Advanced Programming

Lab 04

CONTENTS

- ✓ Objectives
- ✓ Knowledge points
- ✓ Exercises

1 Objectives

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

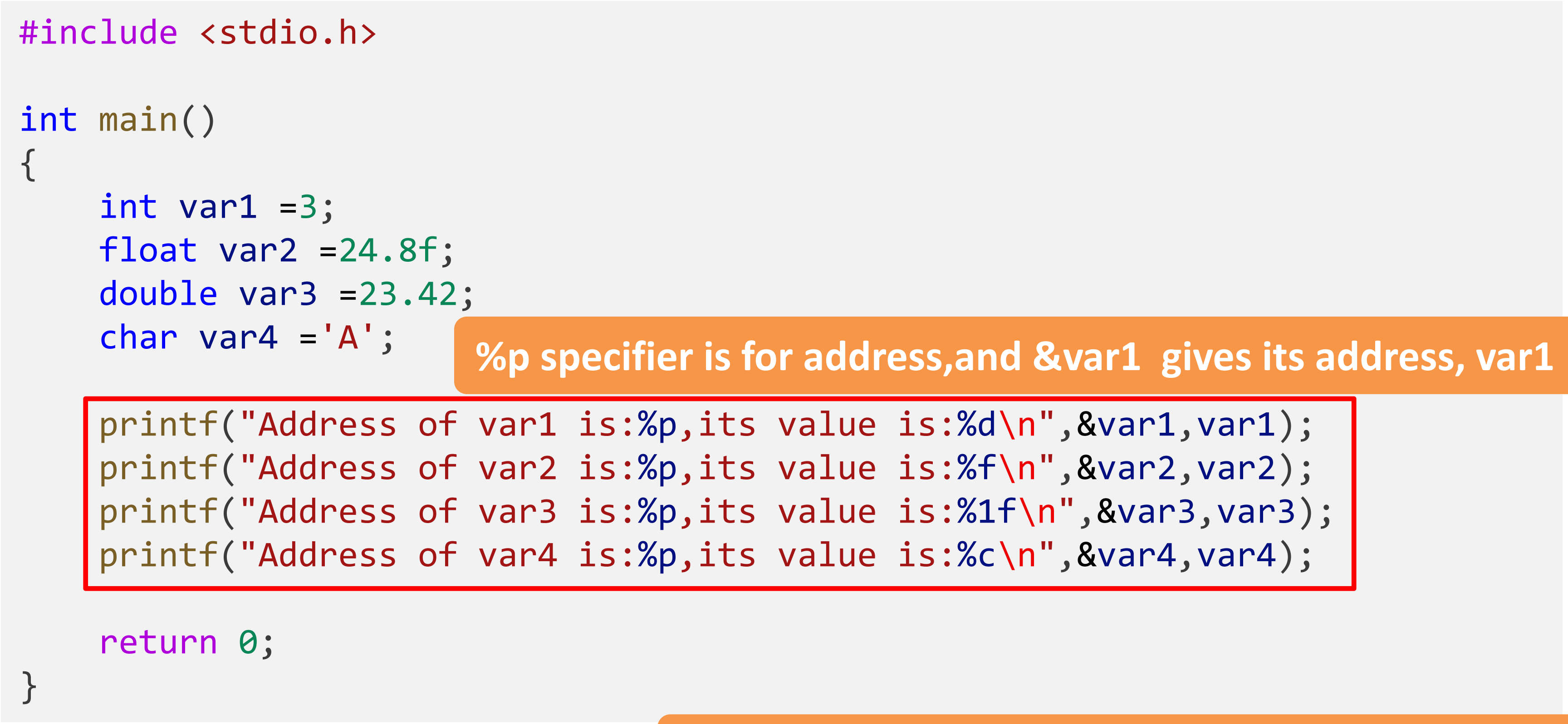
2 Knowledge Points

2.1 Pointers

2.2 Dynamic memory management

2.1 Pointers

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

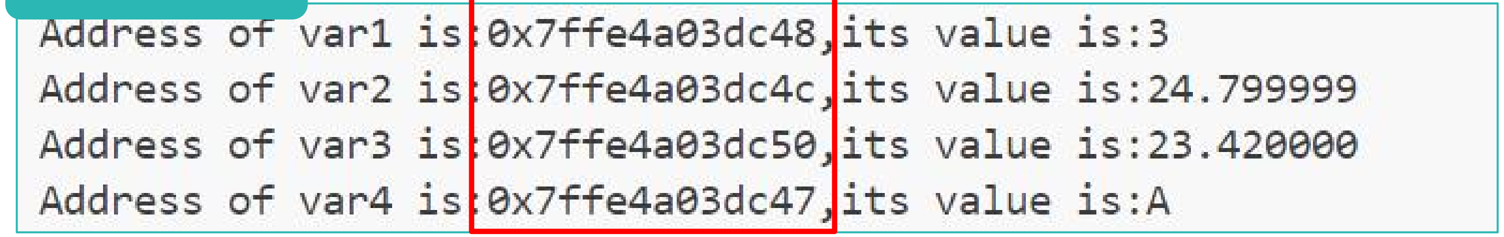


```c
#include <stdio.h>

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:%1f\n",&var3,var3);
    printf("Address of var4 is:%p,its value is:%c\n",&var4,var4);

    return 0;
}
```

%p specifier is for address,and &var1 gives its address, var1 is value.

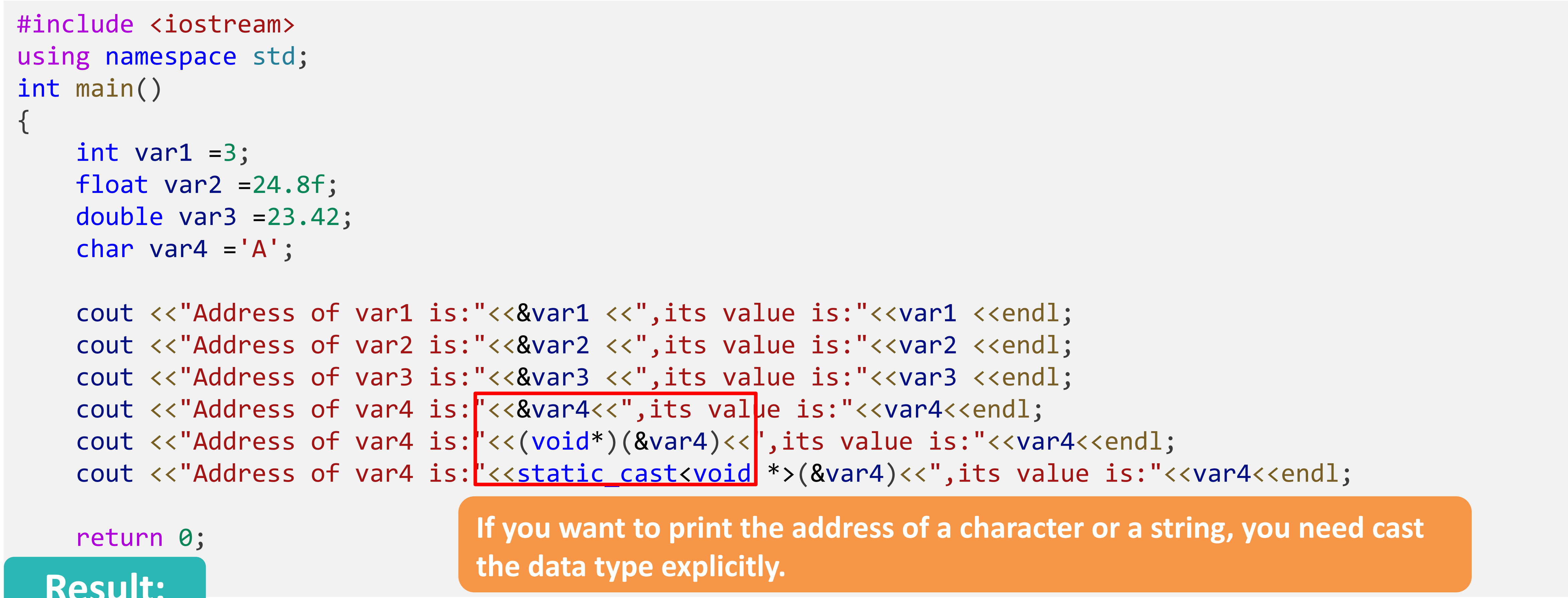
System uses hexadecimal notation when displaying address values.

