C/C++ Program Design

LAB 4

CONTENTS

- Learn to create and use pointers
- Learn to manage dynamic memory

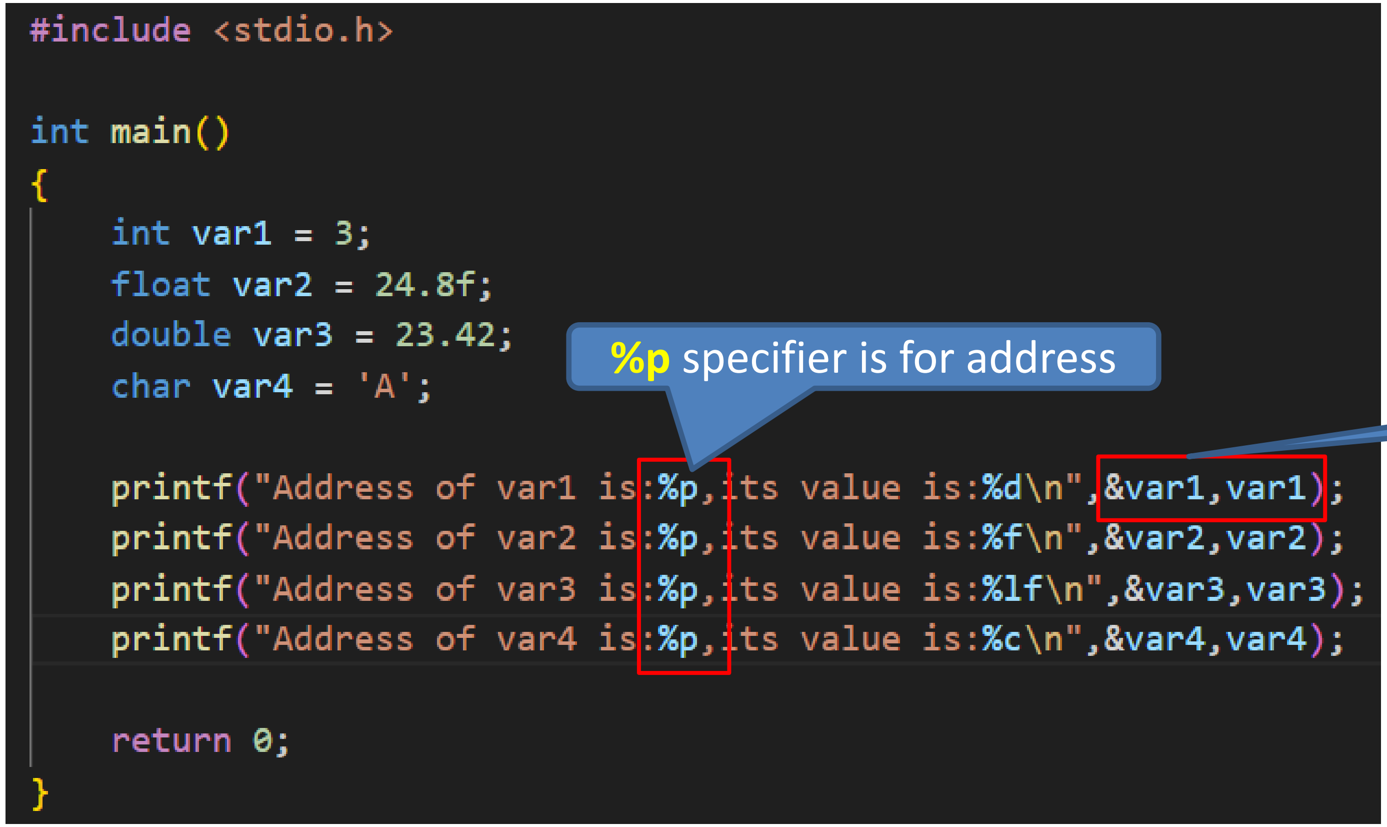
2 Knowledge Points

2.1 Pointers

2.2 Dynamic memory management

2.1 Pointers

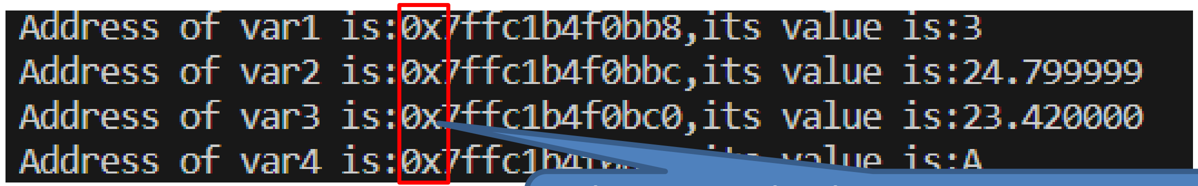
Pointer is a special type who holds the address of value.



```c
#include <stdio.h>

int main()
{
    int var1 = 3;
    float var2 = 24.8f;
    double var3 = 23.42;
    char var4 = 'A';

    printf("Address of var1 is:%p,its value is:%d\n",&var1,var1);
    printf("Address of var2 is:%p,its value is:%f\n",&var2,var2);
    printf("Address of var3 is:%p,its value is:%lf\n",&var3,var3);
    printf("Address of var4 is:%p,its value is:%c\n",&var4,var4);

    return 0;
}
```

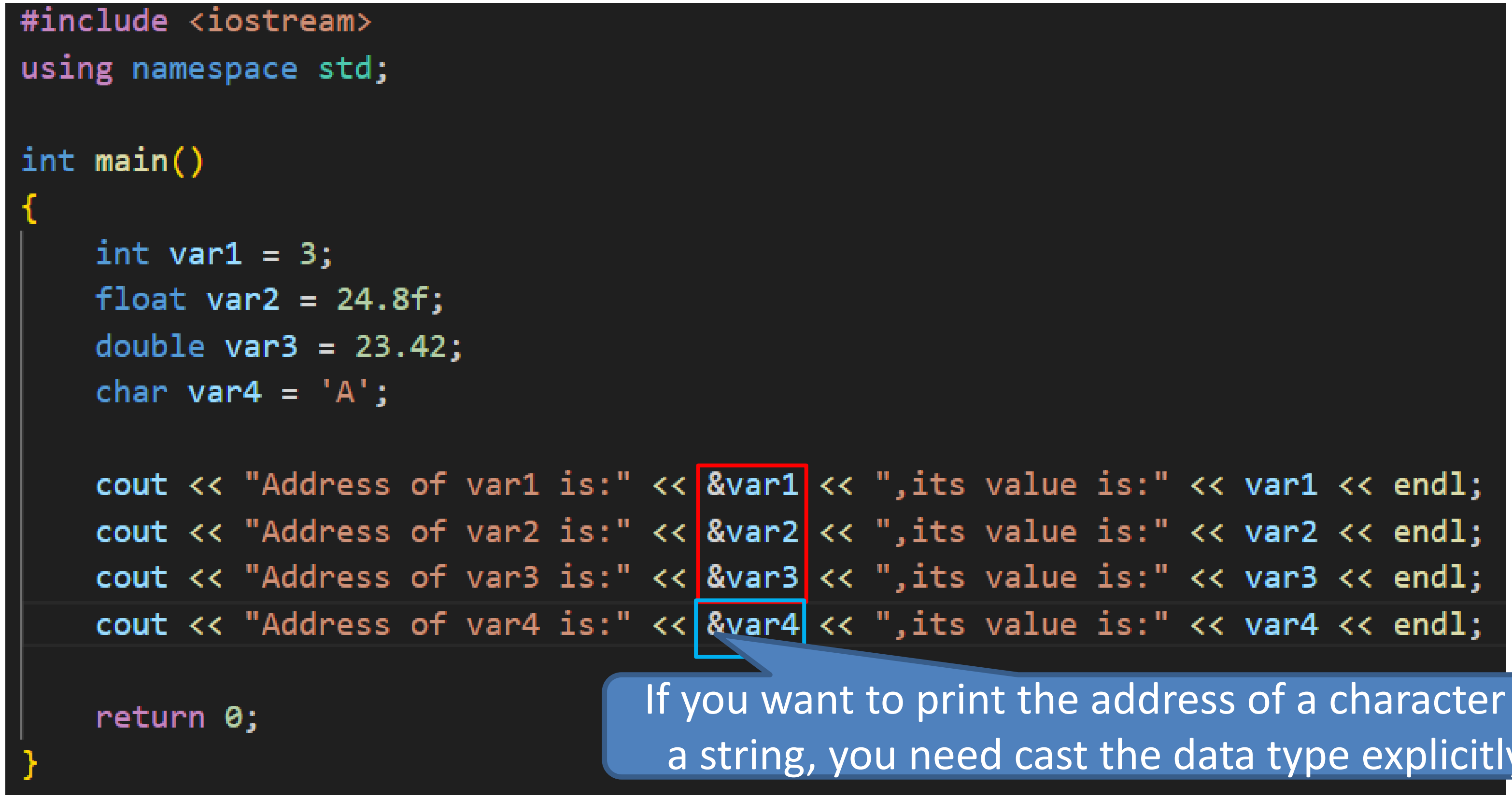
%p specifier is for address

var1 is a variable, **&var1** gives its address.



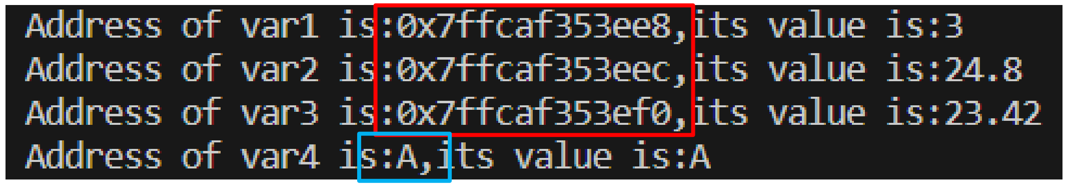
```
Address of var1 is:0x7ffc1b4f0bb8,its value is:3
Address of var2 is:0x7ffc1b4f0bbc,its value is:24.799999
Address of var3 is:0x7ffc1b4f0bc0,its value is:23.420000
Address of var4 is:0x7ffc1b4[illegible] value is:A
```

The 0x in the beginning represents the address in hexadecimal form.

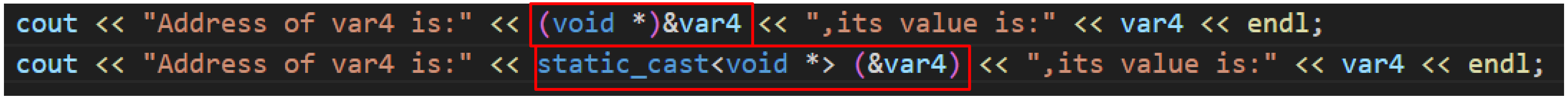


```cpp
#include <iostream>
using namespace std;

int main()
{
    int var1 = 3;
    float var2 = 24.8f;
    double var3 = 23.42;
    char var4 = 'A';

    cout << "Address of var1 is:" << &var1 << ",its value is:" << var1 << endl;
    cout << "Address of var2 is:" << &var2 << ",its value is:" << var2 << endl;
    cout << "Address of var3 is:" << &var3 << ",its value is:" << var3 << endl;
    cout << "Address of var4 is:" << &var4 << ",its value is:" << var4 << endl;

    return 0;
}
```

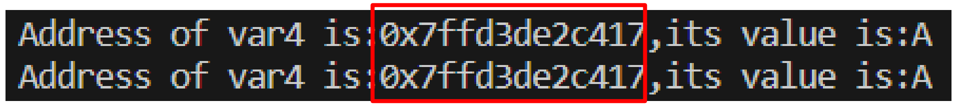
If you want to print the address of a character or a string, you need cast the data type explicitly.



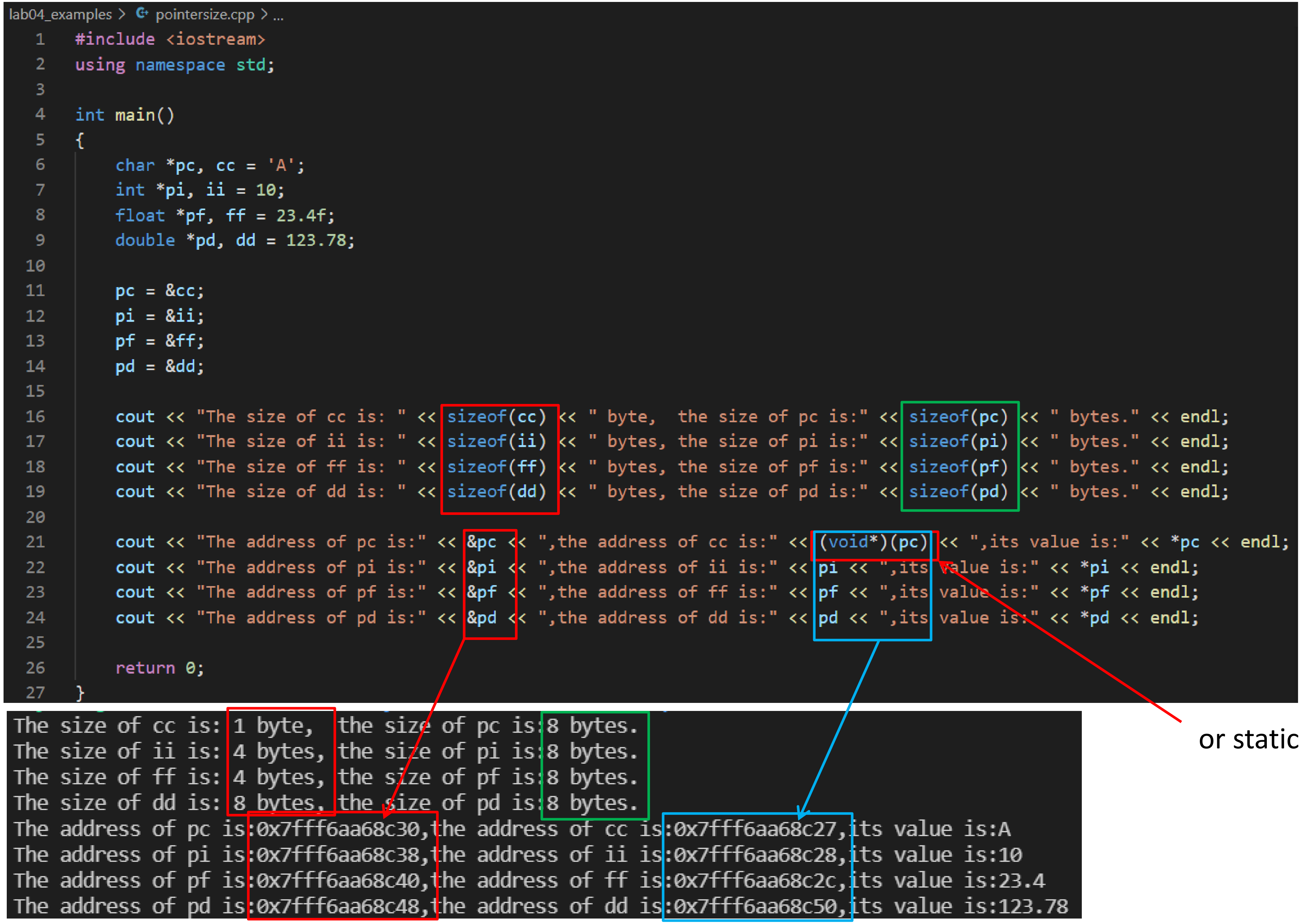
```
Address of var1 is:0x7ffcaf353ee8,its value is:3
Address of var2 is:0x7ffcaf353eec,its value is:24.8
Address of var3 is:0x7ffcaf353ef0,its value is:23.42
Address of var4 is:A,its value is:A
```



```cpp
cout << "Address of var4 is:" << (void *)&var4 << ",its value is:" << var4 << endl;
cout << "Address of var4 is:" << static_cast<void *> (&var4) << ",its value is:" << var4 << endl;
```



```
Address of var4 is:0x7ffd3de2c417,its value is:A
Address of var4 is:0x7ffd3de2c417,its value is:A
```

