EP578 Computing for Physicists

Topic 5
Arrays, Pointers & Vectors

Department of
Engineering Physics
University of Gaziantep

Course web page
www.gantep.edu.tr/~bingul/ep578

Oct 2011

1. Introduction

This lecture covers the following topics:

- Arrays
- References and Pointers
- Arrays and Pointers
- Arrays and Functions
- Dynamic Memory Management
- C++ Vectors
- Examples

2. Arrays

- A variable represents a *single value*; we call this a *scalar* variable.

```
double x;
```

- An array variable can represent *more than one value*, but still with the *same name and same type*.

```
double x[5];
```

Variable **`x`** can now store 5 values, all of type `double`.

Each *element* of the array of variables is accessed with an index:

`x[i]` with index `i = 0, 1, 2, 3, 4.`

In general for **`n`** elements the index range is `i = 0, 1, ... , n-1`.

Array Decleration

The general form of the declaration of an array is:

```
type name[numberOfElements];
```

Examples

```
double mass[10];
```

The elements are:

```
mass[0]
mass[1]
mass[2]
mass[3]
mass[4]
mass[5]
mass[6]
mass[7]
mass[8]
mass[9]
```

```
int scores[3];
```

The elements are:

```
scores[0]
scores[1]
scores[2]
```

```
char status[2];
```

The elements are:

```
status[0]
status[1]
```

Array Initialisation

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

int main () {

  const int n = 5;

  double a[ ] = {8.4, 3.6, 9.1, 4.7, 3.9};
  int    b[n] = {4, 2};
  double c[n] = {0.0};

  for (int i = 0; i<n; i++)
    cout << a[i] << ", "
         << b[i] << ", "
         << c[i] << endl;

  return 0;
}
```

Output

```
8.4, 4, 0
3.6, 2, 0
9.1, 0, 0
4.7, 0, 0
3.9, 0, 0
```

Array Assignment

Consider again the array a containing 5 elements.

```
    double a[5];
```

At this time, the elements have *unpredictable values!*

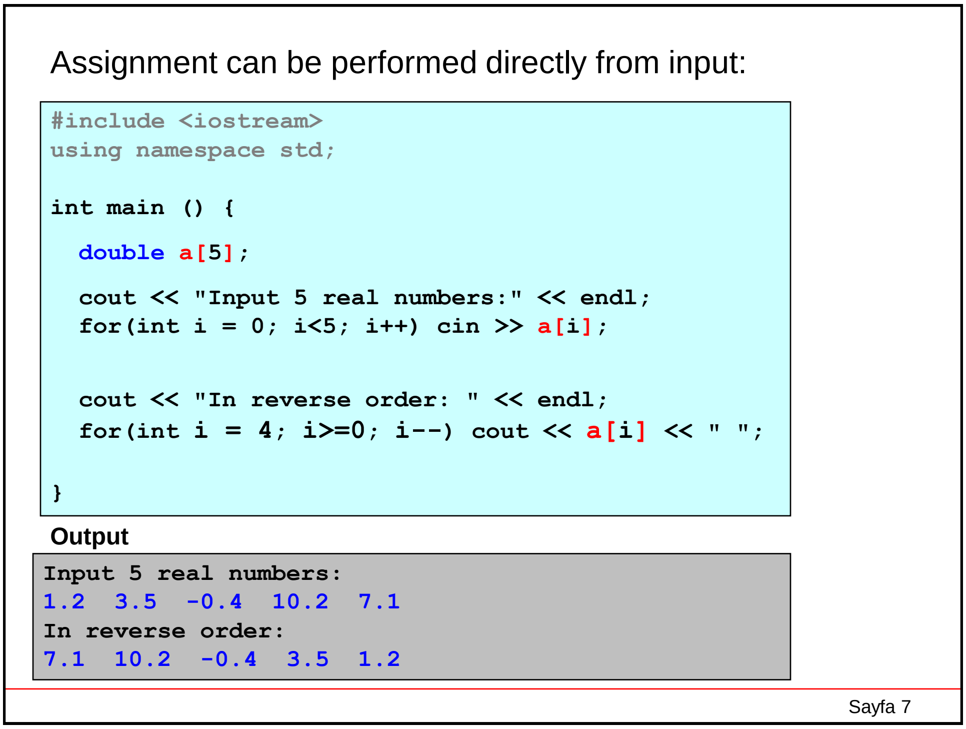
a	?	?	?	?	?
	0	1	2	3	4

Elements of the array can be *assigned* (at any time) as follows:

```
a[0] = 8.4;
a[1] = 3.6;
a[2] = 9.1;
a[4] = 3.9;
```

a	8.4	3.6	9.1	?	3.9
	0	1	2	3	4

Note that element a[3] is still not defined!

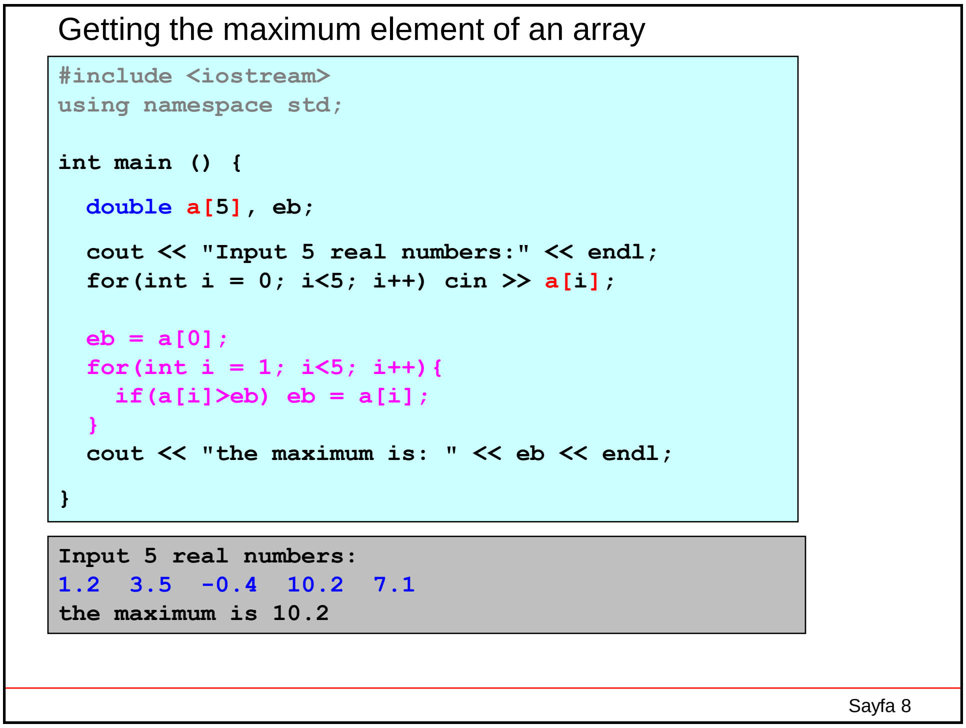


Assignment can be performed directly from input:

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

int main () {

  double a[5];

  cout << "Input 5 real numbers:" << endl;
  for(int i = 0; i<5; i++) cin >> a[i];


  cout << "In reverse order: " << endl;
  for(int i = 4; i>=0; i--) cout << a[i] << " ";

}
```

Output

```
Input 5 real numbers:
1.2  3.5  -0.4  10.2  7.1
In reverse order:
7.1  10.2  -0.4  3.5  1.2
```



Getting the maximum element of an array

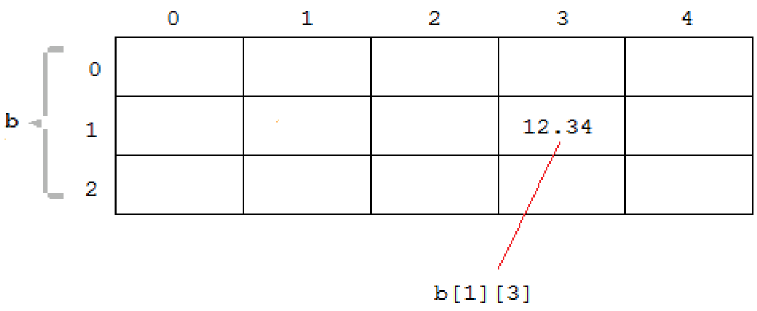
```
#include <iostream>
using namespace std;

int main () {

  double a[5], eb;

  cout << "Input 5 real numbers:" << endl;
  for(int i = 0; i<5; i++) cin >> a[i];

  eb = a[0];
  for(int i = 1; i<5; i++){
    if(a[i]>eb) eb = a[i];
  }
  cout << "the maximum is: " << eb << endl;

}
```

```
Input 5 real numbers:
1.2  3.5  -0.4  10.2  7.1
the maximum is 10.2
```