Result:



```
Address of var1 is:0x7ffe4a03dc48,its value is:3
Address of var2 is:0x7ffe4a03dc4c,its value is:24.799999
Address of var3 is:0x7ffe4a03dc50,its value is:23.420000
Address of var4 is:0x7ffe4a03dc47,its value is:A
```

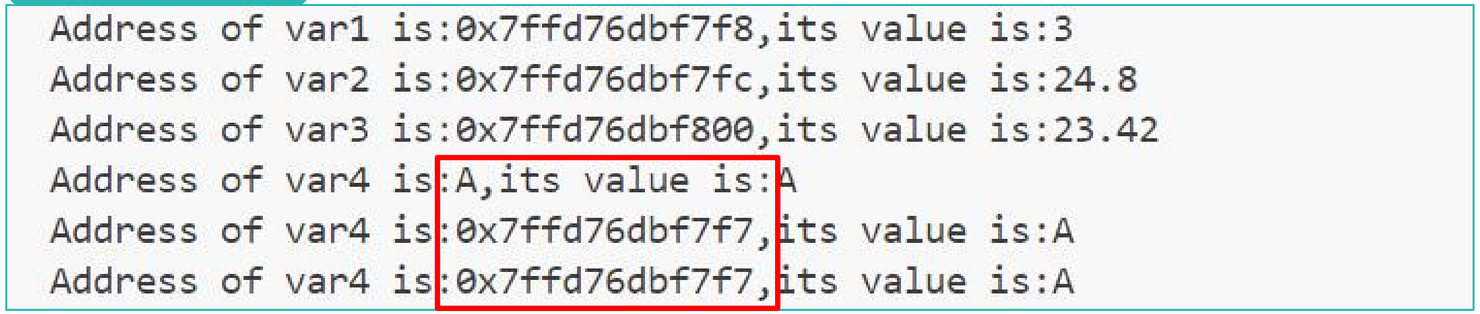
C++ Expression



```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;
    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;

    return 0;
```

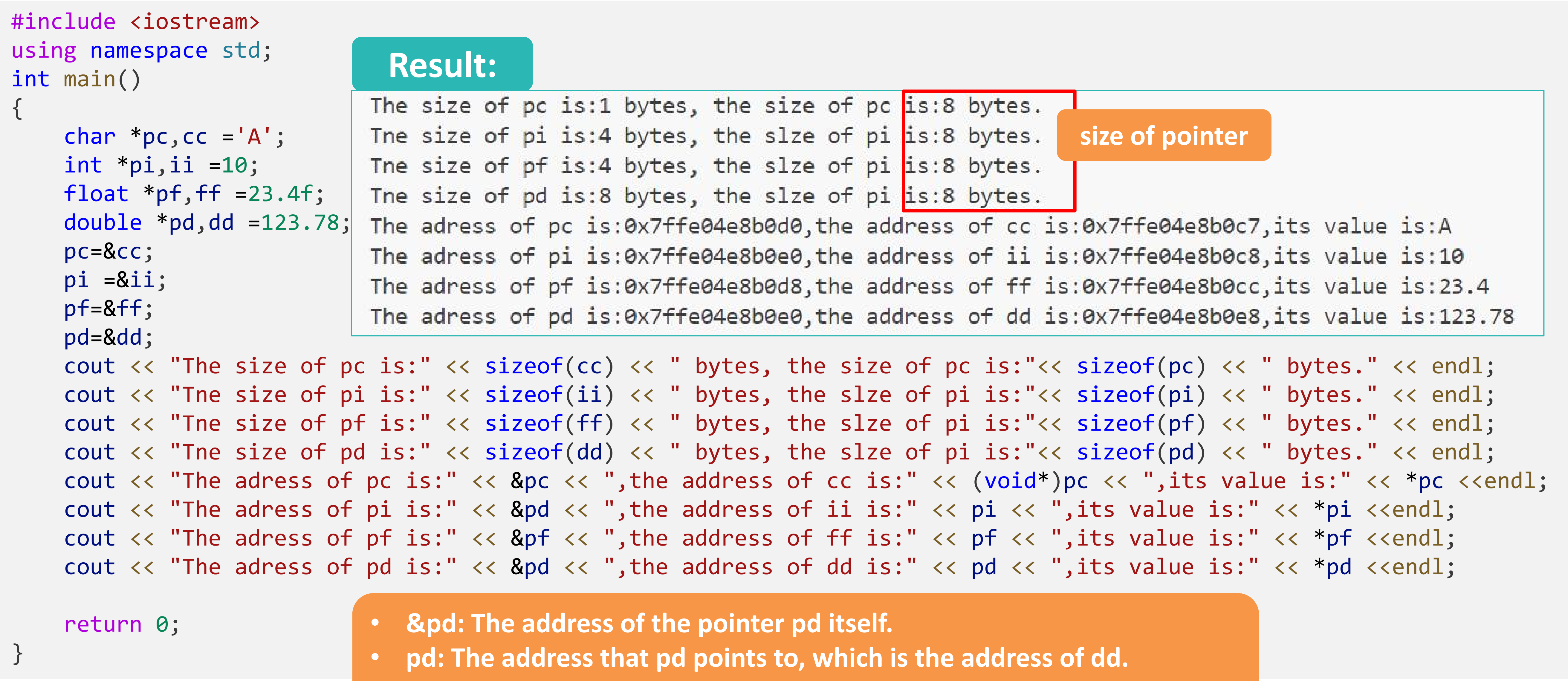
If you want to print the address of a character or a string, you need cast the data type explicitly.

Result:



```
Address of var1 is:0x7ffd76dbf7f8,its value is:3
Address of var2 is:0x7ffd76dbf7fc,its value is:24.8
Address of var3 is:0x7ffd76dbf800,its value is:23.42
Address of var4 is:A,its value is:A
Address of var4 is:0x7ffd76dbf7f7,its value is:A
Address of var4 is:0x7ffd76dbf7f7,its value is:A
```

Size and Address of Pointer



```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 pc is:" << sizeof(cc) << " bytes, the size of pc is:"<< sizeof(pc) << " bytes." << endl;
    cout << "Tne size of pi is:" << sizeof(ii) << " bytes, the slze of pi is:"<< sizeof(pi) << " bytes." << endl;
    cout << "Tne size of pf is:" << sizeof(ff) << " bytes, the slze of pi is:"<< sizeof(pf) << " bytes." << endl;
    cout << "Tne size of pd is:" << sizeof(dd) << " bytes, the slze of pi is:"<< sizeof(pd) << " bytes." << endl;
    cout << "The adress of pc is:" << &pc << ",the address of cc is:" << (void*)pc << ",its value is:" << *pc <<endl;
    cout << "The adress of pi is:" << &pd << ",the address of ii is:" << pi << ",its value is:" << *pi <<endl;
    cout << "The adress of pf is:" << &pf << ",the address of ff is:" << pf << ",its value is:" << *pf <<endl;
    cout << "The adress of pd is:" << &pd << ",the address of dd is:" << pd << ",its value is:" << *pd <<endl;

    return 0;
}
```

Result:

```
The size of pc is:1 bytes, the size of pc is:8 bytes.
Tne size of pi is:4 bytes, the slze of pi is:8 bytes.
Tne size of pf is:4 bytes, the slze of pi is:8 bytes.
Tne size of pd is:8 bytes, the slze of pi is:8 bytes.
The adress of pc is:0x7ffe04e8b0d0,the address of cc is:0x7ffe04e8b0c7,its value is:A
The adress of pi is:0x7ffe04e8b0e0,the address of ii is:0x7ffe04e8b0c8,its value is:10
The adress of pf is:0x7ffe04e8b0d8,the address of ff is:0x7ffe04e8b0cc,its value is:23.4
The adress of pd is:0x7ffe04e8b0e0,the address of dd is:0x7ffe04e8b0e8,its value is:123.78
```

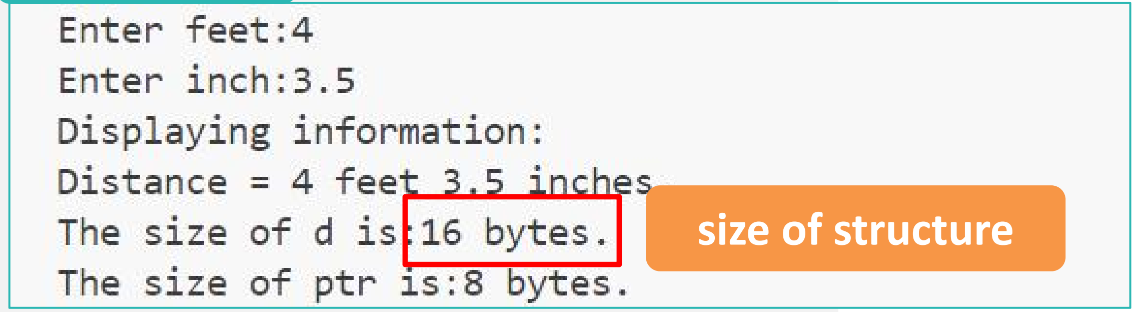
size of pointer

- **&pd: The address of the pointer pd itself.**
- **pd: The address that pd points to, which is the address of dd.**
- ***pd: The value that pd points to, which is the value of dd, 123.78.**