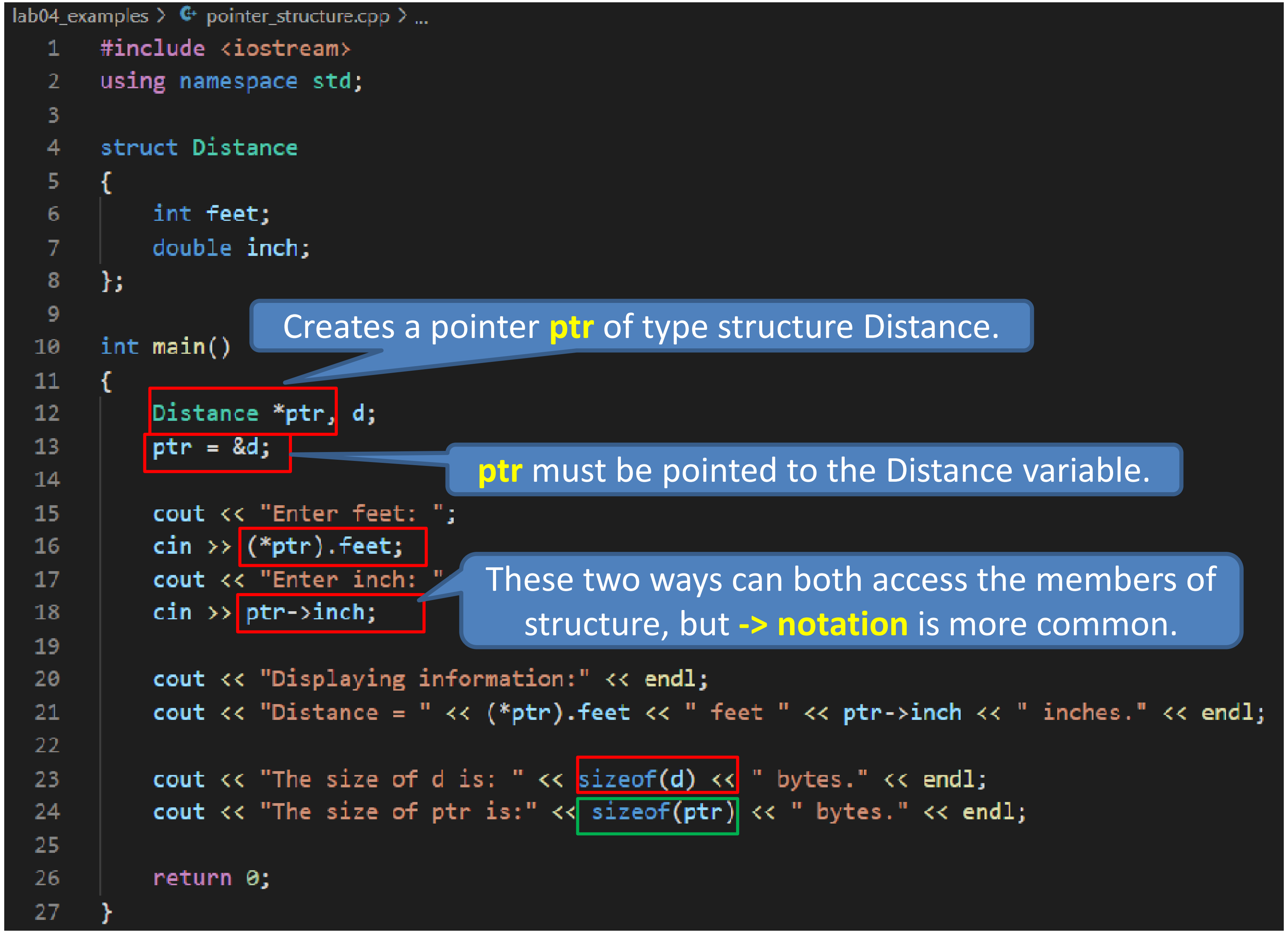


```cpp
#include <iostream>
using namespace std;

int main()
{
    char *pc, cc = 'A';
    int *pi, ii = 10;
    float *pf, ff = 23.4f;
    double *pd, dd = 123.78;

    pc = &cc;
    pi = &ii;
    pf = &ff;
    pd = &dd;

    cout << "The size of cc is: " << sizeof(cc) << " byte,  the size of pc is:" << sizeof(pc) << " bytes." << endl;
    cout << "The size of ii is: " << sizeof(ii) << " bytes, the size of pi is:" << sizeof(pi) << " bytes." << endl;
    cout << "The size of ff is: " << sizeof(ff) << " bytes, the size of pf is:" << sizeof(pf) << " bytes." << endl;
    cout << "The size of dd is: " << sizeof(dd) << " bytes, the size of pd is:" << sizeof(pd) << " bytes." << endl;

    cout << "The address of pc is:" << &pc << ",the address of cc is:" << (void*)(pc) << ",its value is:" << *pc << endl;
    cout << "The address of pi is:" << &pi << ",the address of ii is:" << pi << ",its value is:" << *pi << endl;
    cout << "The address of pf is:" << &pf << ",the address of ff is:" << pf << ",its value is:" << *pf << endl;
    cout << "The address of pd is:" << &pd << ",the address of dd is:" << pd << ",its value is:" << *pd << endl;

    return 0;
}
```

```
The size of cc is: 1 byte,  the size of pc is:8 bytes.
The size of ii is: 4 bytes, the size of pi is:8 bytes.
The size of ff is: 4 bytes, the size of pf is:8 bytes.
The size of dd is: 8 bytes, the size of pd is:8 bytes.
The address of pc is:0x7fff6aa68c30,the address of cc is:0x7fff6aa68c27,its value is:A
The address of pi is:0x7fff6aa68c38,the address of ii is:0x7fff6aa68c28,its value is:10
The address of pf is:0x7fff6aa68c40,the address of ff is:0x7fff6aa68c2c,its value is:23.4
The address of pd is:0x7fff6aa68c48,the address of dd is:0x7fff6aa68c50,its value is:123.78
```

or static_cast<void *> (pc)

Pointer and structure

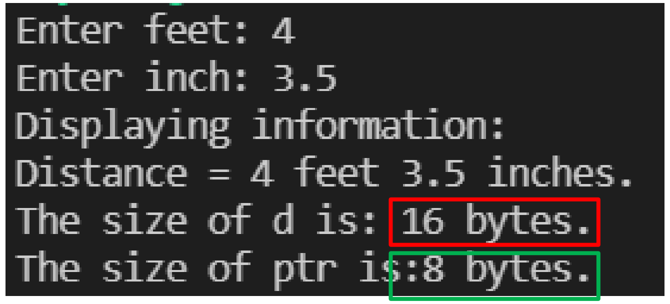


```cpp
#include <iostream>
using namespace std;

struct Distance
{
    int feet;
    double inch;
};

int main()
{
    Distance *ptr, d;
    ptr = &d;

    cout << "Enter feet: ";
    cin >> (*ptr).feet;
    cout << "Enter inch: ";
    cin >> ptr->inch;

    cout << "Displaying information:" << endl;
    cout << "Distance = " << (*ptr).feet << " feet " << ptr->inch << " inches." << endl;

    cout << "The size of d is: " << sizeof(d) << " bytes." << endl;
    cout << "The size of ptr is:" << sizeof(ptr) << " bytes." << endl;

    return 0;
}
```

Creates a pointer **ptr** of type structure Distance.

ptr must be pointed to the Distance variable.

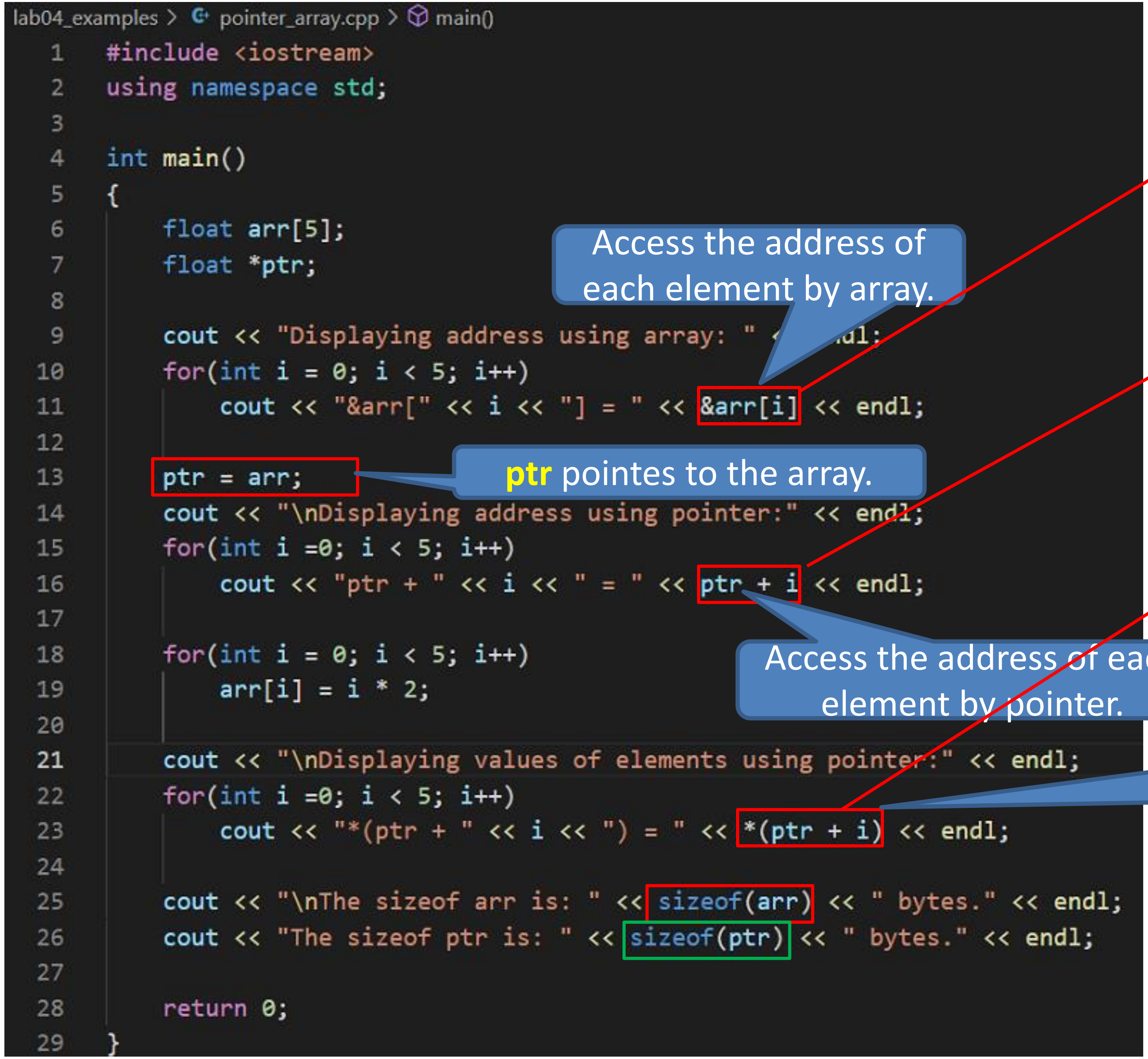
These two ways can both access the members of structure, but **-> notation** is more common.



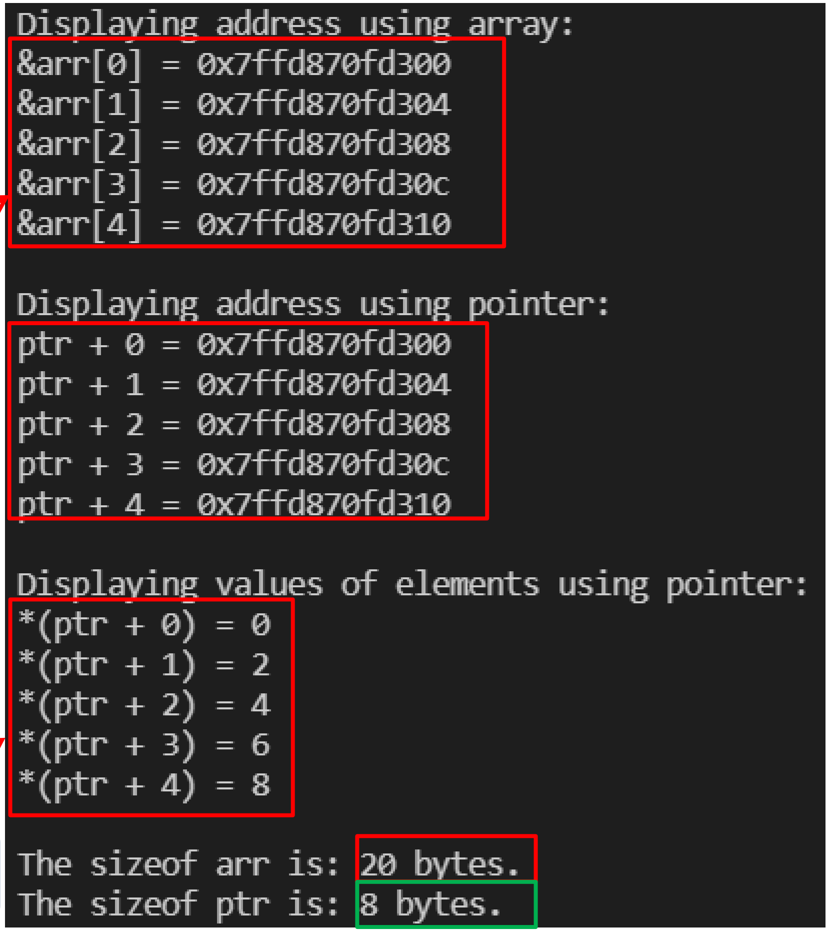
```
Enter feet: 4
Enter inch: 3.5
Displaying information:
Distance = 4 feet 3.5 inches.
The size of d is: 16 bytes.
The size of ptr is:8 bytes.
```

Note: Since pointer ptr is pointed to variable d in this program, (*ptr).inch ,ptr->inch and d.inch are exact the same.