Multidimensional Arrays

```
double a[5];      // 5-element one-dimensional array
float b[3][5];    // 15-element two-dimensional array
int c[5][4][10]; // 200-element three-dimensional ar.
```

```
b[1][3] = 12.34;
```



Passing Arrays to Functions

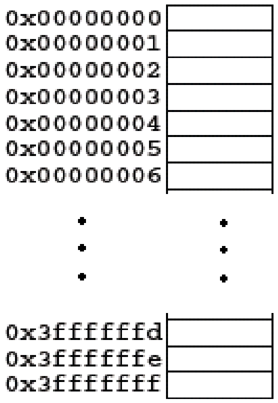
```
#include <iostream>
using namespace std;
// returns the sum of first n elements
double sum(double x[], int n) {
  double t = 0.0;
  for(int i=0; i<n; i++){
     t = t + x[i];
  }
  return t;
}

int main () {
  double a[5], s;
  cout << "Enter 5 reals: ";
  for (int k=0; k<5; k++) cin >> a[k];

  s = sum(a, 5);
  cout << "sum of the elements is " << s << endl;
}
```

```
Enter 5 reals: 1.1 2.2 3.3 4.4 5.5
sum of the elements is 16.5
```

3. Variables and Memory Addresses

Computer memory can be considered as a very large array of bytes.

For example, a computer with
1 GB of RAM actually contains
an array of
1024 x 1024 x 1024 = 1,073,741,824 B.



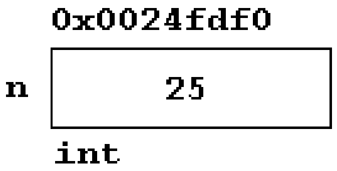
0 = 0x00000000
1,073,741,824 = 0x3fffffff

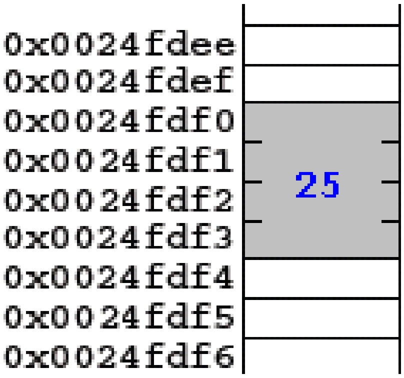
When a variable is declared and assigned to a value four fundamental attributes associated with it:

- its name
- its type
- its value (content)
- its address

e.g.

```
int n = 25;
```





In C/C++ the address operator (**&**) returns the memory address of a variable.

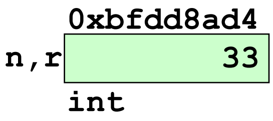
```
int main(){
   int n = 33;
   cout << " n = " <<  n << endl;
   cout << "&n = " << &n << endl;
}
```

```
  n = 33

&n = 0x0024fdf0
```

4. References

- The **reference** is an ***alias,*** a ***synonym*** for a variable.
- It is decelerated by using the ***reference operator*** **&**.