Pointer and structure

```
#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;
}
```

Result:

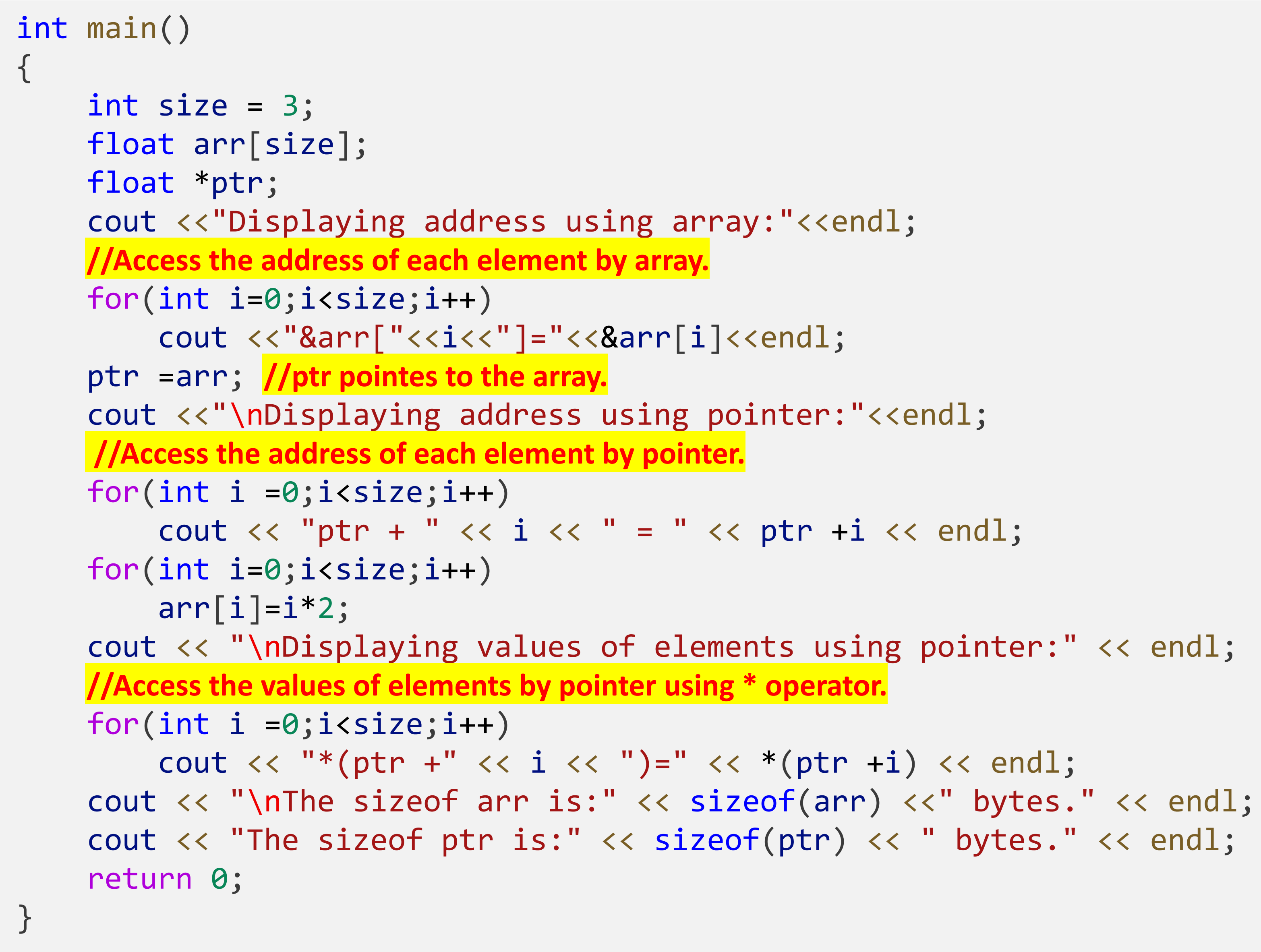


```
Enter feet:4
Enter inch:3.5
Displaying information:
Distance = 4 feet 3.5 inches
The size of d is:16 bytes.
The size of ptr is:8 bytes.
```

size of structure

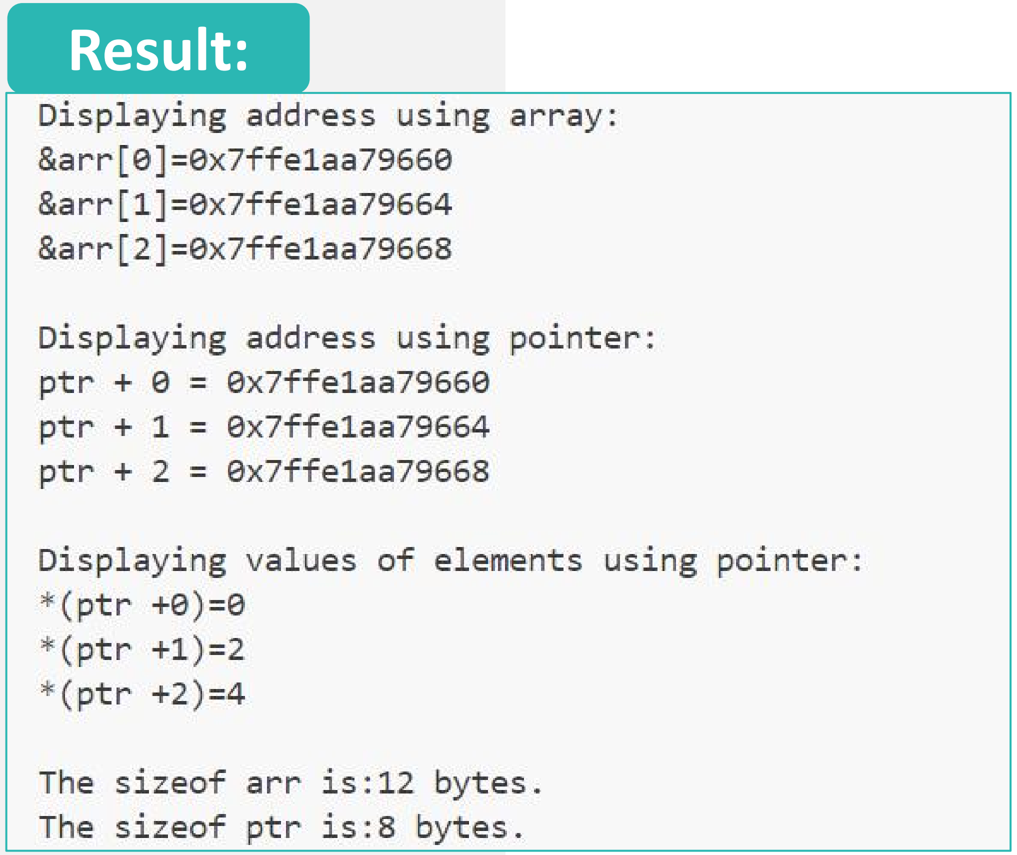
Note: Since pointer ptr is pointed to variable d , (*ptr).inch ,ptr->inch and d.inch are exact the same.

Pointer and array



```
int main()
{
    int size = 3;
    float arr[size];
    float *ptr;
    cout <<"Displaying address using array:"<<endl;
    //Access the address of each element by array.
    for(int i=0;i<size;i++)
        cout <<"&arr["<<i<<"]="<<&arr[i]<<endl;
    ptr =arr;  //ptr pointes to the array.
    cout <<"\nDisplaying address using pointer:"<<endl;
     //Access the address of each element by pointer.
    for(int i =0;i<size;i++)
        cout << "ptr + " << i << " = " << ptr +i << endl;
    for(int i=0;i<size;i++)
        arr[i]=i*2;
    cout << "\nDisplaying values of elements using pointer:" << endl;
    //Access the values of elements by pointer using * operator.
    for(int i =0;i<size;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;
}
```

Result:



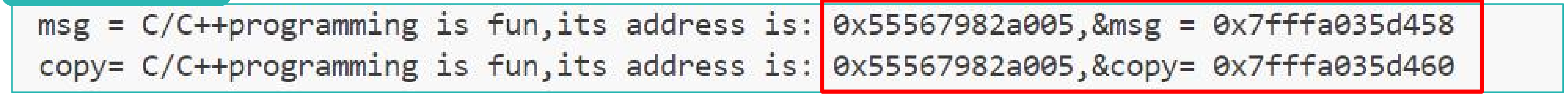
```
Displaying address using array:
&arr[0]=0x7ffe1aa79660
&arr[1]=0x7ffe1aa79664
&arr[2]=0x7ffe1aa79668

Displaying address using pointer:
ptr + 0 = 0x7ffe1aa79660
ptr + 1 = 0x7ffe1aa79664
ptr + 2 = 0x7ffe1aa79668

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

The sizeof arr is:12 bytes.
The sizeof ptr is:8 bytes.
```

Pointer and string

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.

```
#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 << endl;

    return 0;
}
```

Result:

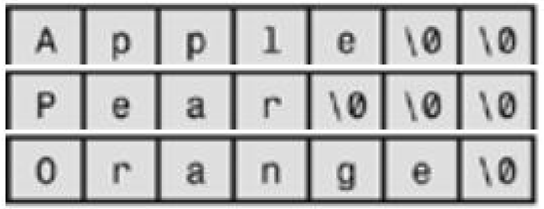


```
msg = C/C++programming is fun,its address is: 0x55567982a005,&msg = 0x7fffa035d458
copy= C/C++programming is fun,its address is: 0x55567982a005,&copy= 0x7fffa035d460
```

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.