Pointer and array



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

int main()
{
    float arr[5];
    float *ptr;

    cout << "Displaying address using array: " << endl;
    for(int i = 0; i < 5; i++)
        cout << "&arr[" << i << "] = " << &arr[i] << endl;

    ptr = arr;
    cout << "\nDisplaying address using pointer:" << endl;
    for(int i =0; i < 5; i++)
        cout << "ptr + " << i << " = " << ptr + i << endl;

    for(int i = 0; i < 5; i++)
        arr[i] = i * 2;

    cout << "\nDisplaying values of elements using pointer:" << endl;
    for(int i =0; i < 5; i++)
        cout << "*(ptr + " << i << ") = " << *(ptr + i) << endl;

    cout << "\nThe sizeof arr is: " << sizeof(arr) << " bytes." << endl;
    cout << "The sizeof ptr is: " << sizeof(ptr) << " bytes." << endl;

    return 0;
}
```



```
Displaying address using array:
&arr[0] = 0x7ffd870fd300
&arr[1] = 0x7ffd870fd304
&arr[2] = 0x7ffd870fd308
&arr[3] = 0x7ffd870fd30c
&arr[4] = 0x7ffd870fd310

Displaying address using pointer:
ptr + 0 = 0x7ffd870fd300
ptr + 1 = 0x7ffd870fd304
ptr + 2 = 0x7ffd870fd308
ptr + 3 = 0x7ffd870fd30c
ptr + 4 = 0x7ffd870fd310

Displaying values of elements using pointer:
*(ptr + 0) = 0
*(ptr + 1) = 2
*(ptr + 2) = 4
*(ptr + 3) = 6
*(ptr + 4) = 8

The sizeof arr is: 20 bytes.
The sizeof ptr is: 8 bytes.
```

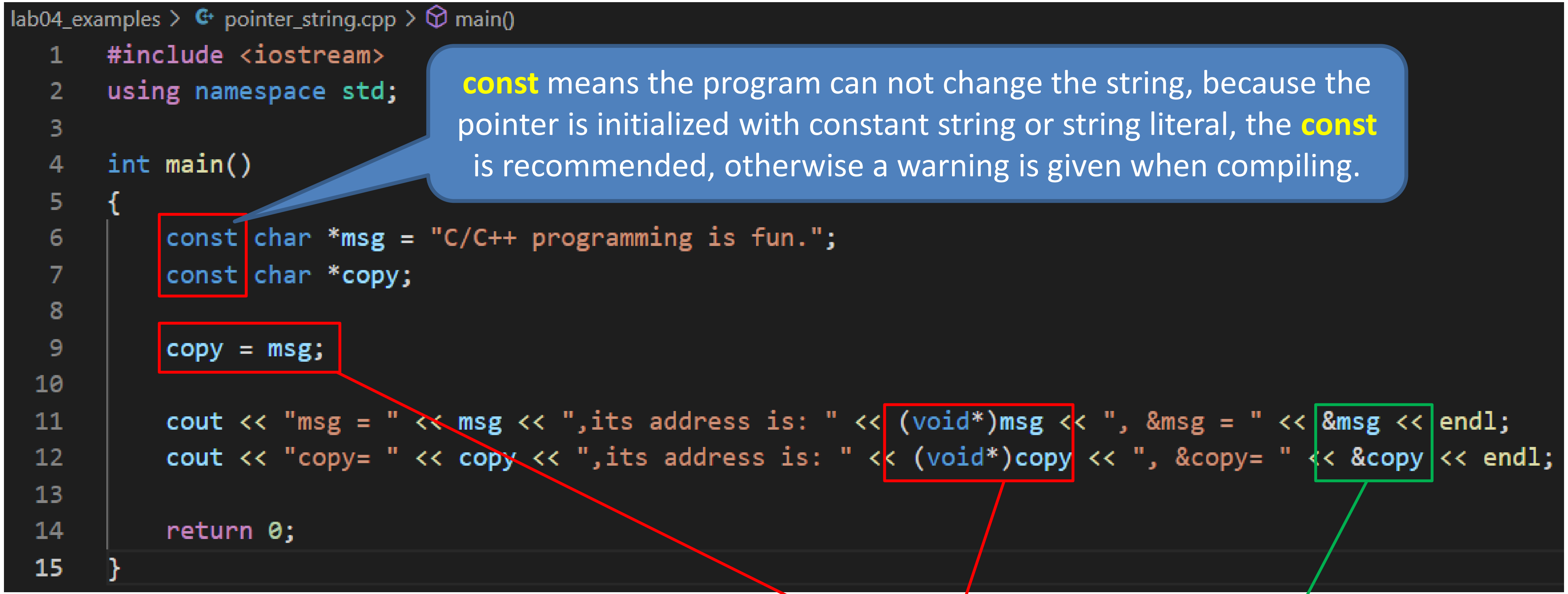
Access the address of each element by array.

ptr pointes to the array.

Access the address of each element by pointer.

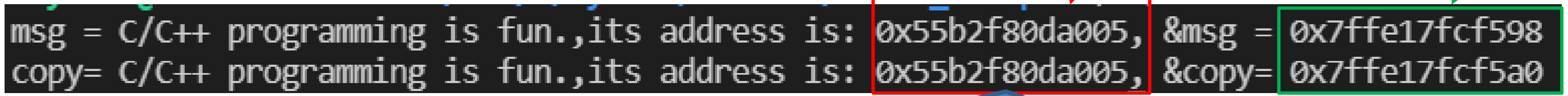
Access the values of elements by pointer using * operator.

Pointer and string



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

int main()
{
    const char *msg = "C/C++ programming is fun.";
    const char *copy;

    copy = msg;

    cout << "msg = " << msg << ",its address is: " << (void*)msg << ", &msg = " << &msg << endl;
    cout << "copy= " << copy << ",its address is: " << (void*)copy << ", &copy= " << &copy << endl;

    return 0;
}
```

const means the program can not change the string, because the pointer is initialized with constant string or string literal, the const is recommended, otherwise a warning is given when compiling.

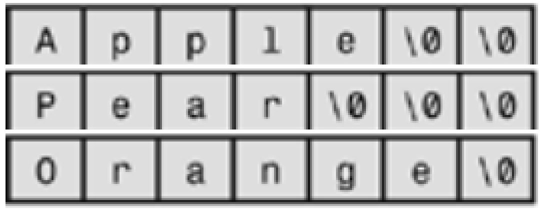


```
msg = C/C++ programming is fun.,its address is: 0x55b2f80da005, &msg = 0x7ffe17fcf598
copy= C/C++ programming is fun.,its address is: 0x55b2f80da005, &copy= 0x7ffe17fcf5a0
```

These two values are equal, indicates both of the pointers are pointed to the same string, although their own address are different.

Pointer array: each element in the array is a pointer.

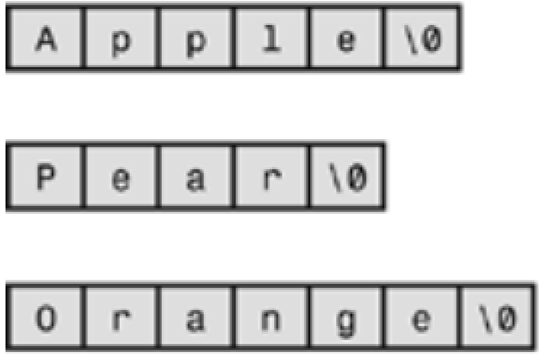
char fruit1[3][7] = { “Apple”, “Pear”, “Orange”};



A	p	p	l	e	\0	\0
P	e	a	r	\0	\0	\0
O	r	a	n	g	e	\0

fruit1 is an array of three elements, and each of these elements is itself an array of 7 char values with all the rows of the same length. In short, fruit1 is an array of arrays of char and is stored consecutively in memory.

const char *fruit2[3] = { “Apple”, “Pear”, “Orange”};

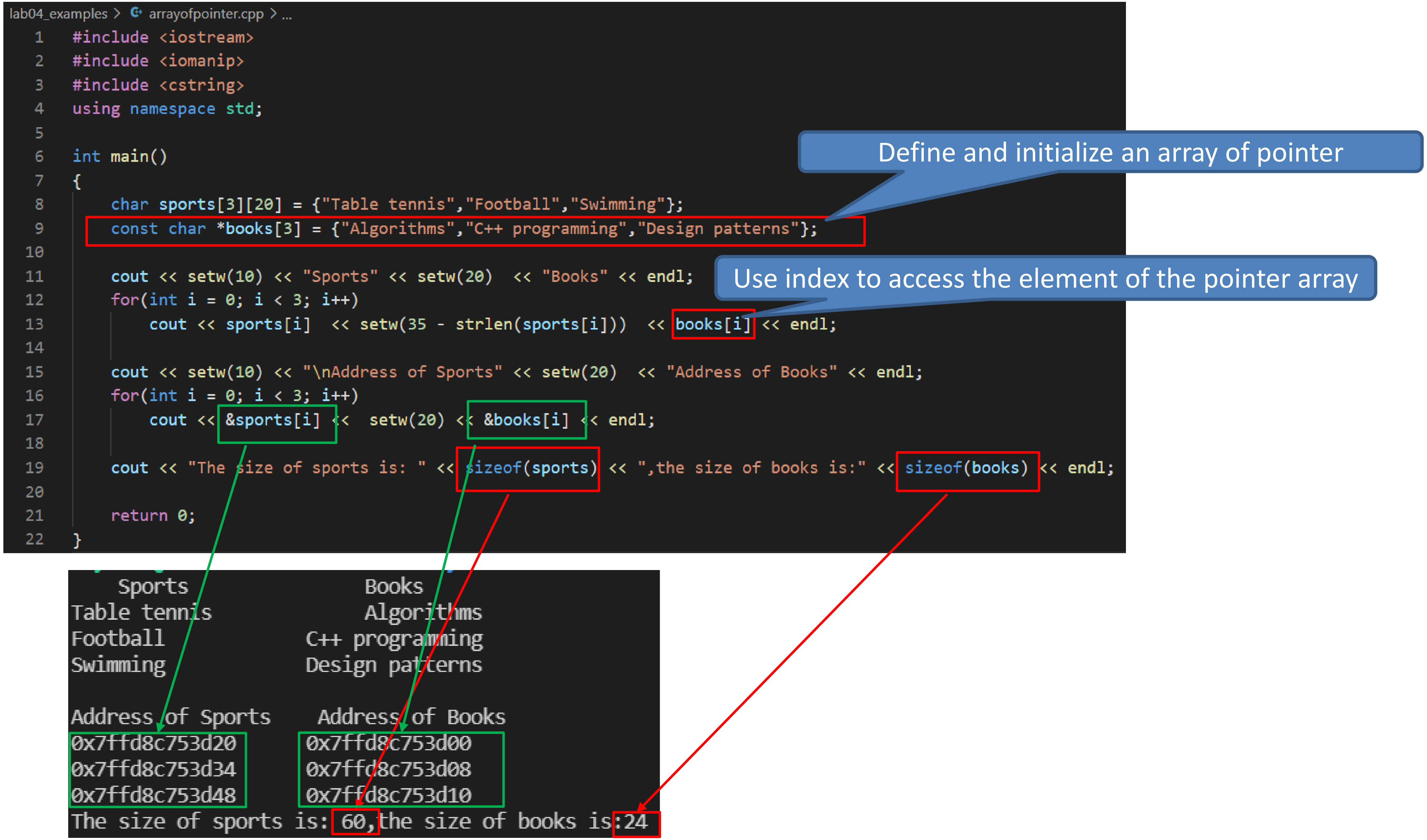


A	p	p	l	e	\0

P	e	a	r	\0

O	r	a	n	g	e	\0

fruit2 is an array of three **pointers-to-char**, each element doesn’t necessarily have to be stored consecutively in memory. It sets up a ragged array.



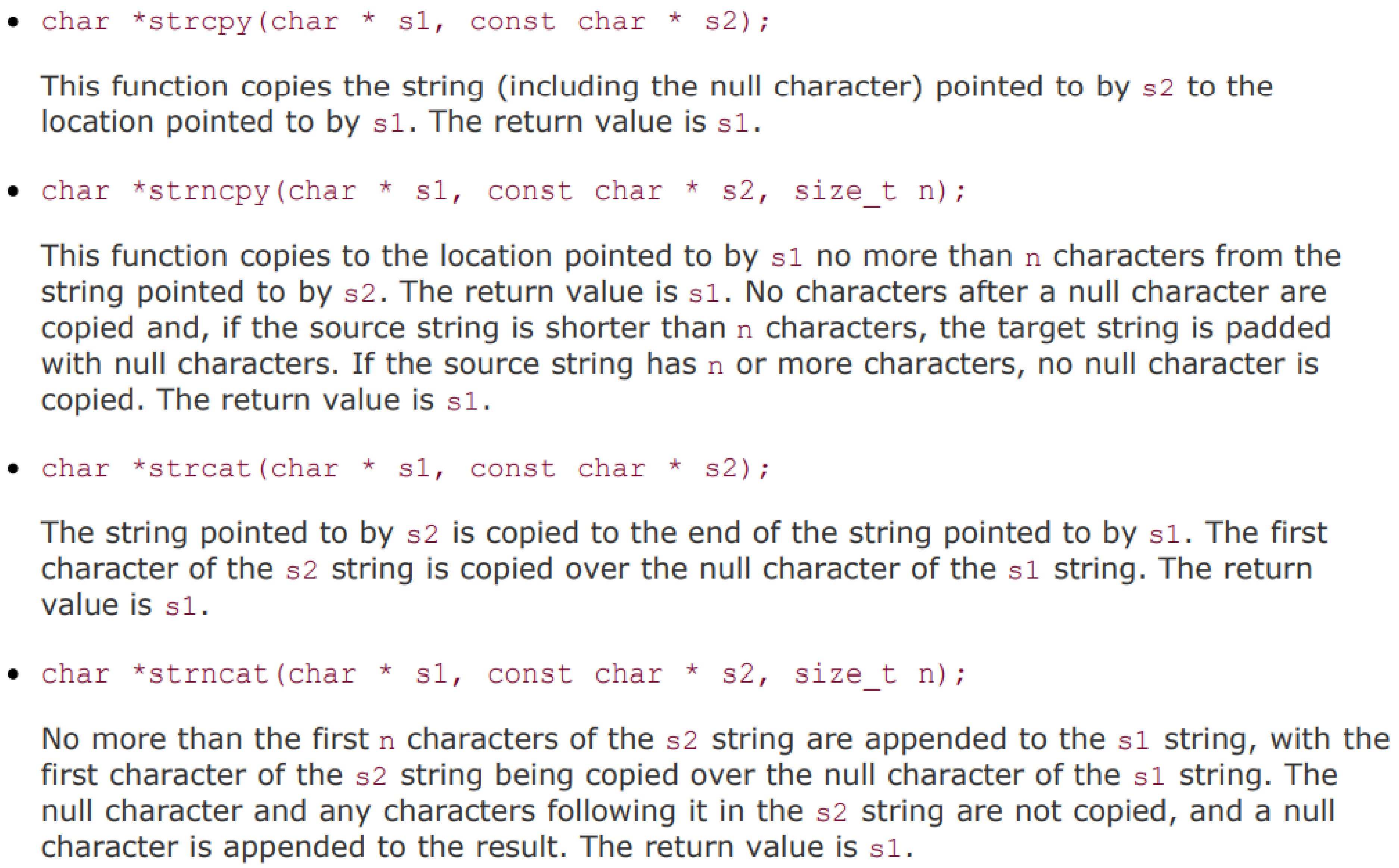
```cpp
#include <iostream>
#include <iomanip>
#include <cstring>
using namespace std;

int main()
{
    char sports[3][20] = {"Table tennis","Football","Swimming"};
    const char *books[3] = {"Algorithms","C++ programming","Design patterns"};

    cout << setw(10) << "Sports" << setw(20)  << "Books" << endl;
    for(int i = 0; i < 3; i++)
        cout << sports[i]  << setw(35 - strlen(sports[i]))  << books[i] << endl;

    cout << setw(10) << "\nAddress of Sports" << setw(20)  << "Address of Books" << endl;
    for(int i = 0; i < 3; i++)
        cout << &sports[i] <<  setw(20) << &books[i] << endl;

    cout << "The size of sports is: " << sizeof(sports) << ",the size of books is:" << sizeof(books) << endl;

    return 0;
}
```

Define and initialize an array of pointer

Use index to access the element of the pointer array

```
    Sports             Books
Table tennis          Algorithms
Football         C++ programming
Swimming         Design patterns

Address of Sports    Address of Books
0x7ffd8c753d20   0x7ffd8c753d00
0x7ffd8c753d34   0x7ffd8c753d08
0x7ffd8c753d48   0x7ffd8c753d10
The size of sports is: 60,the size of books is:24
```

String functions
#include <cstring>



- `char *strcpy(char * s1, const char * s2);`

  This function copies the string (including the null character) pointed to by s2 to the location pointed to by s1. The return value is s1.