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

int main(){
  int n = 33;
  int &r = n; // r is a reference for n

  cout << n  << " " << r << endl;

  --n;
  cout << n  << " " << r << endl;

  r *= 2;
  cout << n  << " " << r << endl;

  cout << &n << " " << &r << endl;
  return 0;
}
```



```
33 33
32 32
64 64
0xbfdd8ad4 0xbfdd8ad4
```

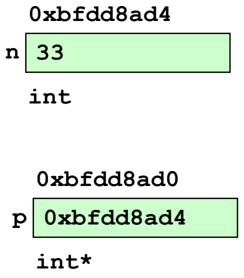
```
#include <iostream>
using namespace std;

void takas(double &x, double &y){
  double z;
  z = x;
  x = y;
  y = z;
}

int main(){
    double a = 11.1,  b = 22.2;

    cout << "a b : " << a << " " << b << endl;

    takas(a,b);

    cout << "a b : " << a << " " << b << endl;
}
```

```
a b: 11.1  22.2
a b: 22.2  11.1
```

5. Pointers

- The address operator returns the memory adress of a variable.
- We can store the address in another variable, called ***pointer***.

```
#include <iostream>
using namespace std;
int main()
{
  int n = 33;
  int* p = &n; // p holds the address of n
  cout << " n = " <<  n << endl;
  cout << "&n = " << &n << endl;
  cout << " p = " <<  p << endl;
  cout << "&p = " << &p << endl;
  cout << "*p = " << *p << endl;
}
```



```
 n = 33
&n = 0xbfdd8ad4
 p = 0xbfdd8ad4
&p = 0xbffafad0
*p = 33
```

Pointers and Arrays

- The name of an array is the address of its first element.
- The array name is a constant pointer.

```
float numbers[20];
float *ptr = &numbers[0];   // valid
```

The following assignments are equivalent:

```
numbers[4] = 25.8;
*(ptr+4) = 25.8;
```

6. Dynamic Memory Management

The declaration:

`double mass[10];` Array size define at ***compile-time***

Alternatively we can use a *named constant*;