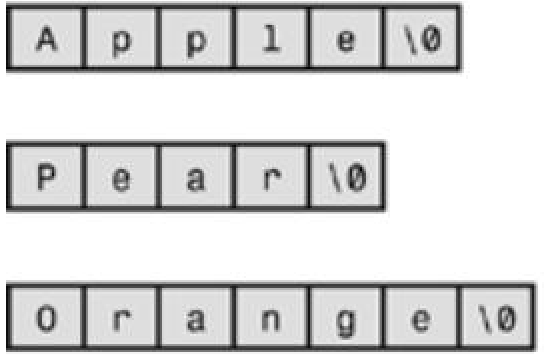
char fruit1[3][7] = { “Apple”, “Pear”, “Orange”};



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.

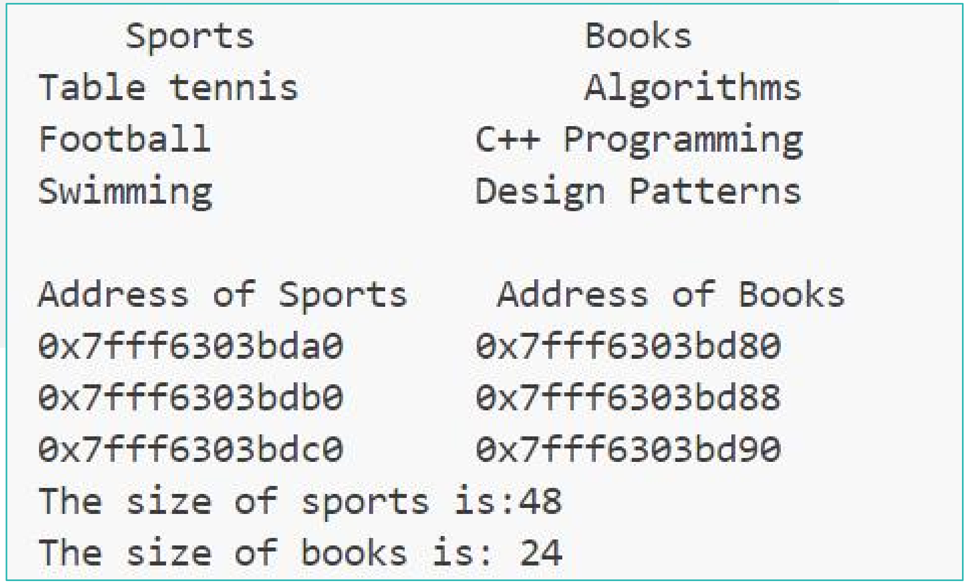
Pointer Array: address and size

```cpp
#include <iostream>
#include <iomanip>
#include <cstring>
using namespace std;
 int main()
 {
    char sports[3][16]={"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) << endl;
    cout << "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

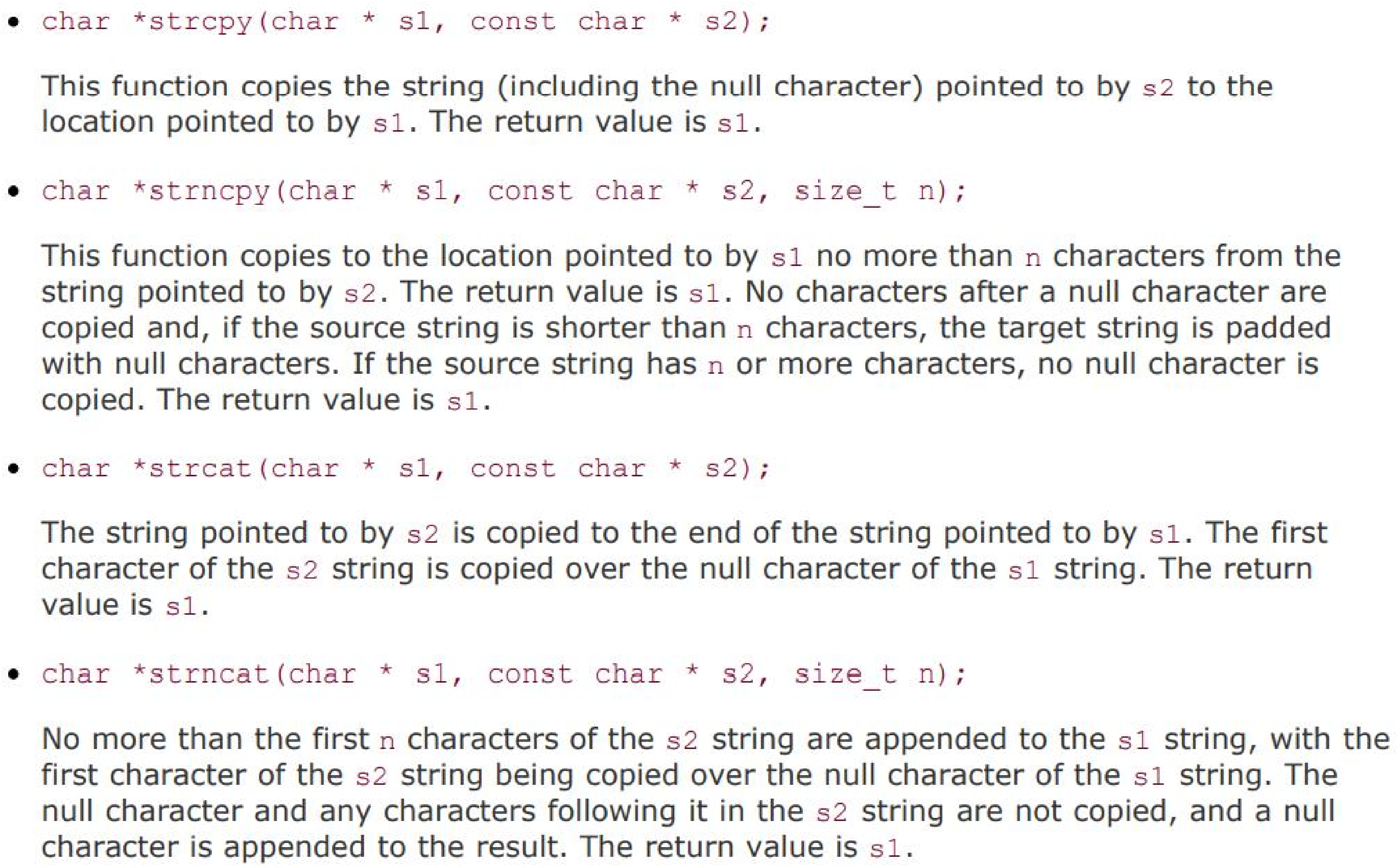
Result:



```
    Sports                  Books
Table tennis           Algorithms
Football          C++ Programming
Swimming          Design Patterns

Address of Sports    Address of Books
0x7fff6303bda0    0x7fff6303bd80
0x7fff6303bdb0    0x7fff6303bd88
0x7fff6303bdc0    0x7fff6303bd90
The size of sports is:48
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`.