- `char *strncpy(char * s1, const char * s2, size_t n);`

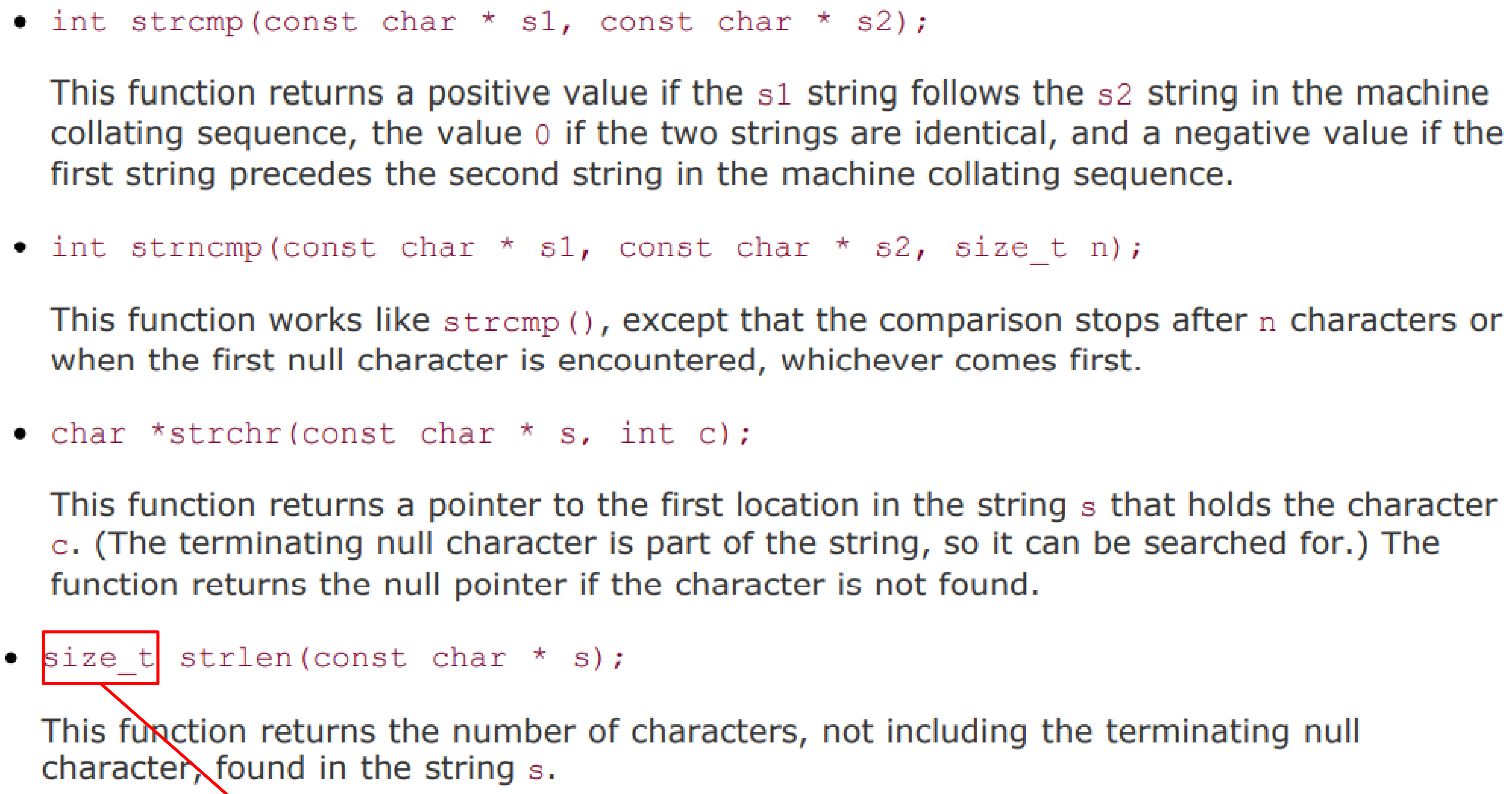
  This function copies to the location pointed to by s1 no more than n characters from the string pointed to by s2. The return value is s1. No characters after a null character are copied and, if the source string is shorter than n characters, the target string is padded with null characters. If the source string has n or more characters, no null character is copied. The return value is s1.

- `char *strcat(char * s1, const char * s2);`

  The string pointed to by s2 is copied to the end of the string pointed to by s1. The first character of the s2 string is copied over the null character of the s1 string. The return value is s1.

- `char *strncat(char * s1, const char * s2, size_t n);`

  No more than the first n characters of the s2 string are appended to the s1 string, with the first character of the s2 string being copied over the null character of the s1 string. The null character and any characters following it in the s2 string are not copied, and a null character is appended to the result. The return value is s1.



- `int strcmp(const char * s1, const char * s2);`

  This function returns a positive value if the `s1` string follows the `s2` string in the machine collating sequence, the value `0` if the two strings are identical, and a negative value if the first string precedes the second string in the machine collating sequence.

- `int strncmp(const char * s1, const char * s2, size_t n);`

  This function works like `strcmp()`, except that the comparison stops after `n` characters or when the first null character is encountered, whichever comes first.

- `char *strchr(const char * s, int c);`

  This function returns a pointer to the first location in the string `s` that holds the character `c`. (The terminating null character is part of the string, so it can be searched for.) The function returns the null pointer if the character is not found.

- `size_t strlen(const char * s);`

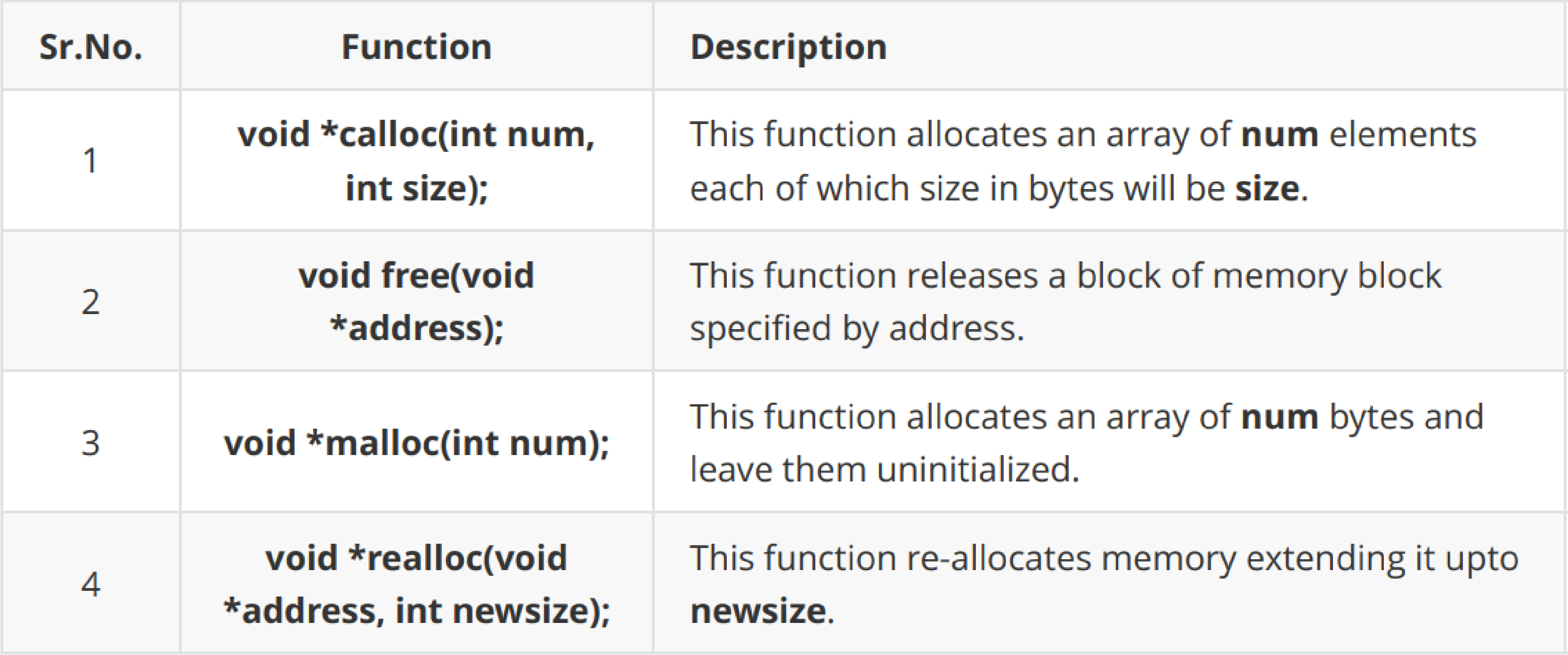
  This function returns the number of characters, not including the terminating null character, found in the string `s`.

```
typedef  unsigned int  size_t;
```

2.2 Dynamic Memory

2.2.1 C Dynamic Memory

These functions can be found in the **<stdlib.h>** header file.

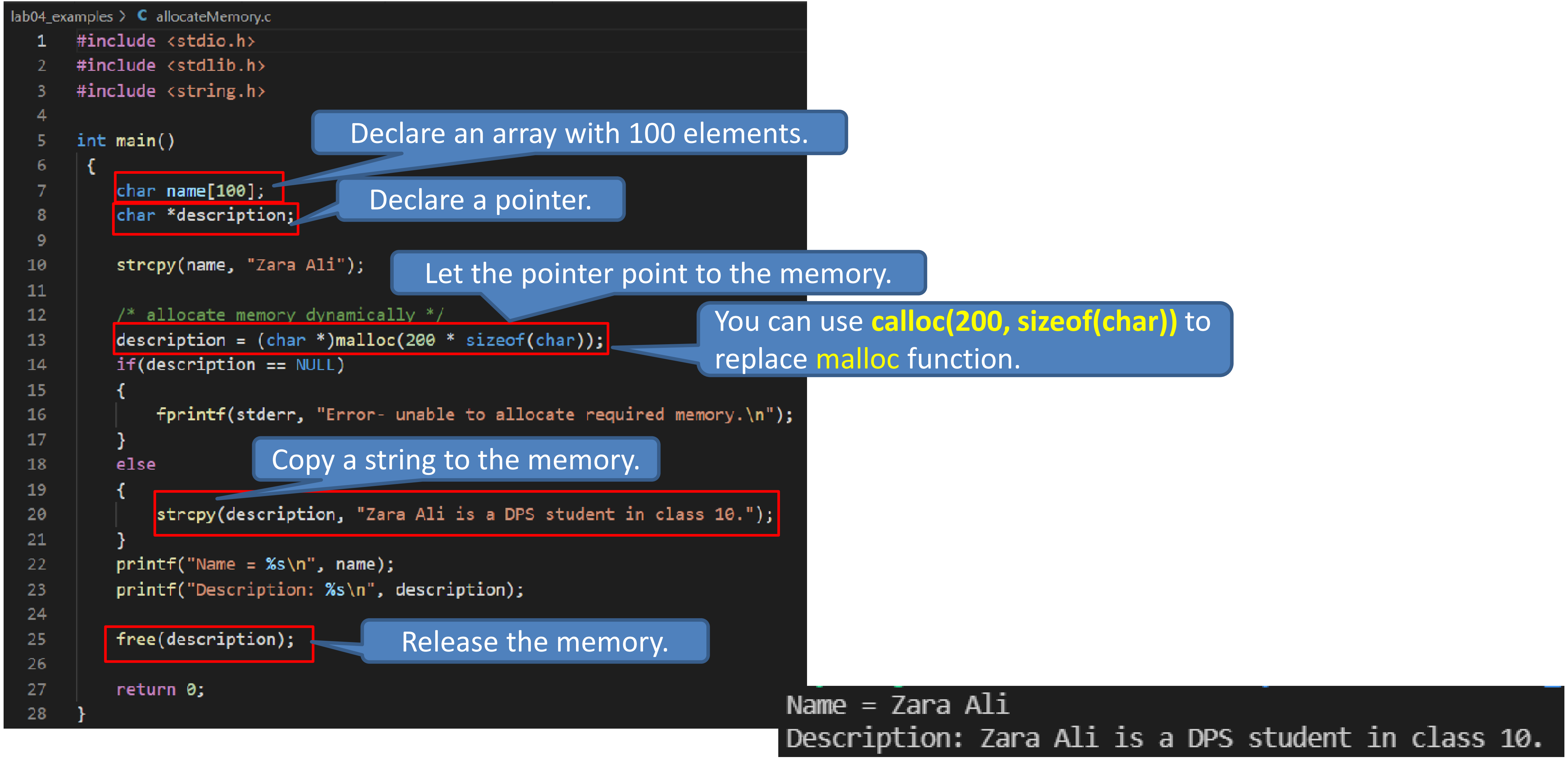


Sr.No.	Function	Description
1	**void *calloc(int num, int size);**	This function allocates an array of **num** elements each of which size in bytes will be **size**.
2	**void free(void *address);**	This function releases a block of memory block specified by address.
3	**void *malloc(int num);**	This function allocates an array of **num** bytes and leave them uninitialized.
4	**void *realloc(void *address, int newsize);**	This function re-allocates memory extending it upto **newsize**.

