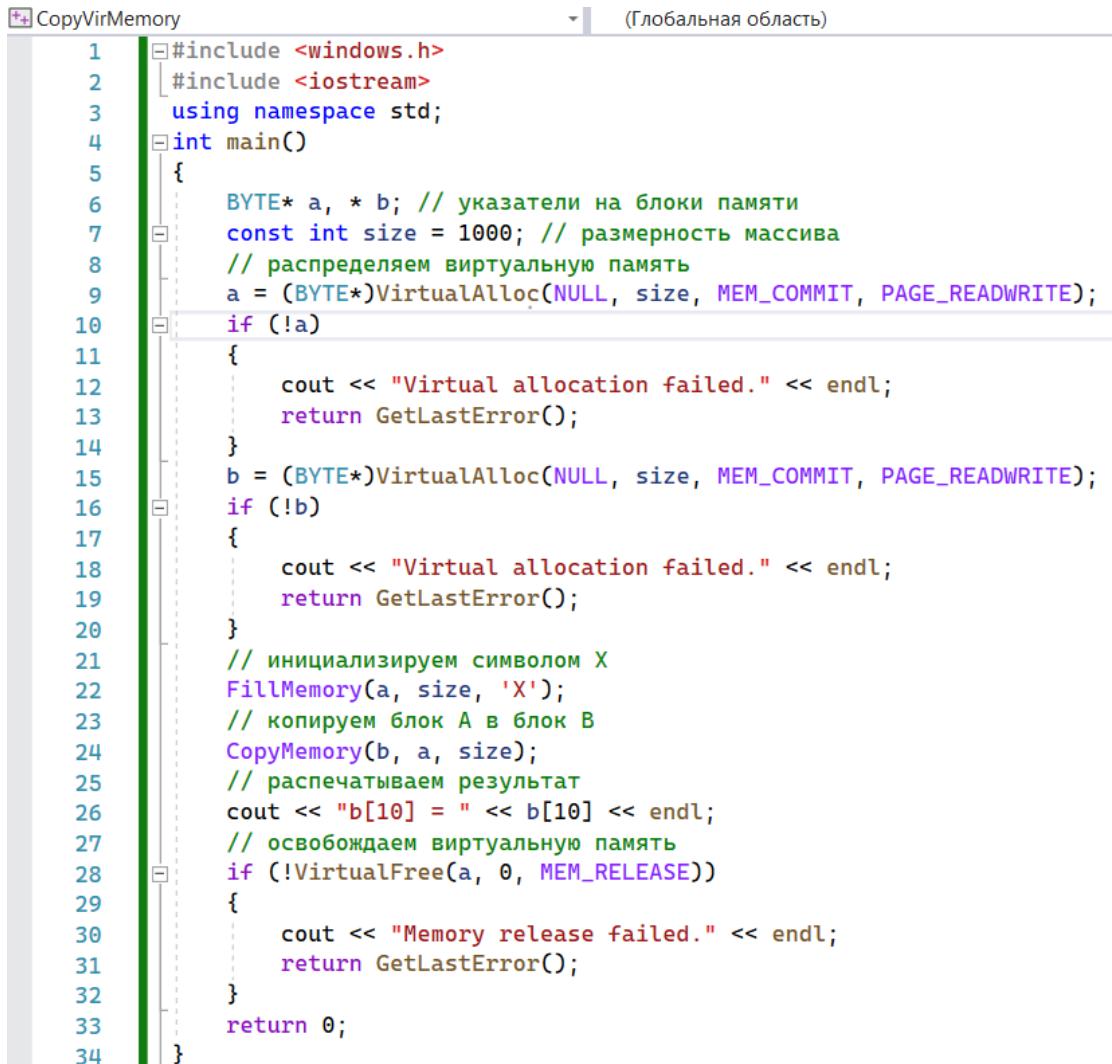
```
    CONST VOID *Source, // адрес исходной области
```

```
    SIZE_T Length // длина блока памяти
```

```
);
```

Параметры этой функции полностью совпадают с параметрами функции CopyMemory.

## Инициализация и копирование блоков виртуальной памяти



The screenshot shows a code editor window with the title "CopyVirMemory". The code is written in C++ and performs the following steps:

- #include <windows.h>
- #include <iostream>
- using namespace std;
- int main()
- {
- BYTE\* a, \* b; // указатели на блоки памяти
- const int size = 1000; // размерность массива
- // распределяем виртуальную память
- a = (BYTE\*)VirtualAlloc(NULL, size, MEM\_COMMIT, PAGE\_READWRITE);
- if (!a)
- {
- cout << "Virtual allocation failed." << endl;
- return GetLastError();
- }
- b = (BYTE\*)VirtualAlloc(NULL, size, MEM\_COMMIT, PAGE\_READWRITE);
- if (!b)
- {
- cout << "Virtual allocation failed." << endl;
- return GetLastError();
- }
- // инициализируем символом X
- FillMemory(a, size, 'X');
- // копируем блок A в блок B
- CopyMemory(b, a, size);
- // распечатываем результат
- cout << "b[10] = " << b[10] << endl;
- // освобождаем виртуальную память
- if (!VirtualFree(a, 0, MEM\_RELEASE))
- {
- cout << "Memory release failed." << endl;
- return GetLastError();
- }
- return 0;