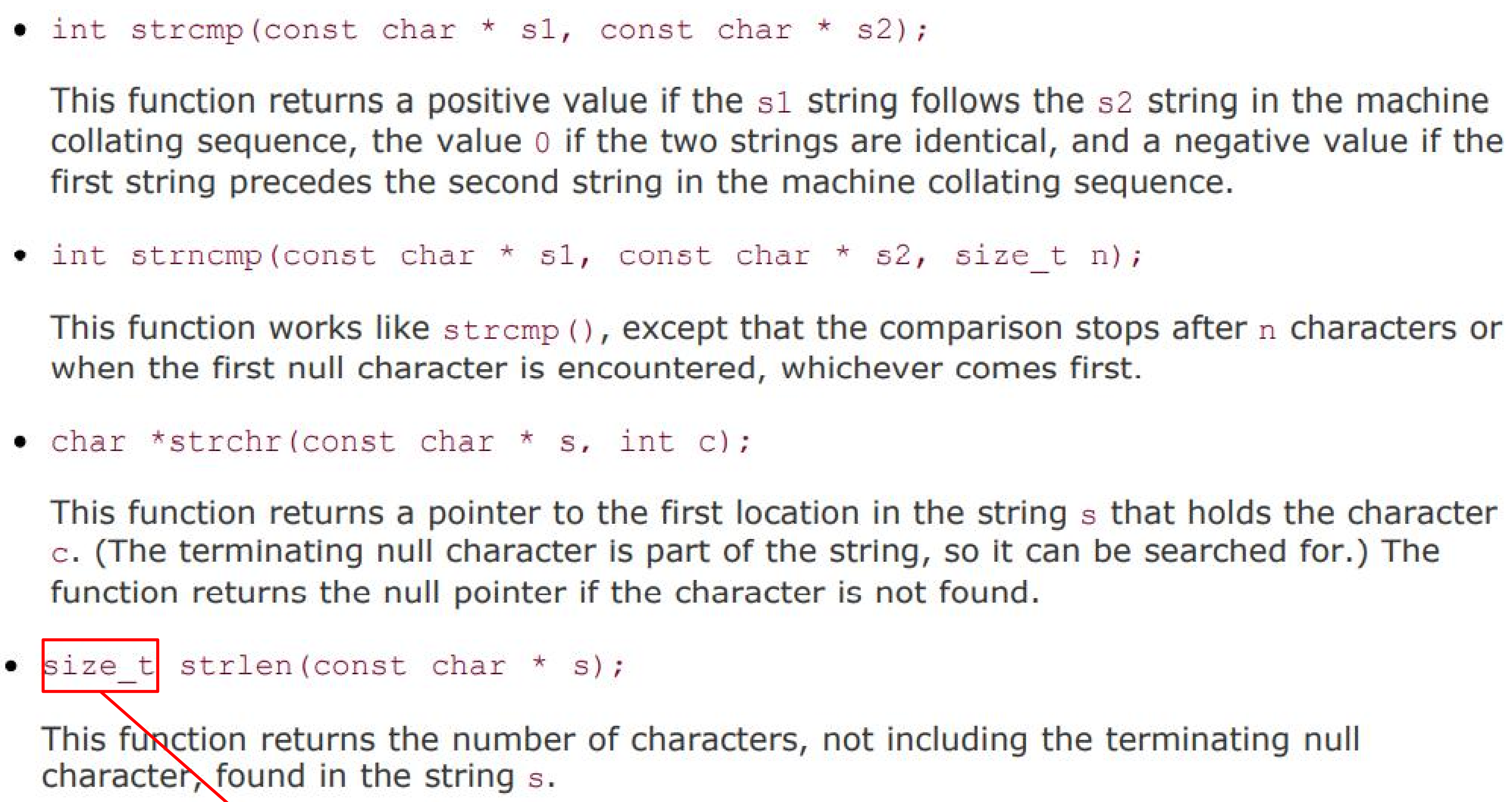
- `char *strcat(char * s1, const char * s2);`

  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`.

https://en.cppreference.com/w/cpp/header/cstring



- `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);`

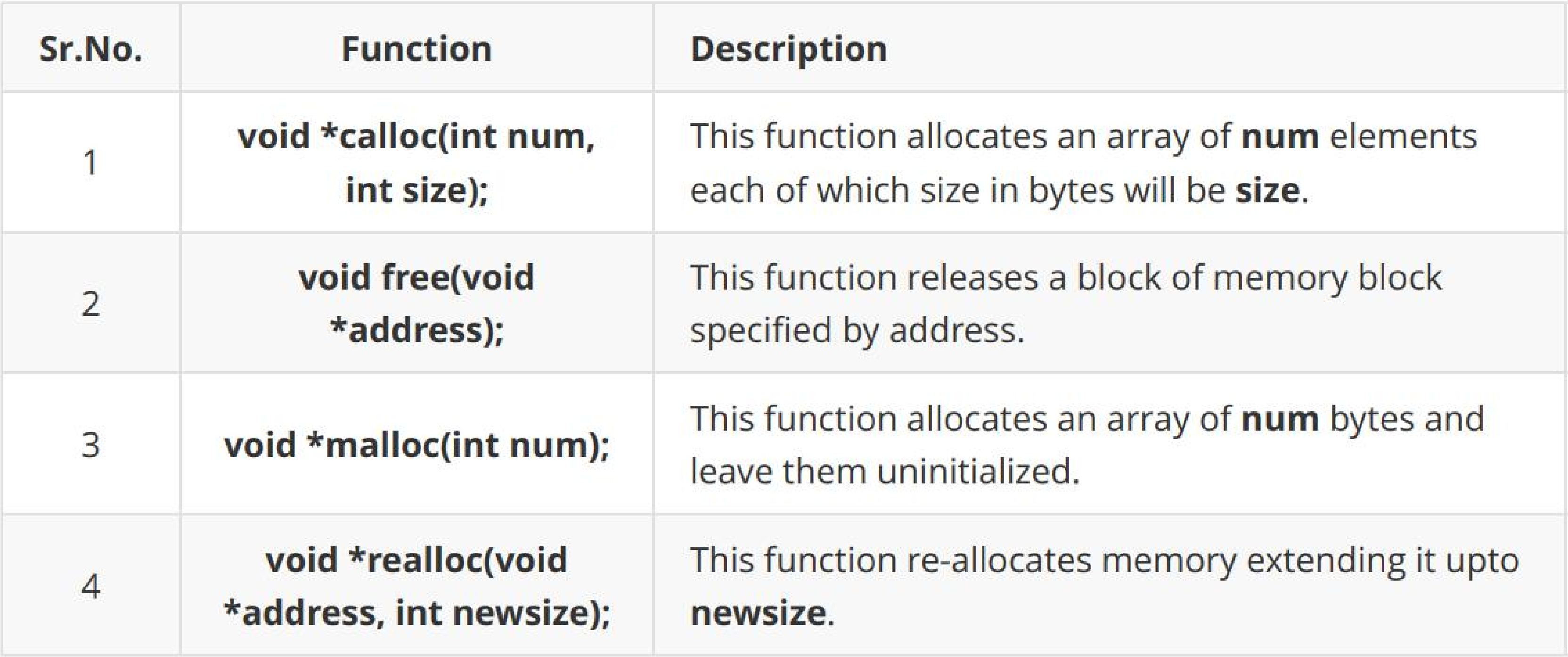
  This function returns the number of characters, not including the terminating null character, found in the string `s`.

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().**

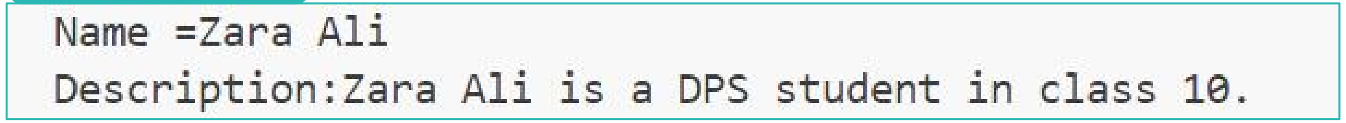
1 Allocating Memory Dynamically

```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;
}
```

Let the pointer point to the memory, you can use calloc(200, sizeof(char)) to replace malloc function.

Release the memory.