When you are not in need of memory any more, you should release that memory by calling the function **free().**

1. Allocating Memory Dynamically

When you declare an array, you must specify the number of the elements. Sometimes you don't know the amount of the elements, you can declare a pointer, and let it point to the memory which allocated dynamically.



lab04_examples > C allocateMemory.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char name[100];
    char *description;

    strcpy(name, "Zara Ali");

    /* allocate memory dynamically */
    description = (char *)malloc(200 * sizeof(char));
    if(description == NULL)
    {
        fprintf(stderr, "Error- unable to allocate required memory.\n");
    }
    else
    {
        strcpy(description, "Zara Ali is a DPS student in class 10.");
    }
    printf("Name = %s\n", name);
    printf("Description: %s\n", description);

    free(description);

    return 0;
}
```

Declare an array with 100 elements.

Declare a pointer.

Let the pointer point to the memory.

You can use **calloc(200, sizeof(char))** to replace malloc function.

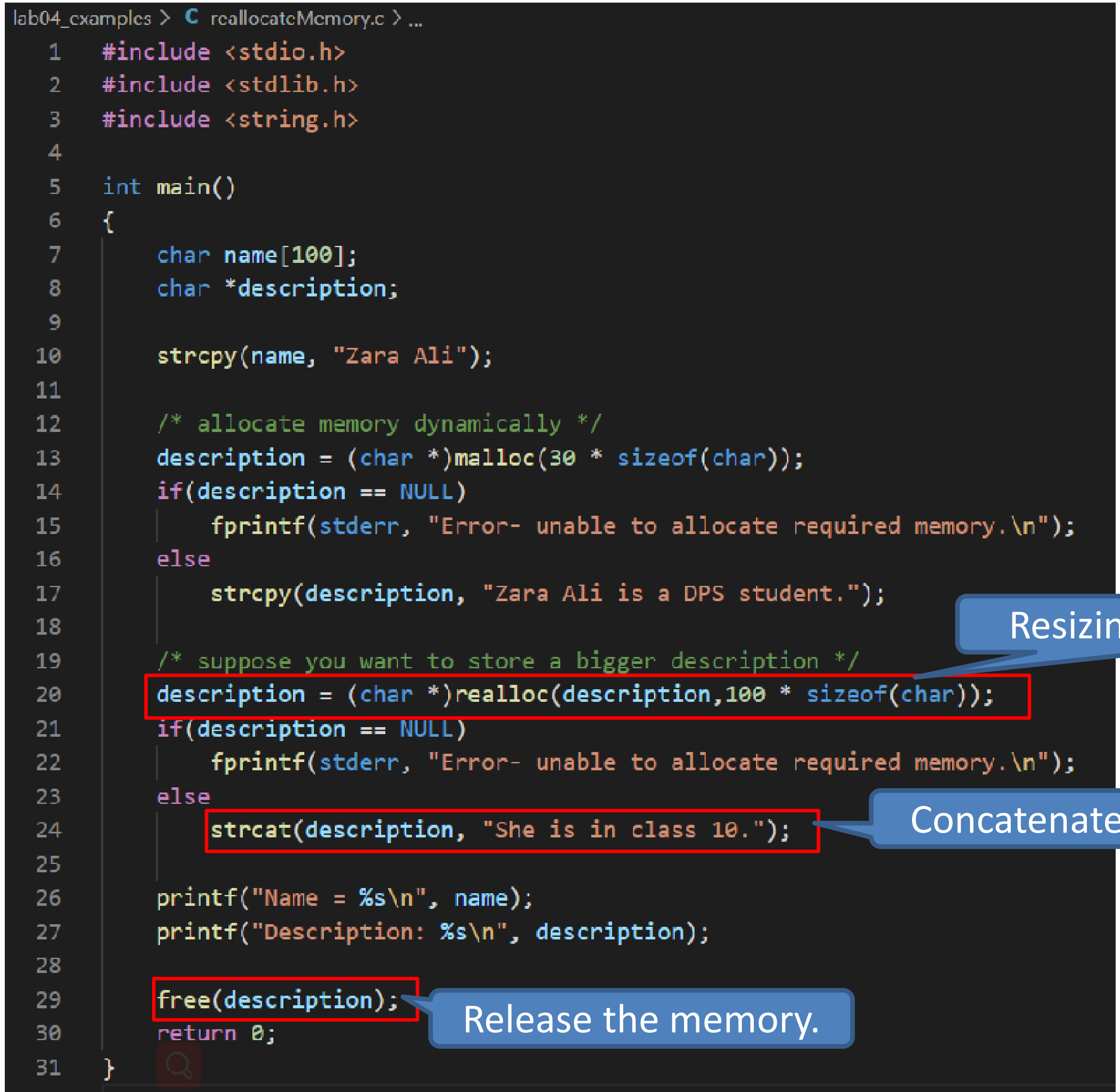
Copy a string to the memory.

Release the memory.

```
Name = Zara Ali
Description: Zara Ali is a DPS student in class 10.
```

2. Resizing Memory

You can increase or decrease the size of an allocated memory block by calling the function **realloc()**.



lab04_examples > C reallocateMemory.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char name[100];
    char *description;

    strcpy(name, "Zara Ali");

    /* allocate memory dynamically */
    description = (char *)malloc(30 * sizeof(char));
    if(description == NULL)
        fprintf(stderr, "Error- unable to allocate required memory.\n");
    else
        strcpy(description, "Zara Ali is a DPS student.");

    /* suppose you want to store a bigger description */
    description = (char *)realloc(description,100 * sizeof(char));
    if(description == NULL)
        fprintf(stderr, "Error- unable to allocate required memory.\n");
    else
        strcat(description, "She is in class 10.");

    printf("Name = %s\n", name);
    printf("Description: %s\n", description);

    free(description);
    return 0;
}
```

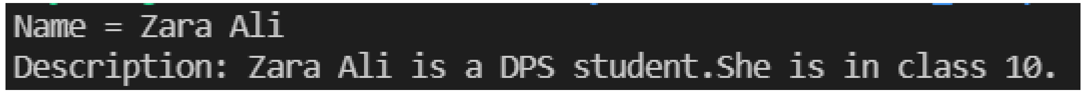
Resizing the memory.

Concatenate the string.

Release the memory.

```c
void *realloc(void *ptr, size_t size);
```

***realloc** deallocates the old object pointed to by **ptr** and returns a pointer to a new object that has the size specified by size. The contents of the new object is identical to that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any bytes in the new object beyond the size of the old object have indeterminate values.*



```
Name = Zara Ali
Description: Zara Ali is a DPS student.She is in class 10.
```

2.2.2 C++ Dynamic Memory

1. new and delete Operators

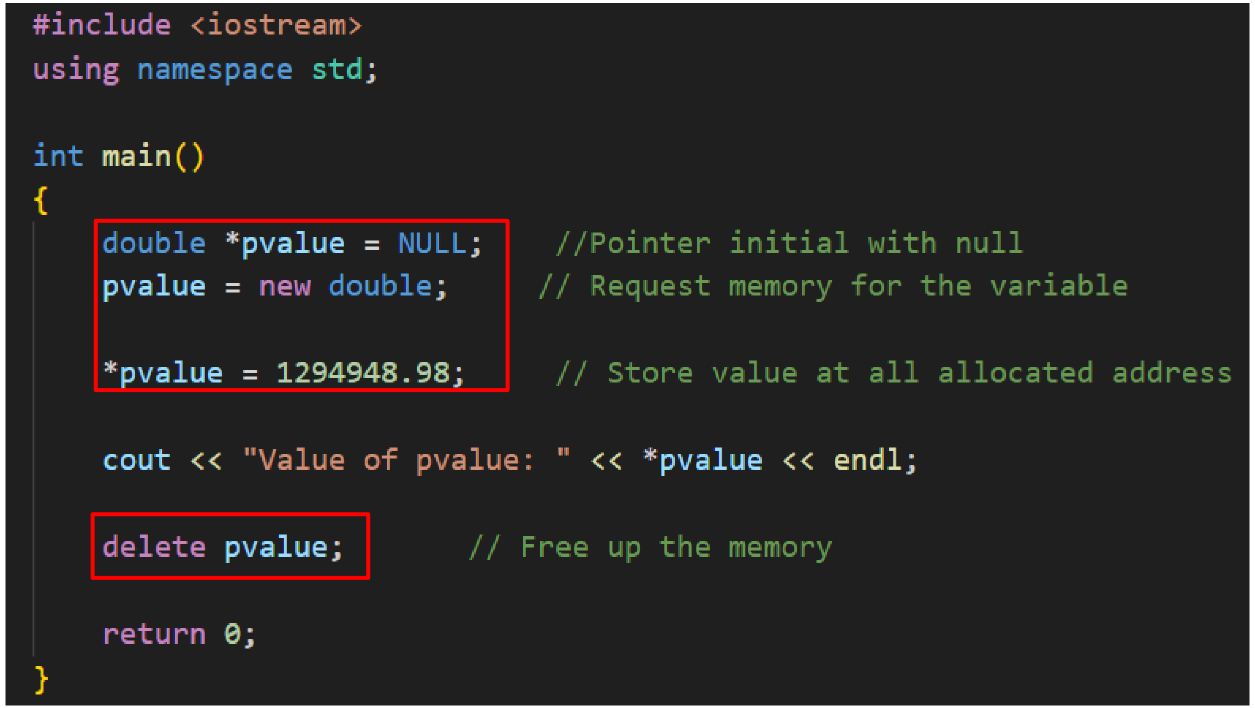
new data-type;

Use **new** operator to allocate memory dynamically for any data-type.

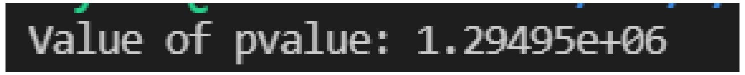
data-type could be any built-in data type including an array or any user defined data types such as structure or class.

delete pointer variable;

Use **delete** operator to de-allocate memory that was previously allocated by new operator.

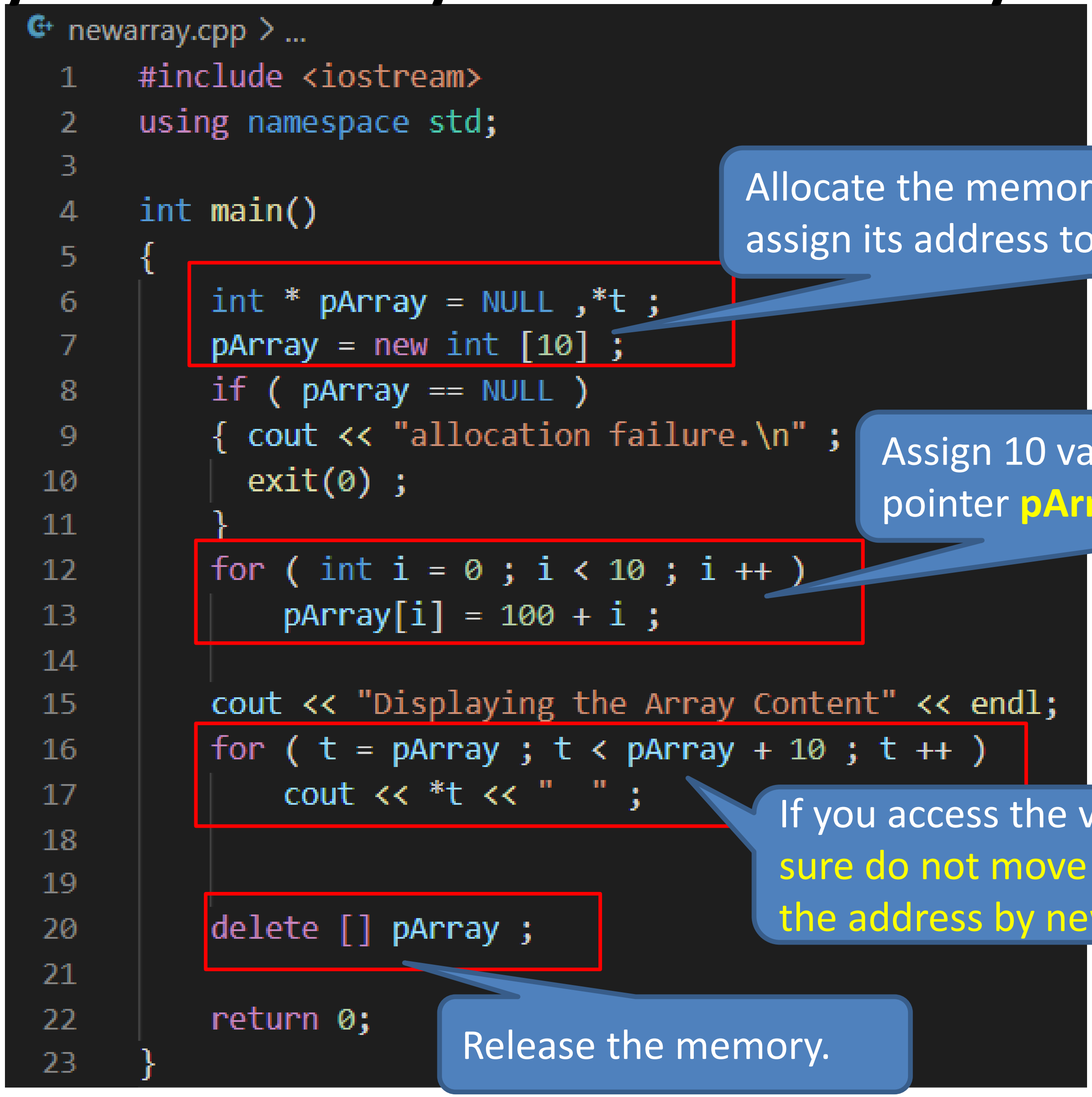


```
#include <iostream>
using namespace std;

int main()
{
    double *pvalue = NULL;     //Pointer initial with null
    pvalue = new double;      // Request memory for the variable

    *pvalue = 1294948.98;      // Store value at all allocated address

    cout << "Value of pvalue: " << *pvalue << endl;

    delete pvalue;          // Free up the memory

    return 0;
}
```



```
Value of pvalue: 1.29495e+06
```

2. Dynamic Memory Allocation for Arrays



newarray.cpp > ...

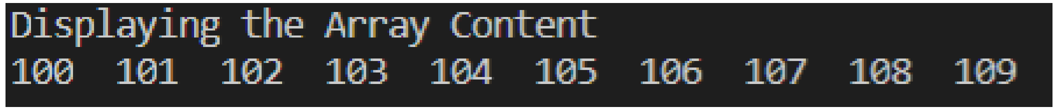
```cpp
#include <iostream>
using namespace std;

int main()
{
    int * pArray = NULL ,*t ;
    pArray = new int [10] ;
    if ( pArray == NULL )
    { cout << "allocation failure.\n" ;
      exit(0) ;
    }
    for ( int i = 0 ; i < 10 ; i ++ )
        pArray[i] = 100 + i ;

    cout << "Displaying the Array Content" << endl;
    for ( t = pArray ; t < pArray + 10 ; t ++ )
        cout << *t << "  " ;


    delete [] pArray ;

    return 0;
}
```

Allocate the memory to store 10 integers, and assign its address to the pointer **pArray**.

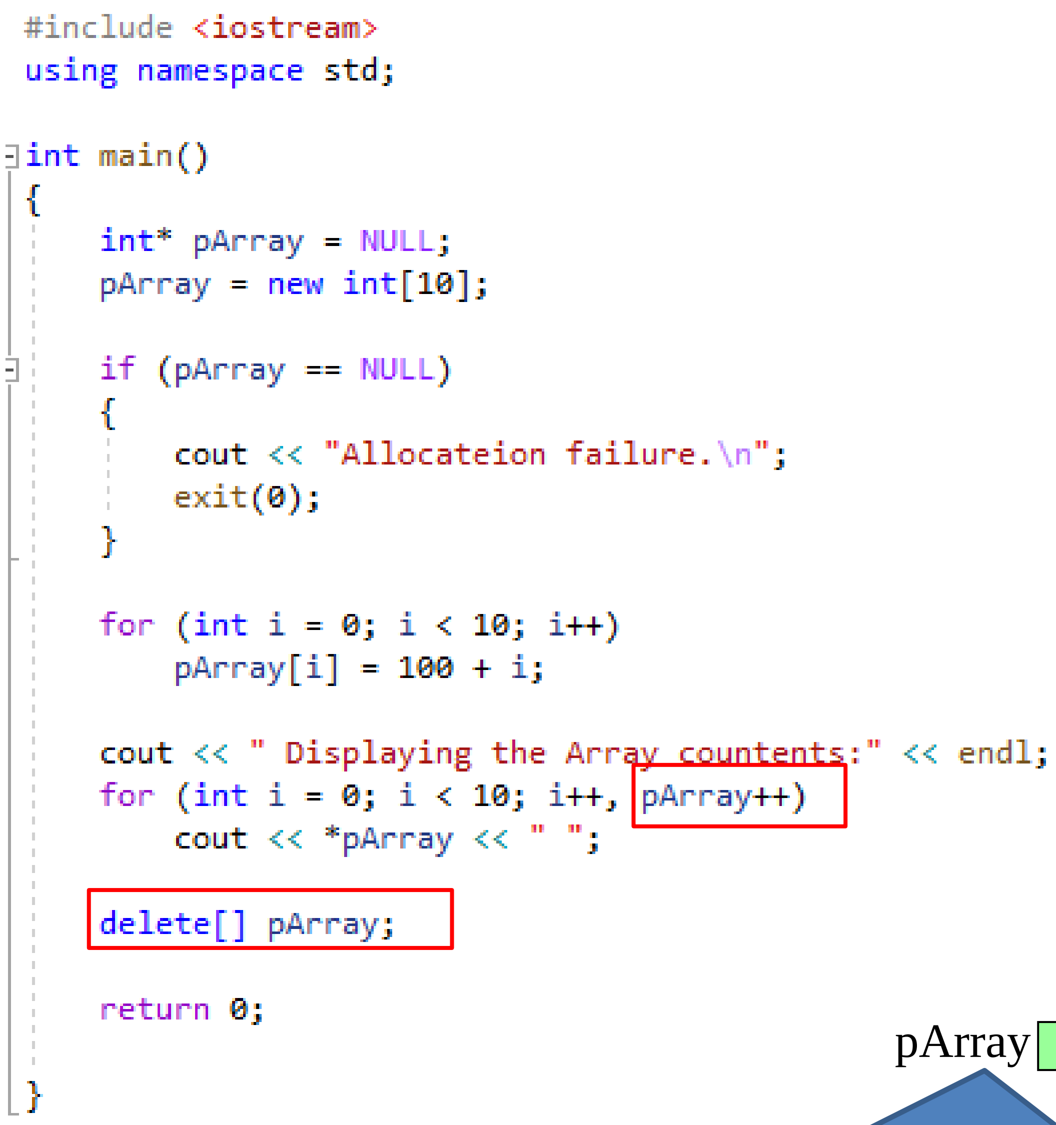
Assign 10 values to the memory by the pointer **pArray**.

If you access the value by * operator, be sure do not move the pointer which assign the address by new.

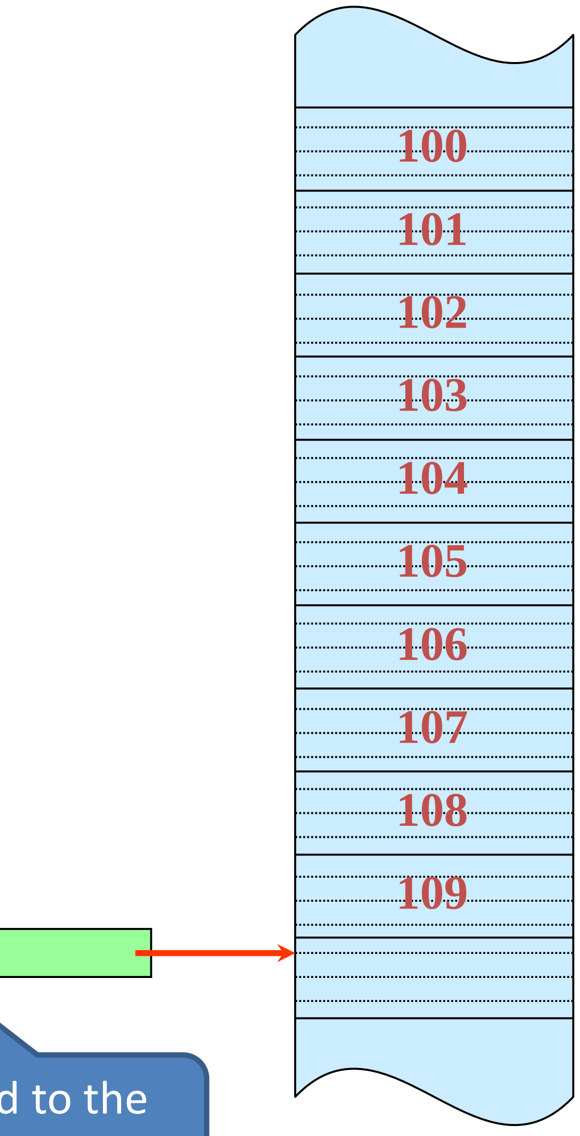
Release the memory.



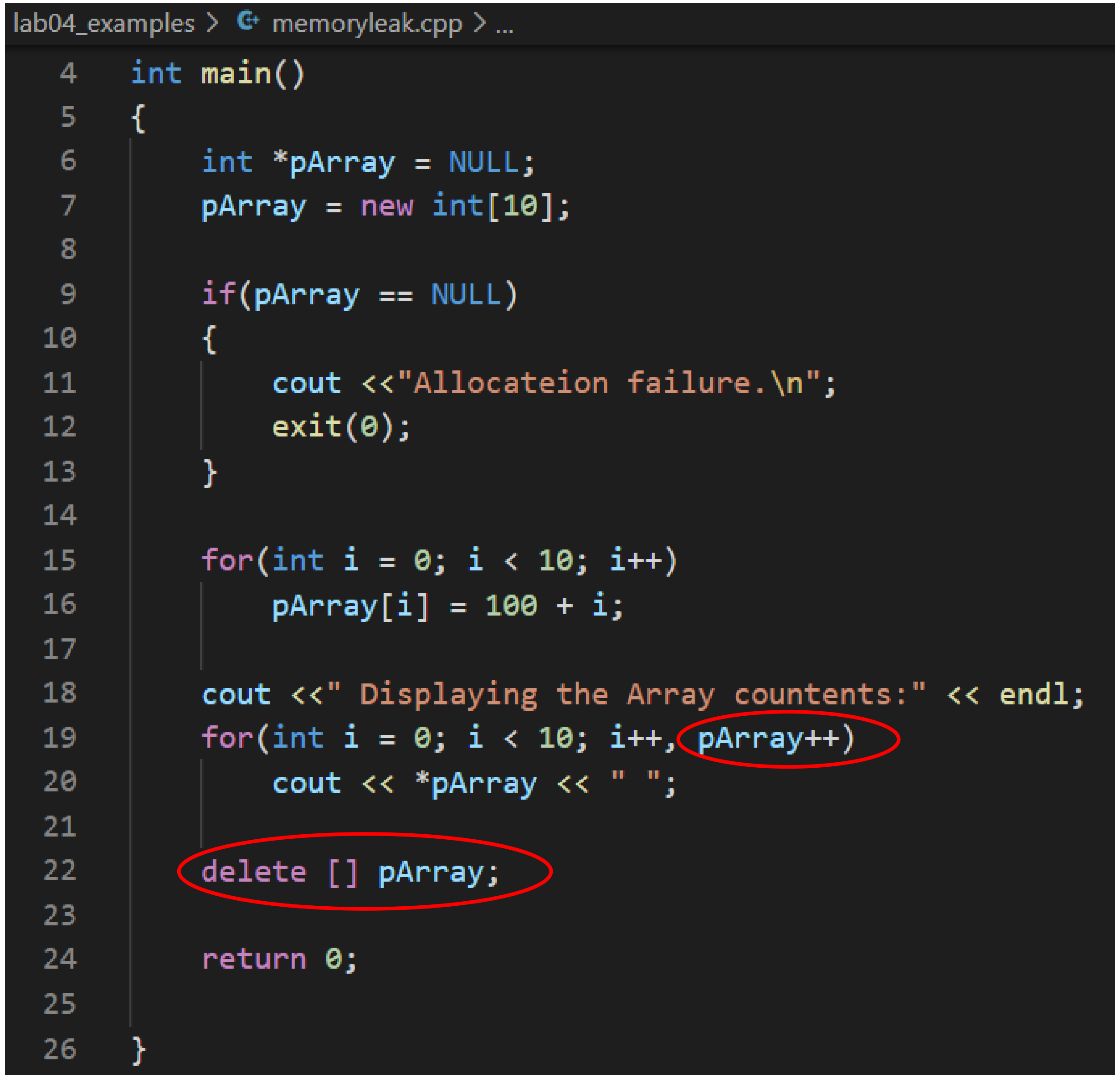
```
Displaying the Array Content
100  101  102  103  104  105  106  107  108  109
```



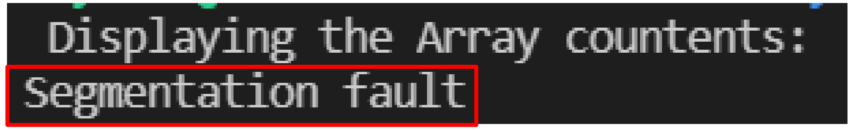
```
#include <iostream>
using namespace std;

int main()
{
    int* pArray = NULL;
    pArray = new int[10];

    if (pArray == NULL)
    {
        cout << "Allocateion failure.\n";
        exit(0);
    }

    for (int i = 0; i < 10; i++)
        pArray[i] = 100 + i;

    cout << " Displaying the Array countents:" << endl;
    for (int i = 0; i < 10; i++, pArray++)
        cout << *pArray << " ";

    delete[] pArray;

    return 0;

}
```



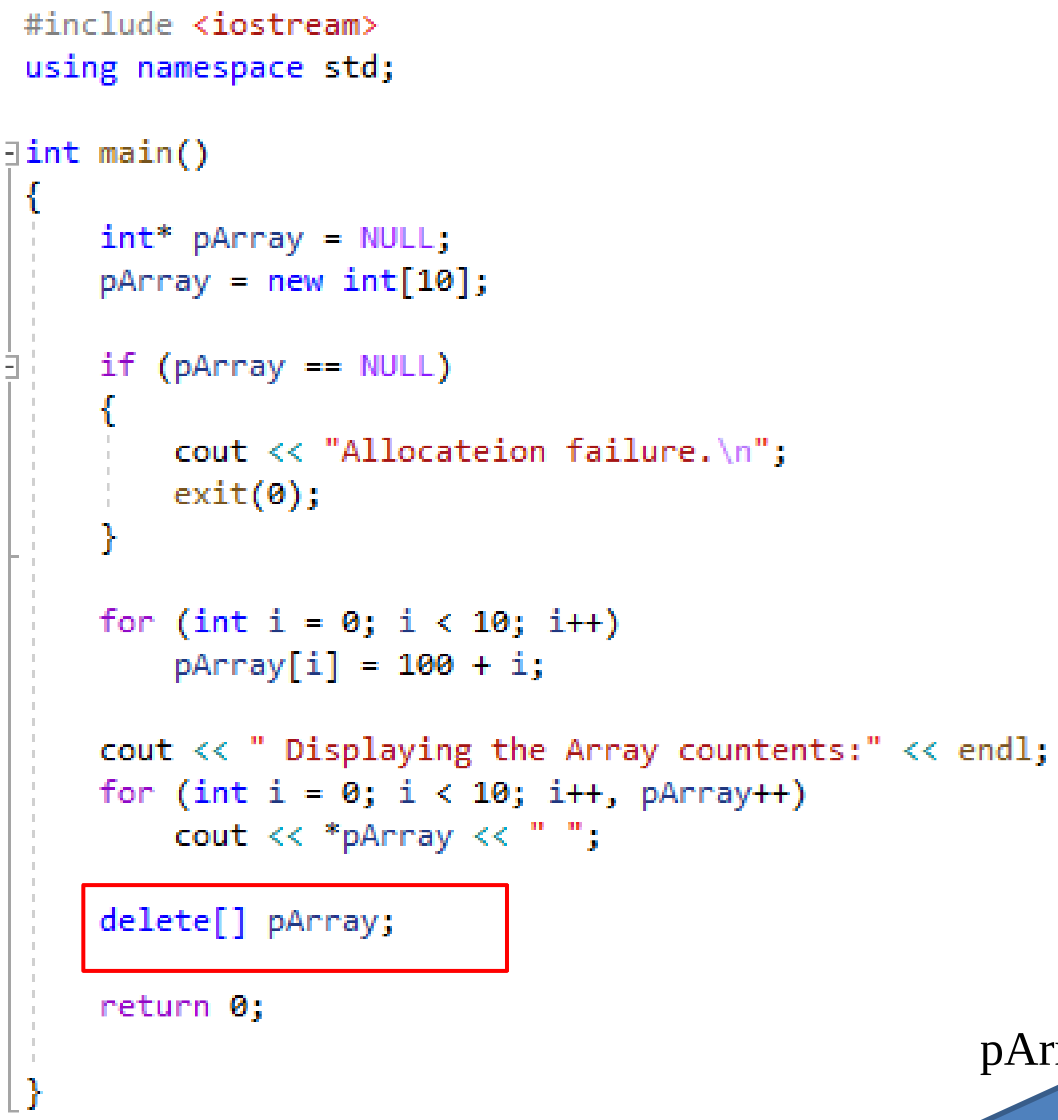
After for loop, the pointer is now pointed to the memory out of the range you have requested.



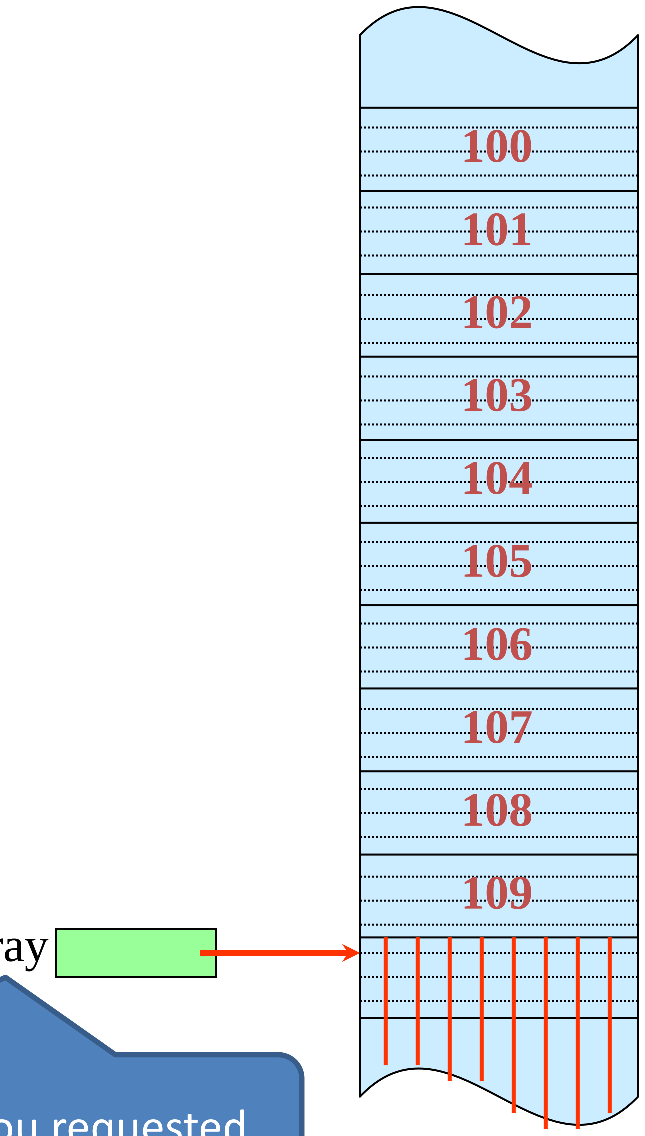
```cpp
int main()
{
    int *pArray = NULL;
    pArray = new int[10];

    if(pArray == NULL)
    {
        cout <<"Allocateion failure.\n";
        exit(0);
    }

    for(int i = 0; i < 10; i++)
        pArray[i] = 100 + i;

    cout <<" Displaying the Array countents:" << endl;
    for(int i = 0; i < 10; i++, pArray++)
        cout << *pArray << " ";

    delete [] pArray;

    return 0;

}
```