Result:



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

2 Resizing Memory

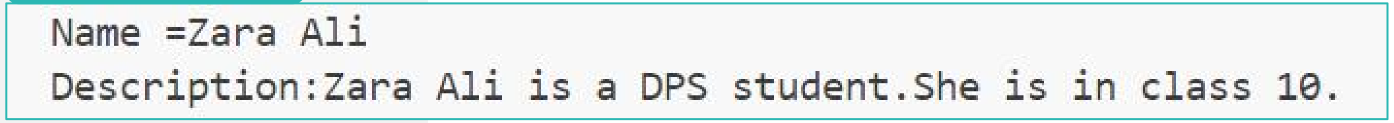
```c
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
int main()
{
    char name[100];
    char *description;
    strcpy(name,"Zara Ali");
    /*allocate memory dynamically */
    description =(char *)malloc(30*sizeof(char));
    if(description ==NULL)
        fprintf(stderr,"Error-unable to allocate required memory.\n");
    else
        strcpy(description,"Zara Ali is a DPS student.");

    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;
}
```

```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.

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

Result:



```
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

```cpp
#include <iostream>
using namespace std;
int main()
{
    //Pointer initial with null
    double *pvalue = NULL;

    //Request memory for the variable
    pvalue = new double;

    //Store value at allocated address
    *pvalue = 1294948.98;
    cout << "Value of pvalue:" << *pvalue << endl;

    //Free up the memory
    delete pvalue;

    return 0;
}
```

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

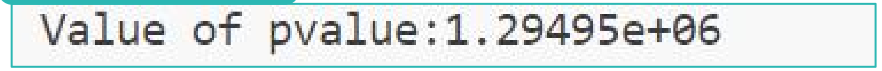
new data-type;

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

delete pointer variable;

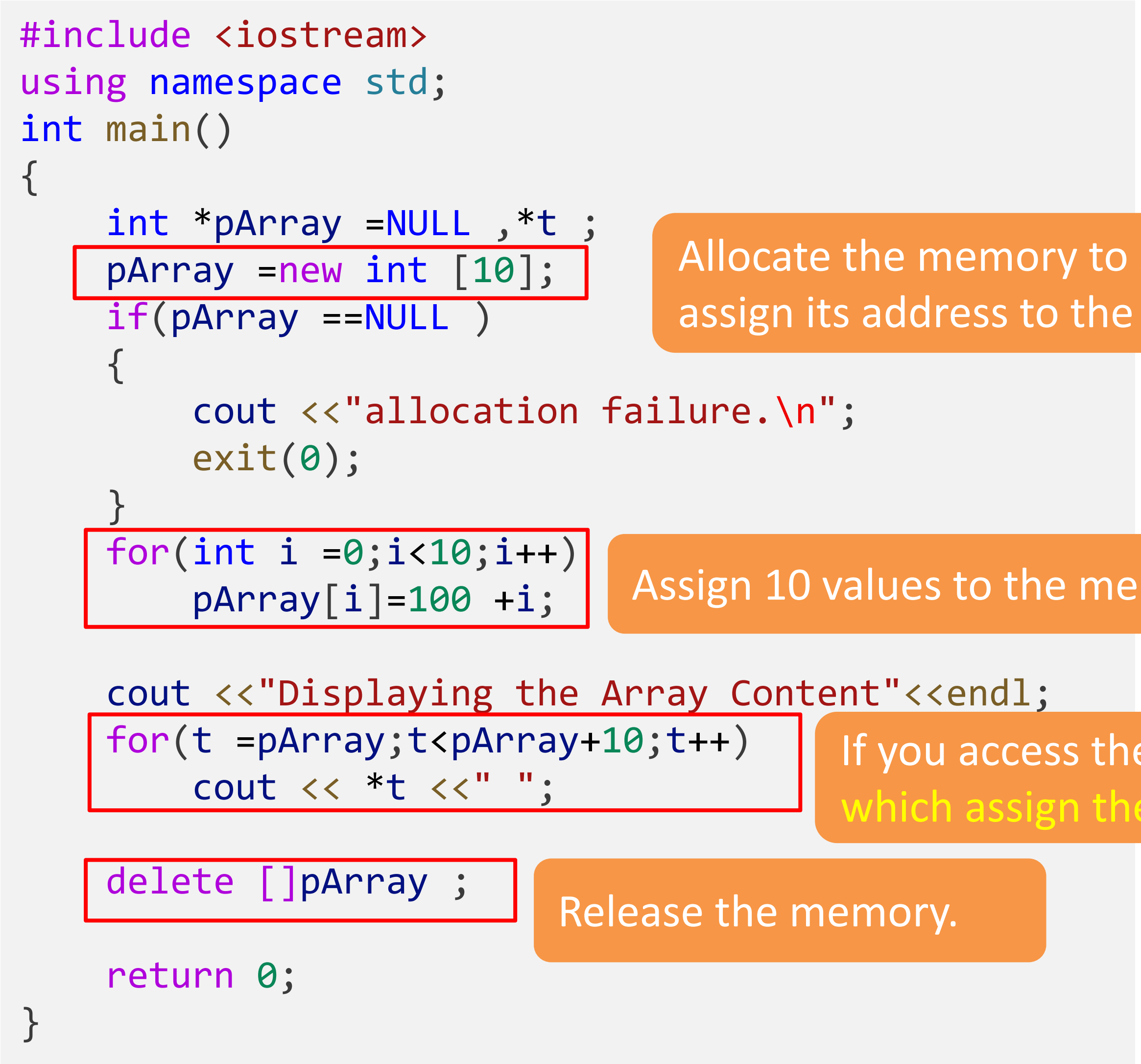
Use delete operator to de-allocate memory that was previously allocated by new operator.

Result:



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

2. Dynamic Memory Allocation for Arrays



```
#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;
}
```

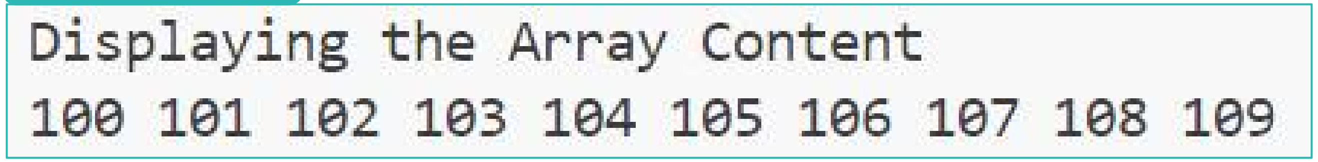
Allocate the memory to store 10 integers, and assign its address to the pointer **pArray**.

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.

Result:

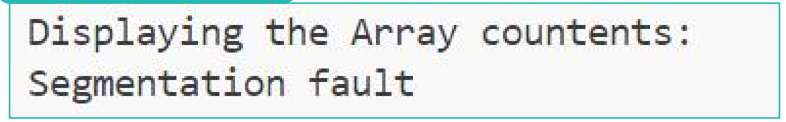


```
Displaying the Array Content
100 101 102 103 104 105 106 107 108 109
```

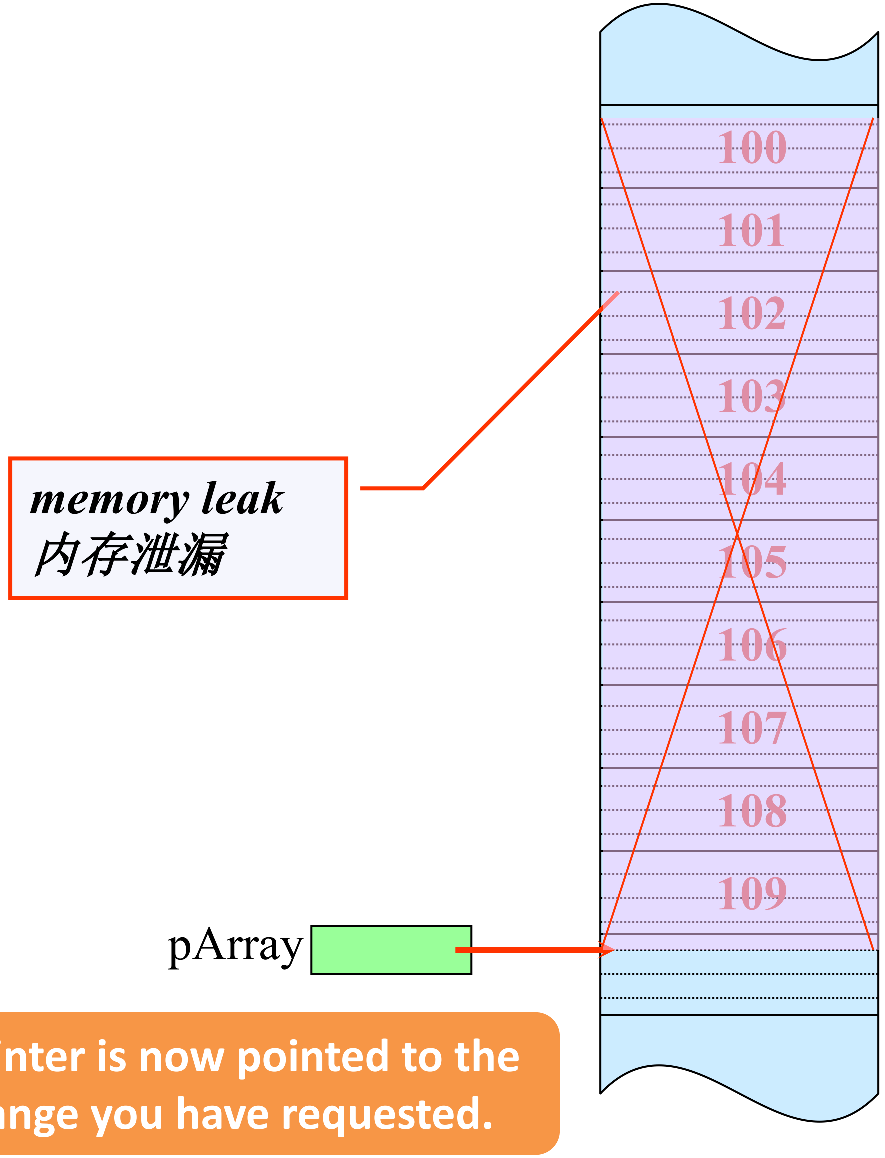
Memory Leak

```
#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 contents:" << endl;
    for(int i=0;i<10;i++,pArray++)
        cout << *pArray << " ";

    delete[] pArray;
    return 0;
}
```