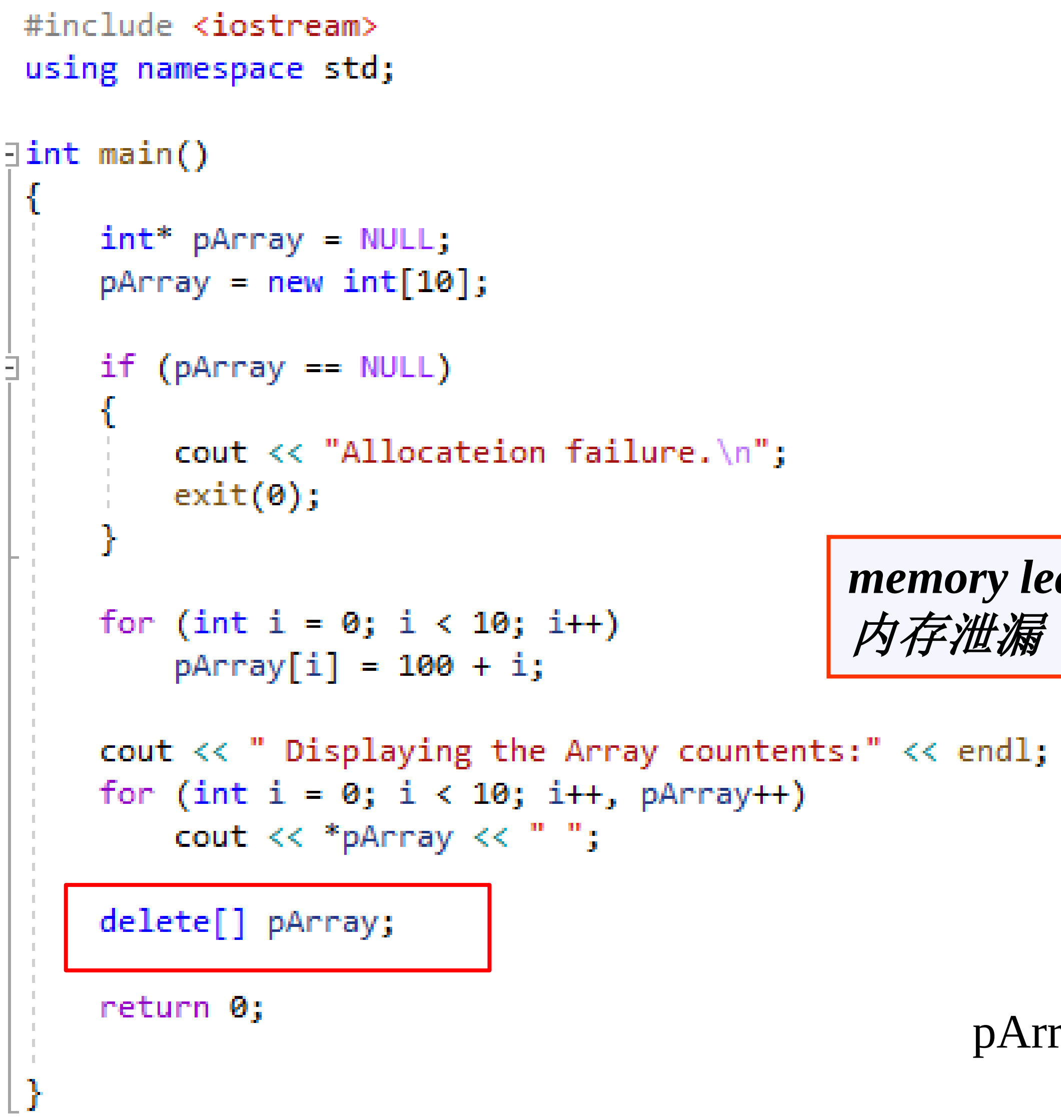
```
 Displaying the Array countents:
Segmentation fault
```



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

int main()
{
    int* pArray = NULL;
    pArray = new int[10];

    if (pArray == NULL)
    {
        cout << "Allocateion failure.\n";
        exit(0);
    }

    for (int i = 0; i < 10; i++)
        pArray[i] = 100 + i;

    cout << " Displaying the Array countents:" << endl;
    for (int i = 0; i < 10; i++, pArray++)
        cout << *pArray << " ";

    delete[] pArray;

    return 0;

}
```

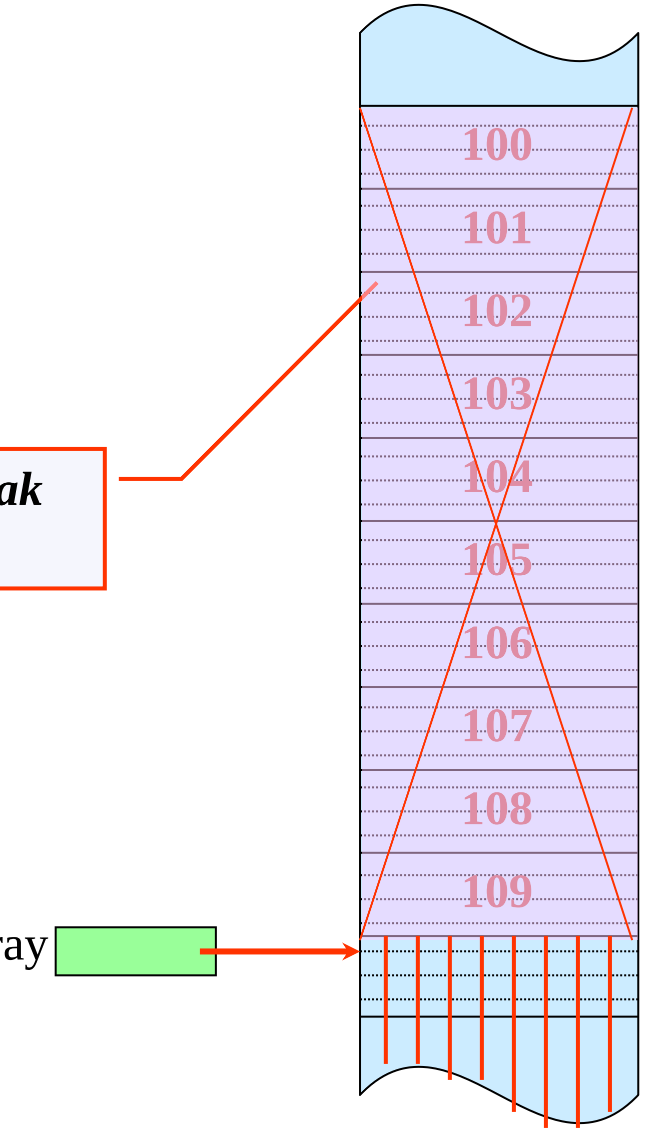


The memory you release will not what you requested.

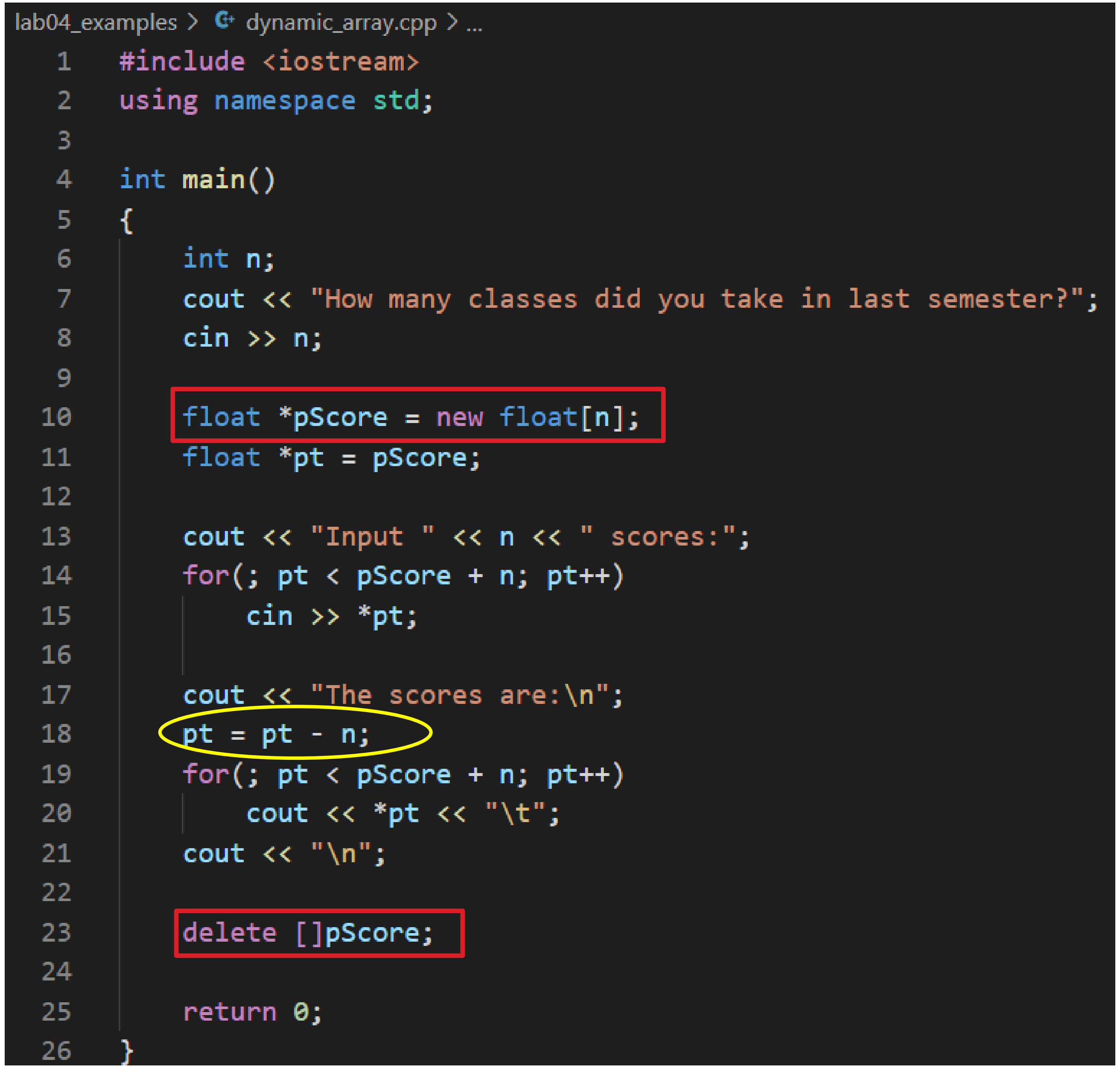


```
#include <iostream>
using namespace std;

int main()
{
    int* pArray = NULL;
    pArray = new int[10];

    if (pArray == NULL)
    {
        cout << "Allocateion failure.\n";
        exit(0);
    }

    for (int i = 0; i < 10; i++)
        pArray[i] = 100 + i;

    cout << " Displaying the Array countents:" << endl;
    for (int i = 0; i < 10; i++, pArray++)
        cout << *pArray << " ";

    delete[] pArray;

    return 0;

}
```

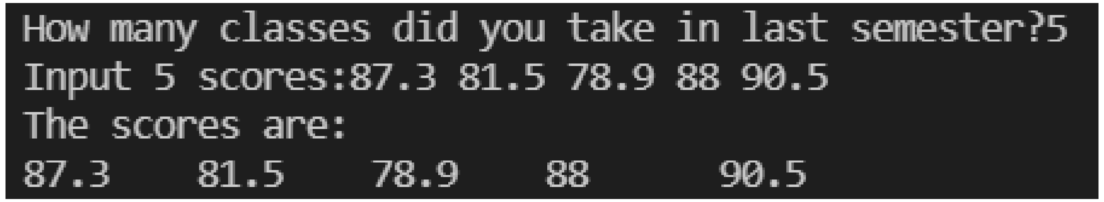
memory leak
内存泄漏



Many times, you are not aware in advance how much memory you will need to store particular information in a defined variable, but the size of required memory can be determined at run time.

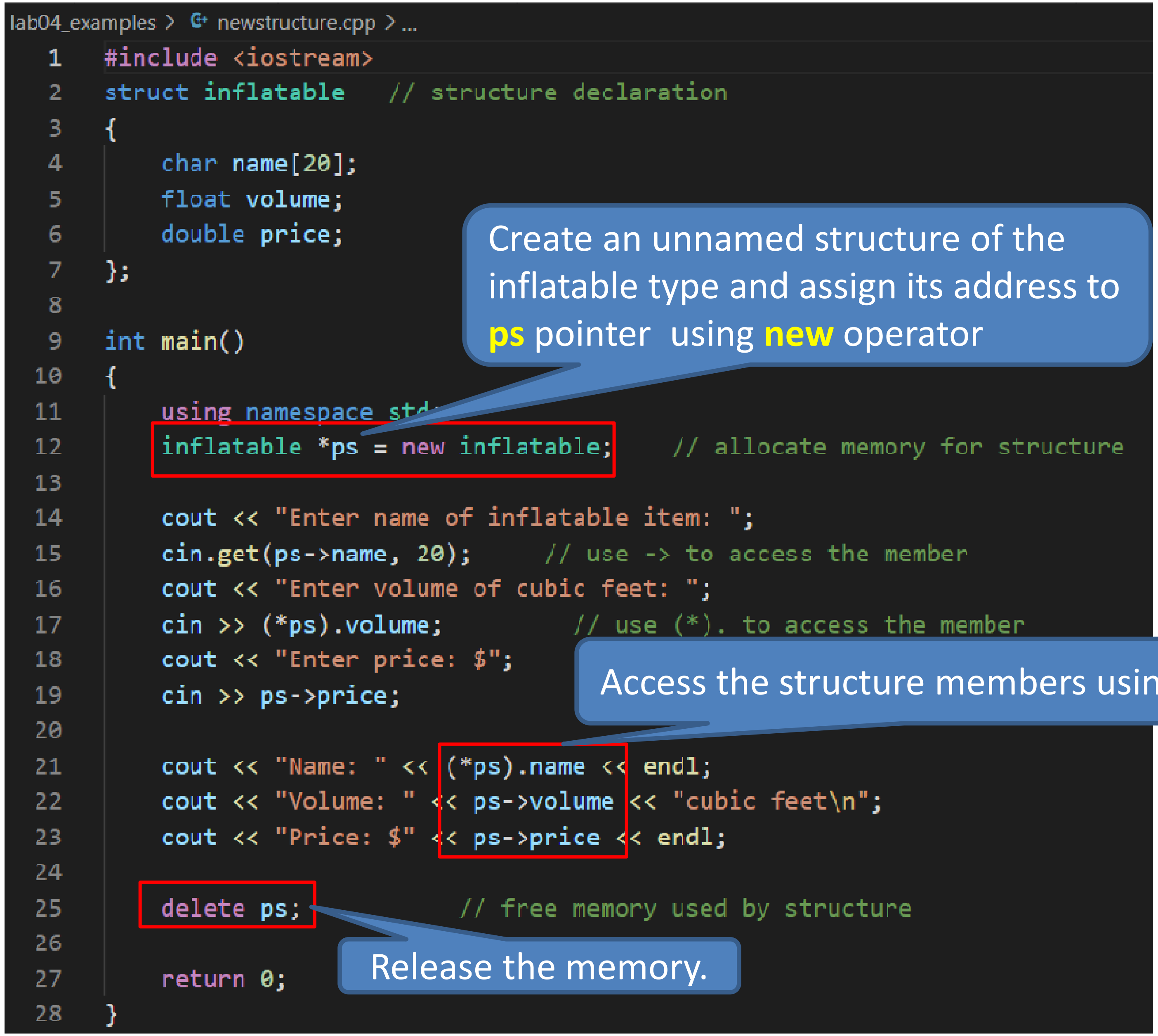


```
#include <iostream>
using namespace std;

int main()
{
    int n;
    cout << "How many classes did you take in last semester?";
    cin >> n;

    float *pScore = new float[n];
    float *pt = pScore;

    cout << "Input " << n << " scores:";
    for(; pt < pScore + n; pt++)
        cin >> *pt;

    cout << "The scores are:\n";
    pt = pt - n;
    for(; pt < pScore + n; pt++)
        cout << *pt << "\t";
    cout << "\n";

    delete []pScore;

    return 0;
}
```



```
How many classes did you take in last semester?5
Input 5 scores:87.3 81.5 78.9 88 90.5
The scores are:
87.3    81.5    78.9    88      90.5
```

3. Dynamic Memory Allocation for Structures



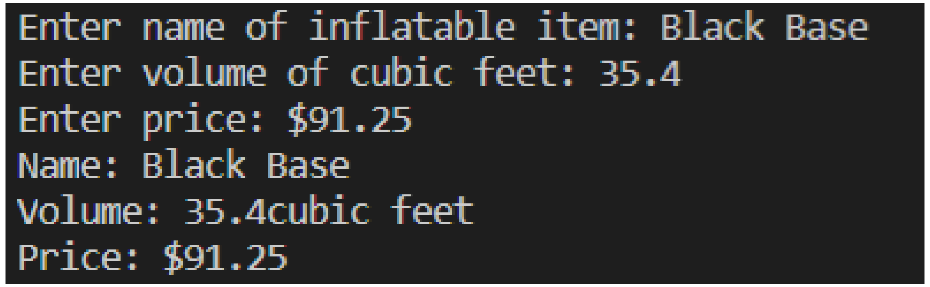
lab04_examples > newstructure.cpp > ...