Result:



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



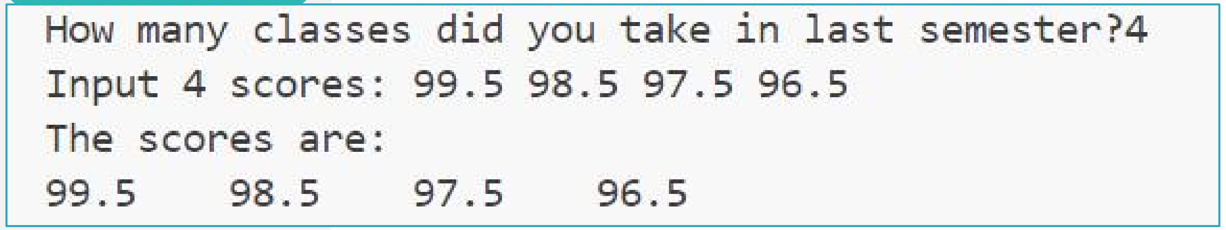
memory leak
内存泄漏

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

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.

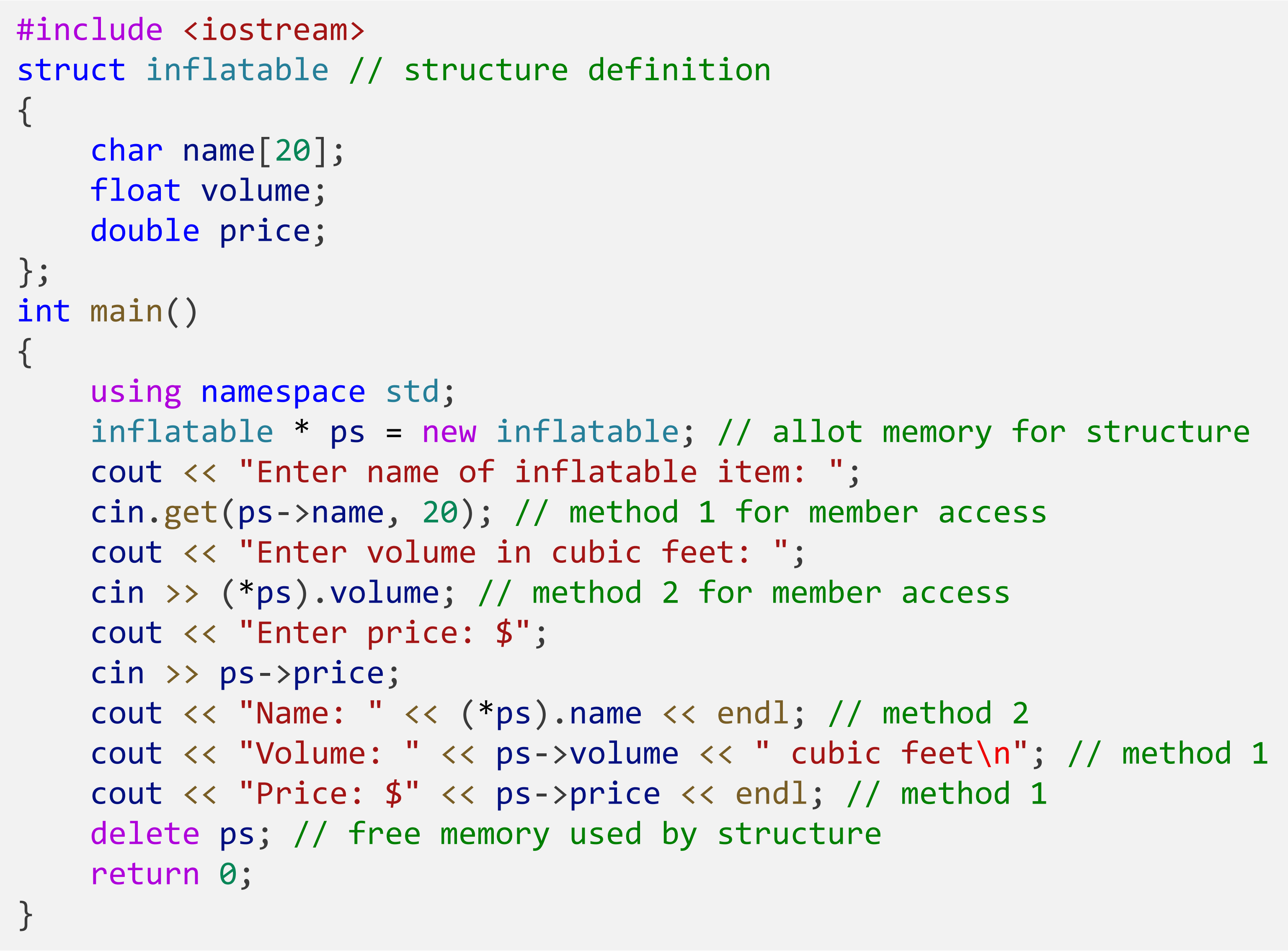
```
#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "How many classes did you take in last semester?";
    cin >> n;

    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;
}
```

Result:

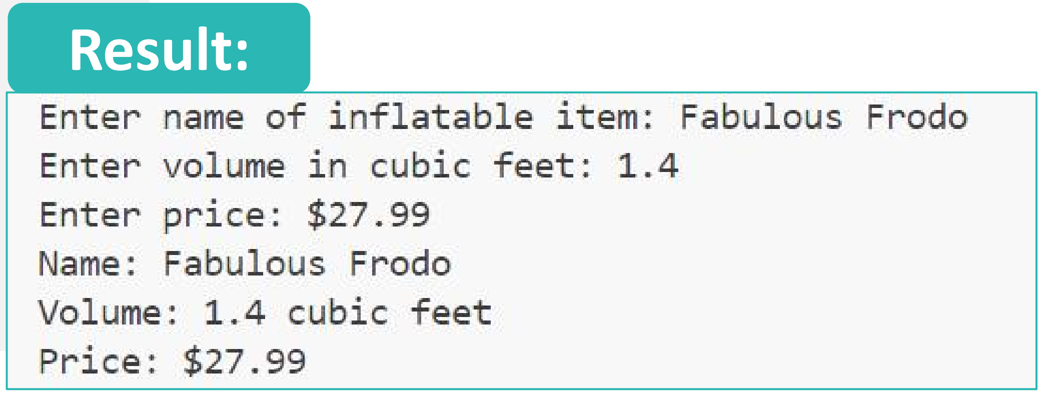


```
How many classes did you take in last semester?4
Input 4 scores: 99.5 98.5 97.5 96.5
The scores are:
99.5    98.5    97.5    96.5
```

3. Dynamic Memory Allocation for Structures



```cpp
#include <iostream>
struct inflatable // structure definition
{
    char name[20];
    float volume;
    double price;
};
int main()
{
    using namespace std;
    inflatable * ps = new inflatable; // allot memory for structure
    cout << "Enter name of inflatable item: ";
    cin.get(ps->name, 20); // method 1 for member access
    cout << "Enter volume in cubic feet: ";
    cin >> (*ps).volume; // method 2 for member access
    cout << "Enter price: $";
    cin >> ps->price;
    cout << "Name: " << (*ps).name << endl; // method 2
    cout << "Volume: " << ps->volume << " cubic feet\n"; // method 1
    cout << "Price: $" << ps->price << endl; // method 1
    delete ps; // free memory used by structure
    return 0;
}
```



Result:

```
Enter name of inflatable item: Fabulous Frodo
Enter volume in cubic feet: 1.4
Enter price: $27.99
Name: Fabulous Frodo
Volume: 1.4 cubic feet
Price: $27.99
```

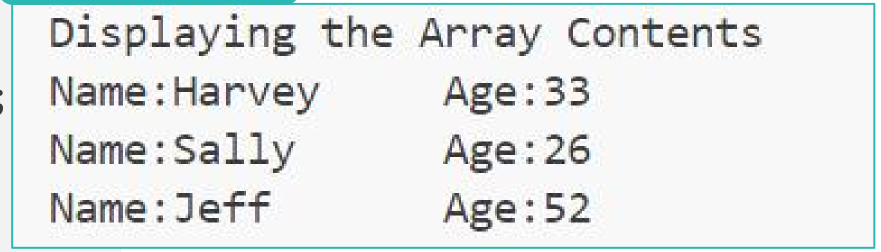
Structured array

```
#include <iostream>
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

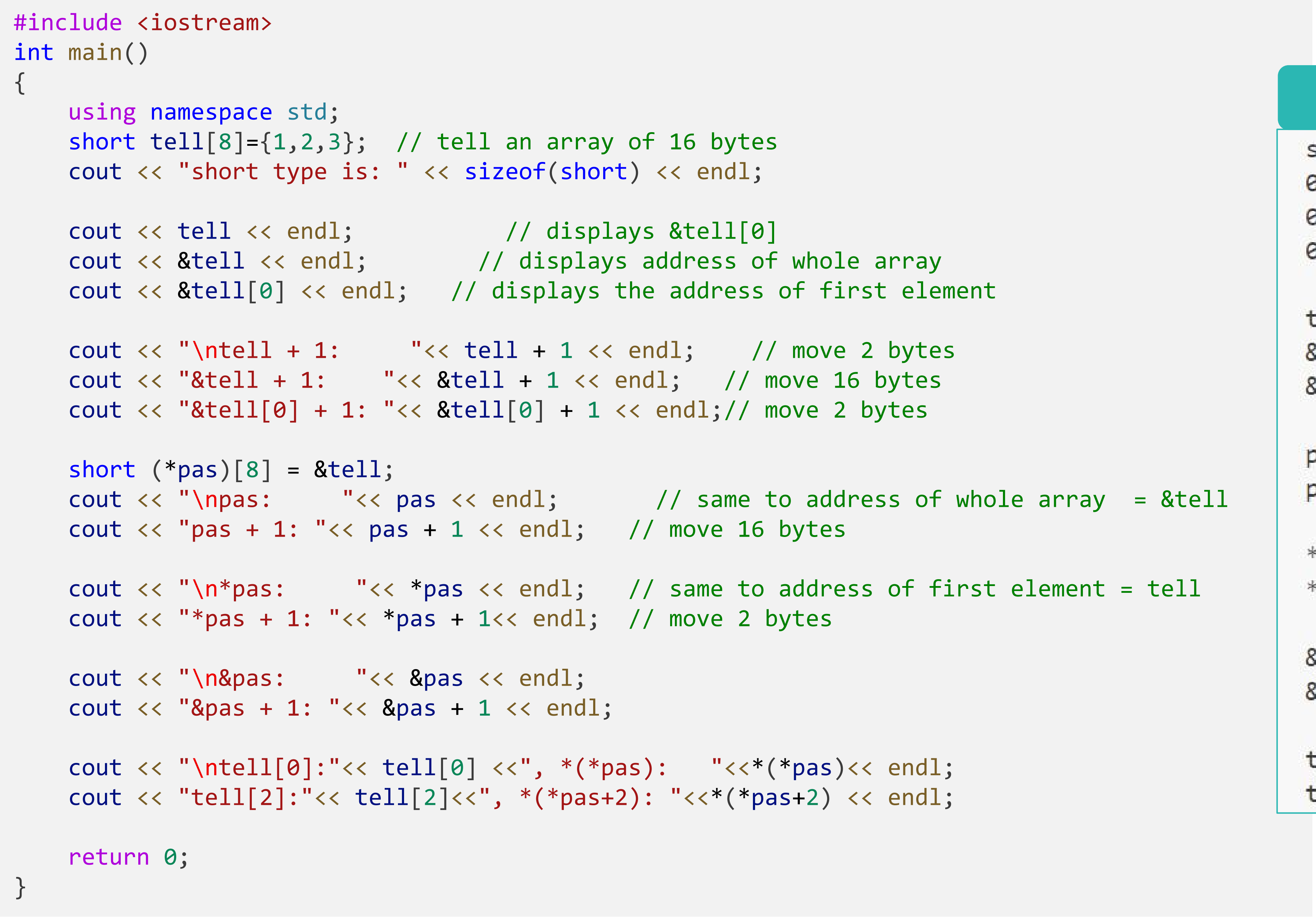
nothrow constant, this constant value is used as an argument for [**operator new**] and [**operator new[]**] to indicate that these functions shall not throw an exception on failure, but return a *null pointer* instead.

Result:



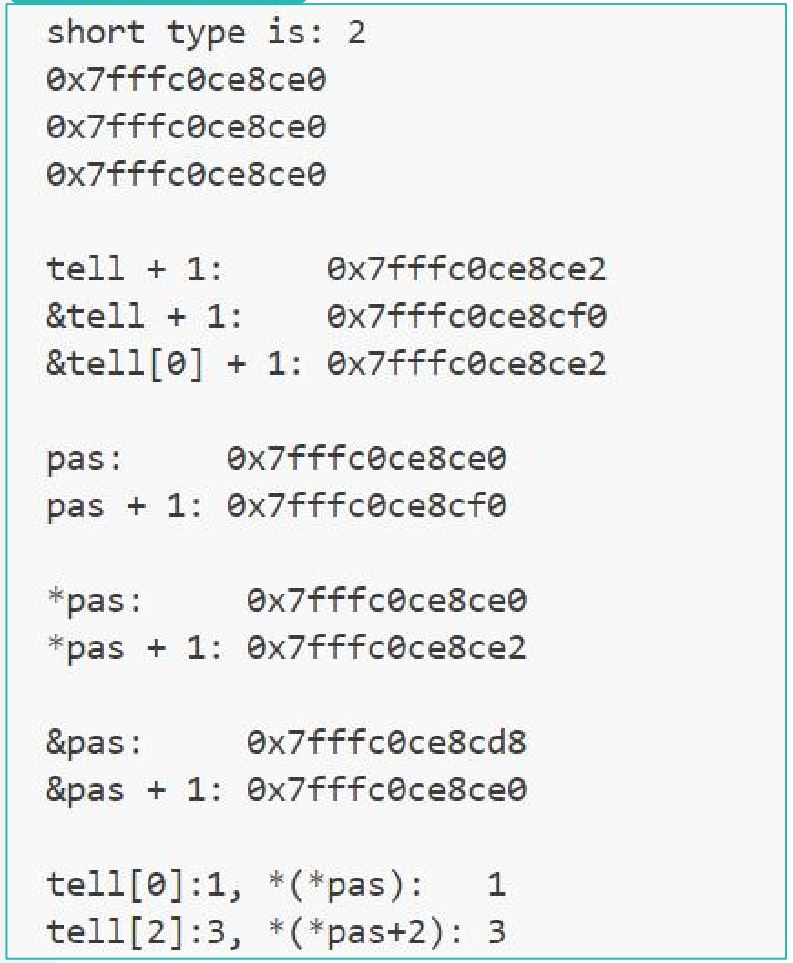
```
Displaying the Array Contents
Name:Harvey     Age:33
Name:Sally      Age:26
Name:Jeff       Age:52
```

The Address of an Array



```cpp
#include <iostream>
int main()
{
    using namespace std;
    short tell[8]={1,2,3};  // tell an array of 16 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 << "\ntell + 1:     "<< tell + 1 << endl;    // move 2 bytes
    cout << "&tell + 1:    "<< &tell + 1 << endl;   // move 16 bytes
    cout << "&tell[0] + 1: "<< &tell[0] + 1 << endl;// move 2 bytes

    short (*pas)[8] = &tell;
    cout << "\npas:     "<< pas << endl;       // same to address of whole array  = &tell
    cout << "pas + 1: "<< pas + 1 << endl;   // move 16 bytes

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

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

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

    return 0;
}
```

Result:



```
short type is: 2
0x7fffc0ce8ce0
0x7fffc0ce8ce0
0x7fffc0ce8ce0

tell + 1:     0x7fffc0ce8ce2
&tell + 1:    0x7fffc0ce8cf0
&tell[0] + 1: 0x7fffc0ce8ce2

pas:     0x7fffc0ce8ce0
pas + 1: 0x7fffc0ce8cf0

*pas:     0x7fffc0ce8ce0
*pas + 1: 0x7fffc0ce8ce2

&pas:     0x7fffc0ce8cd8
&pas + 1: 0x7fffc0ce8ce0

tell[0]:1, *(*pas):   1
tell[2]:3, *(*pas+2): 3
```

Exercise 1

Run the program and explain the result to SA.

```
#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;
}
```

Exercise 2

Run the program and explain the result to SA.

```
#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;
}
```

Exercise 3

Run the program and explain the result to SA.

```c
#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;
}
```

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.