```cpp
#include <iostream>
struct inflatable   // structure declaration
{
    char name[20];
    float volume;
    double price;
};

int main()
{
    using namespace std;
    inflatable *ps = new inflatable;    // allocate memory for structure

    cout << "Enter name of inflatable item: ";
    cin.get(ps->name, 20);     // use -> to access the member
    cout << "Enter volume of cubic feet: ";
    cin >> (*ps).volume;          // use (*). to access the member
    cout << "Enter price: $";
    cin >> ps->price;

    cout << "Name: " << (*ps).name << endl;
    cout << "Volume: " << ps->volume << "cubic feet\n";
    cout << "Price: $" << ps->price << endl;

    delete ps;                 // free memory used by structure

    return 0;
}
```

Create an unnamed structure of the inflatable type and assign its address to **ps** pointer using **new** operator

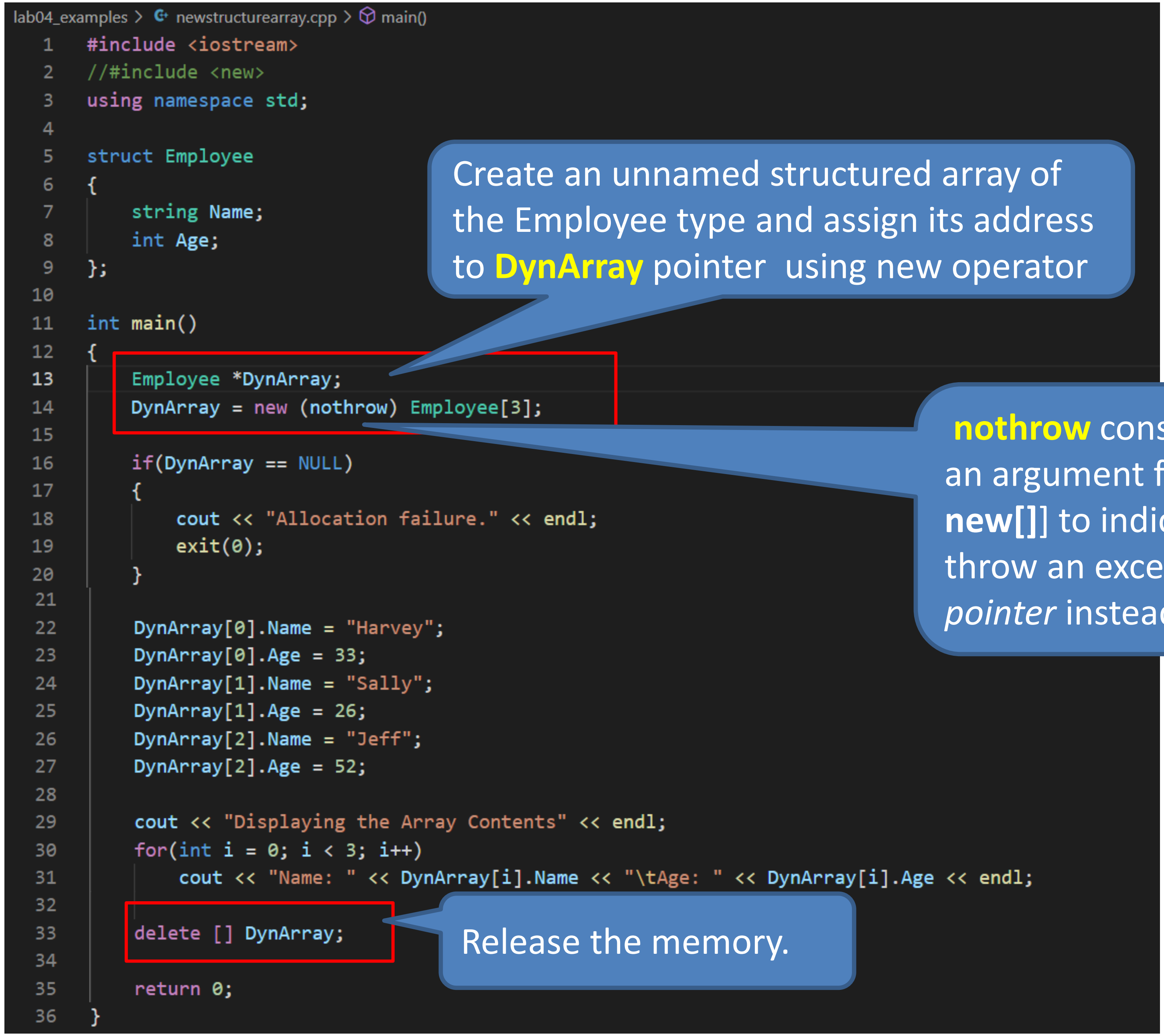
Access the structure members using **->** or **(*).**

Release the memory.



```
Enter name of inflatable item: Black Base
Enter volume of cubic feet: 35.4
Enter price: $91.25
Name: Black Base
Volume: 35.4cubic feet
Price: $91.25
```

Structured array



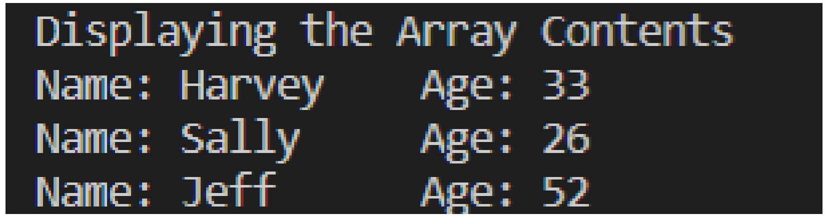
lab04_examples > newstructurearray.cpp > main()

```cpp
#include <iostream>
//#include <new>
using namespace std;

struct Employee
{
    string Name;
    int Age;
};

int main()
{
    Employee *DynArray;
    DynArray = new (nothrow) Employee[3];

    if(DynArray == NULL)
    {
        cout << "Allocation failure." << endl;
        exit(0);
    }

    DynArray[0].Name = "Harvey";
    DynArray[0].Age = 33;
    DynArray[1].Name = "Sally";
    DynArray[1].Age = 26;
    DynArray[2].Name = "Jeff";
    DynArray[2].Age = 52;

    cout << "Displaying the Array Contents" << endl;
    for(int i = 0; i < 3; i++)
        cout << "Name: " << DynArray[i].Name << "\tAge: " << DynArray[i].Age << endl;

    delete [] DynArray;

    return 0;
}
```

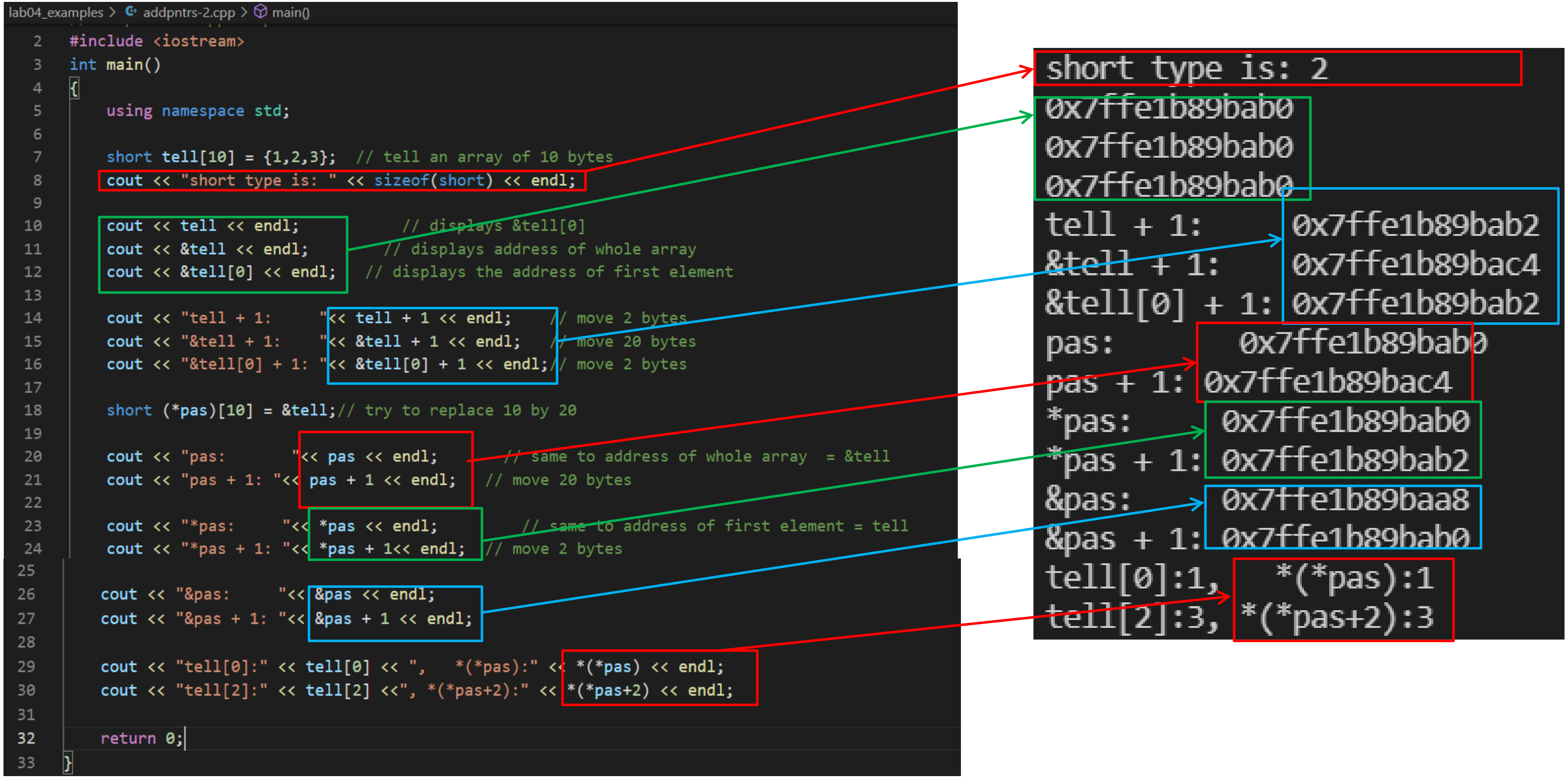
Create an unnamed structured array of the Employee type and assign its address to **DynArray** pointer using new operator

nothrow constant, this constant value is used as an argument for [**operator new**] and [**operator new[]**] to indicate that these functions shall not throw an exception on failure, but return a *null pointer* instead.

Release the memory.



```
Displaying the Array Contents
Name: Harvey    Age: 33
Name: Sally     Age: 26
Name: Jeff      Age: 52
```



```
#include <iostream>
int main()
{
    using namespace std;

    short tell[10] = {1,2,3};  // tell an array of 10 bytes
    cout << "short type is: " << sizeof(short) << endl;

    cout << tell << endl;          // displays &tell[0]
    cout << &tell << endl;        // displays address of whole array
    cout << &tell[0] << endl;   // displays the address of first element

    cout << "tell + 1:      "<< tell + 1 << endl;      // move 2 bytes
    cout << "&tell + 1:     "<< &tell + 1 << endl;     // move 20 bytes
    cout << "&tell[0] + 1:  "<< &tell[0] + 1 << endl;// move 2 bytes

    short (*pas)[10] = &tell;// try to replace 10 by 20

    cout << "pas:       "<< pas << endl;       // same to address of whole array  = &tell
    cout << "pas + 1: "<< pas + 1 << endl;  // move 20 bytes

    cout << "*pas:     "<< *pas << endl;      // same to address of first element = tell
    cout << "*pas + 1: "<< *pas + 1<< endl;  // move 2 bytes

    cout << "&pas:     "<< &pas << endl;
    cout << "&pas + 1: "<< &pas + 1 << endl;

    cout << "tell[0]:" << tell[0] << ",   *(*pas):" << *(*pas) << endl;
    cout << "tell[2]:" << tell[2] <<", *(*pas+2):" << *(*pas+2) << endl;

    return 0;
}
```

```
short type is: 2
0x7ffe1b89bab0
0x7ffe1b89bab0
0x7ffe1b89bab0
tell + 1:      0x7ffe1b89bab2
&tell + 1:     0x7ffe1b89bac4
&tell[0] + 1:  0x7ffe1b89bab2
pas:       0x7ffe1b89bab0
pas + 1: 0x7ffe1b89bac4
*pas:     0x7ffe1b89bab0
*pas + 1: 0x7ffe1b89bab2
&pas:     0x7ffe1b89baa8
&pas + 1: 0x7ffe1b89bab0
tell[0]:1,   *(*pas):1
tell[2]:3, *(*pas+2):3
```

Exercise 1

```
#include<stdio.h>

int main()
{
   int a[]={2,4,6,8,10},y=1,*p;
   p=&a[1];

   printf("a = %p\np = %p\n",a, p);

   for(int i = 0; i < 3; i++)
      y += *(p+i);

   printf("y = %d\n\n",y);

   int b[5]={1,2,3,4,5};
   int *ptr=(int*)(&b+1);

   printf("b = %p\nb+4 = %p\nptr = %p\n",b,b+4,ptr);
   printf("%d,%d\n",*(b+1),*(ptr-1));

   return 0;

}
```

Run the program and explain the result to SA.

Exercise 2

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

int main()
{
   int a[][4]={1,3,5,7,9,11,13,15,17,19};
   int *p=*(a+1);
   p += 3;
   cout << "*p++ = " << *p++ << ",*p = " << *p << endl;

   const char *pc = "Welcome to programming.", *r;
   long *q = (long *)pc;
   q++;
   r = (char *)q;

   cout << r << endl;

   unsigned int m = 0x3E56AF67;
   unsigned short *pm = (unsigned short *) &m;

   cout << "*pm = " <<  hex << *pm << endl;

   return 0;
}
```

Run the program and explain the result to SA.

Exercise 3

```
#include <stdio.h>


int main()
{
   int aa[2][5] = { 1,2,3,4,5,6,7,8,9,10 };
   int* paa1 = (int*)(&aa + 1);
   int* paa2 = (int*)(*(aa + 1));
   printf("%d,%d\n", *(paa1 - 1), *(paa2 - 1));

   char* str[] = { "work","at","alibaba" };
   char** ps = str;
   ps++;
   printf("%s\n", *ps);

   return 0;
}
```

Run the program and explain the result to SA.

Exercise 4

Write a program that use ***new*** to allocate the array dynamically for five integers.

- The five values will be stored in an array using a pointer.
- Print the elements of the array in reverse order using a pointer.