```
const int n = 10;
double mass[n];
```

Array size define at ***compile-time***

Note that *"Standard* C++" Array size defined at ***run-time FORBIDDEN!***

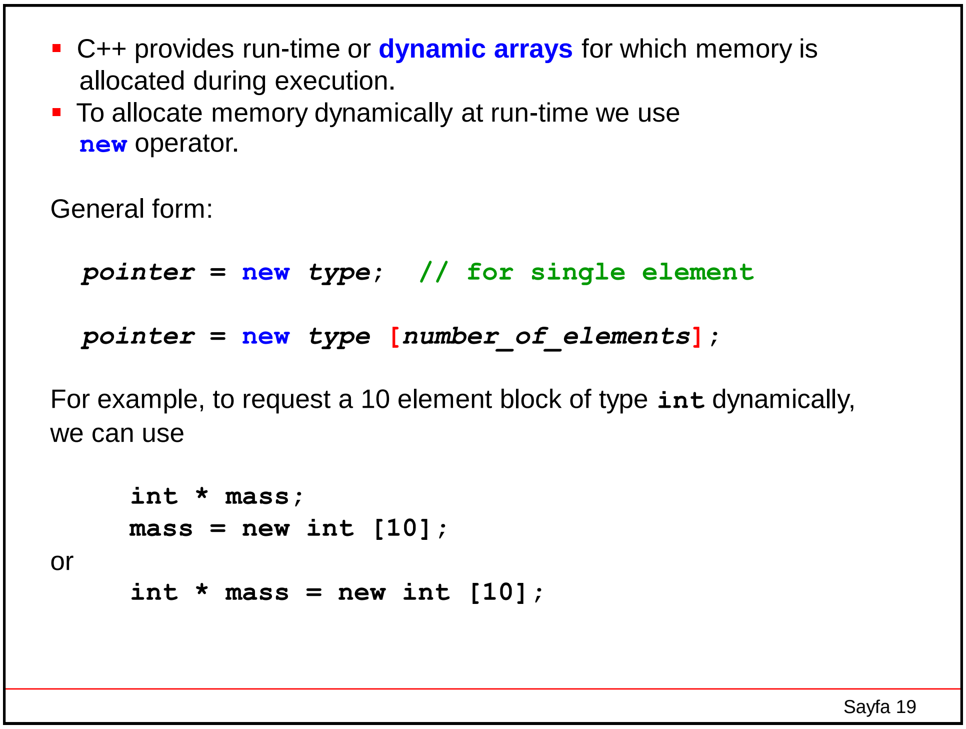
~~`int n;`~~
~~`cin >> n;`~~
~~`double mass[n];`~~

or

~~`int n = 10;`~~
~~`double mass[n];`~~

* * * This type of arrays are called **Static Arrays * * ***

Your compiler might allow you to do this, but it is best to use only standard C++ features so that your program can be compiled on any platform that has a standard C++ compiler.



- C++ provides run-time or **dynamic arrays** for which memory is allocated during execution.
- To allocate memory dynamically at run-time we use `new` operator.

General form:

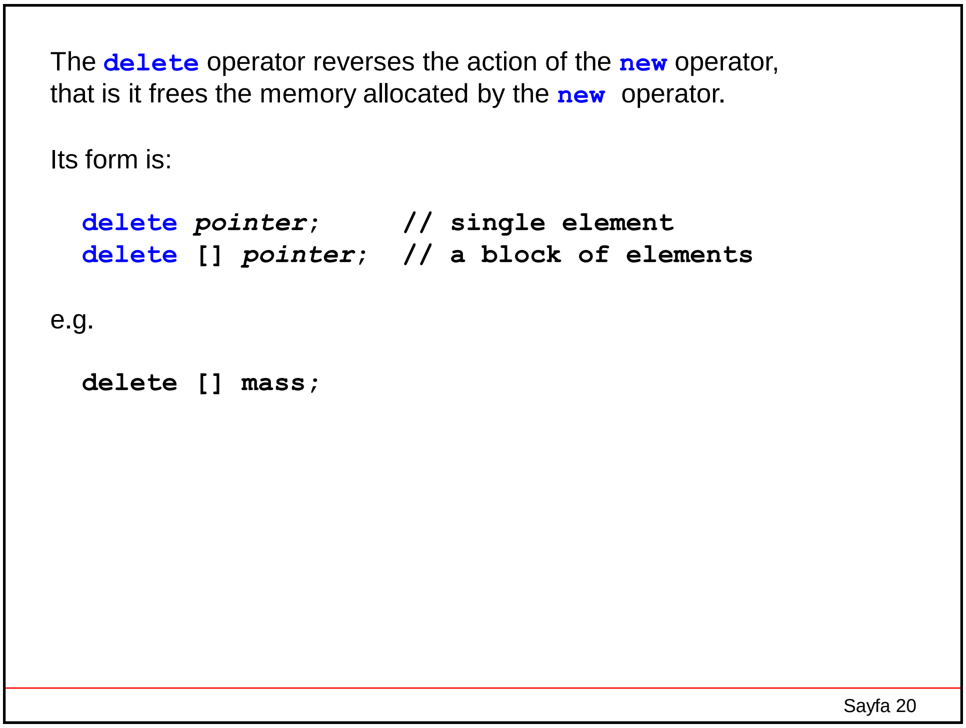
```
pointer = new type;   // for single element

pointer = new type [number_of_elements];
```

For example, to request a 10 element block of type `int` dynamically, we can use

```
int * mass;
mass = new int [10];
```

or

```
int * mass = new int [10];
```



The `delete` operator reverses the action of the `new` operator, that is it frees the memory allocated by the `new` operator.

Its form is:

```
delete pointer;      // single element
delete [] pointer;   // a block of elements
```

e.g.

```
delete [] mass;
```

```
int main (){
    double *x, mean, s;
    int i, n;

    while(true){
        cout << "How many elements: ";  cin >> n;
        if(n<=0) break;

        x = new double[n];
        s = 0.0;
        cout << "Input elements: ";
        for(i = 0; i<n; i++){
            cin >> x[i];
            s += x[i];
        }

        mean = s/n;
        cout << "Mean = " << mean << endl;
        delete [] x;
    }
} // main
```

Sample output of the previous program:

```
How many elements: 3
Input elements: 1 2 3
Mean = 2.0
How many elements: 6
Input elements: 2 4 5 9 1 0
Mean = 3.5
How many elements: 0
```

7. C++ Vectors

*** Static Arrays (SA):**

- the size of SA cannot be defined at run-time
- the size of SA cannot be changed at run-time

*** Dynamic Arrays (DA):**

- the size of DA can be defined at run-time
- the size of DA may change at run-time

*** Vectors:**

C++ provides the *vector* data class that enables the programmer to create *dynamic* arrays:

- the size of a vector can be defined at run-time
- the size of a vector may change at run-time

The vector data class provides many powerful methods for processing dynamic memory management.

Vector Declaration and Initialisation

First, to use the vector class the following header must be included:

```
#include <vector>
```

The general form of the declaration of a vector array is:

```
vector<type> name(numberOfElements);
```

Examples

```
vector<double>  mass(6);
```

The elements are:

```
mass[0]
mass[1]
mass[2]
mass[3]
mass[4]
mass[5]
```

```
vector<int> scores;
```

This is an *empty* vector! The size is zero and so there are no elements.

Note that the indexing of the elements of vectors is the same as that of arrays.

Vector Initialisation

The general form of vector declaration:

```
vector<type> name(numberOfElements);
```

initialises all elements of the vector to zero.

Alternatively an initialiser can be given at declaration:

```
vector<type> name(numberOfElements, value);
```

initialises all elements of the vector to `value`.

Examples

```
vector<double>  mass(6);
```

all elements of `mass` are initialised to `0.0`

```
vector<double>  mass(6, 1.8);
```

all elements of `mass` are initialised to `1.8`

Vector Assignment

Consider the vector declaration:

```
vector<double> a(5);
```

At this time, the elements are all automatically initialised to zero.

a	0.0	0.0	0.0	0.0	0.0
	0	1	2	3	4

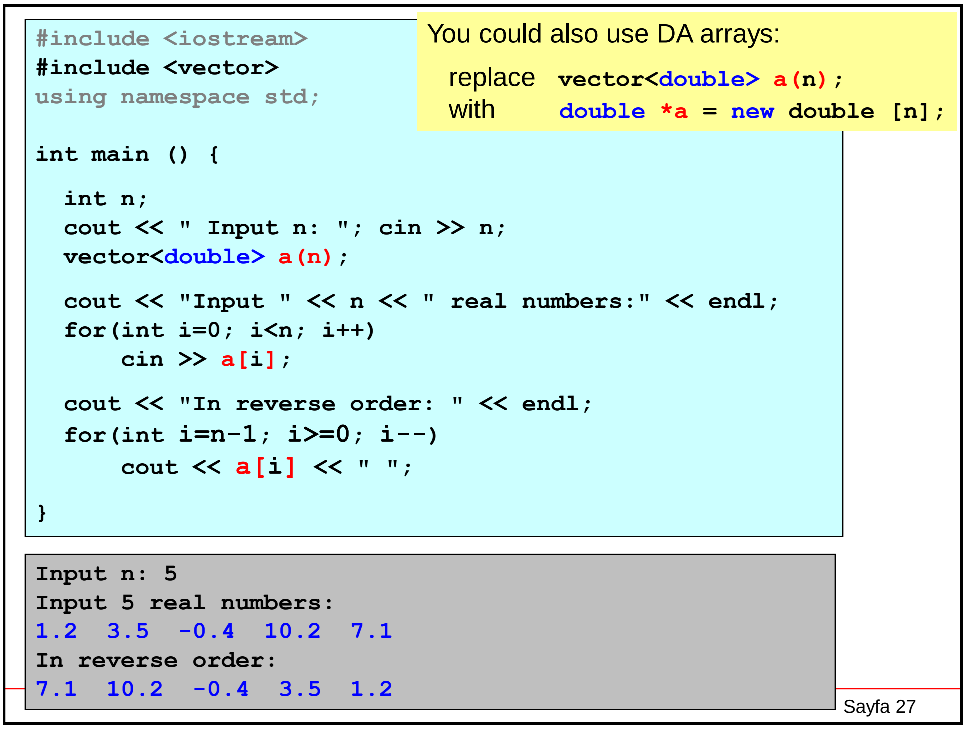
Elements of a vector array can be *assigned* (at any time) as follows:

```
a[0] = 8.4;
a[1] = 3.6;
a[2] = 9.1;
a[4] = 3.9;
```

Note that vector assignment is performed in the same way as array assignment.

a	8.4	3.6	9.1	0.0	3.9
	0	1	2	3	4

Note that the value of element `a[3]` is still 0.0

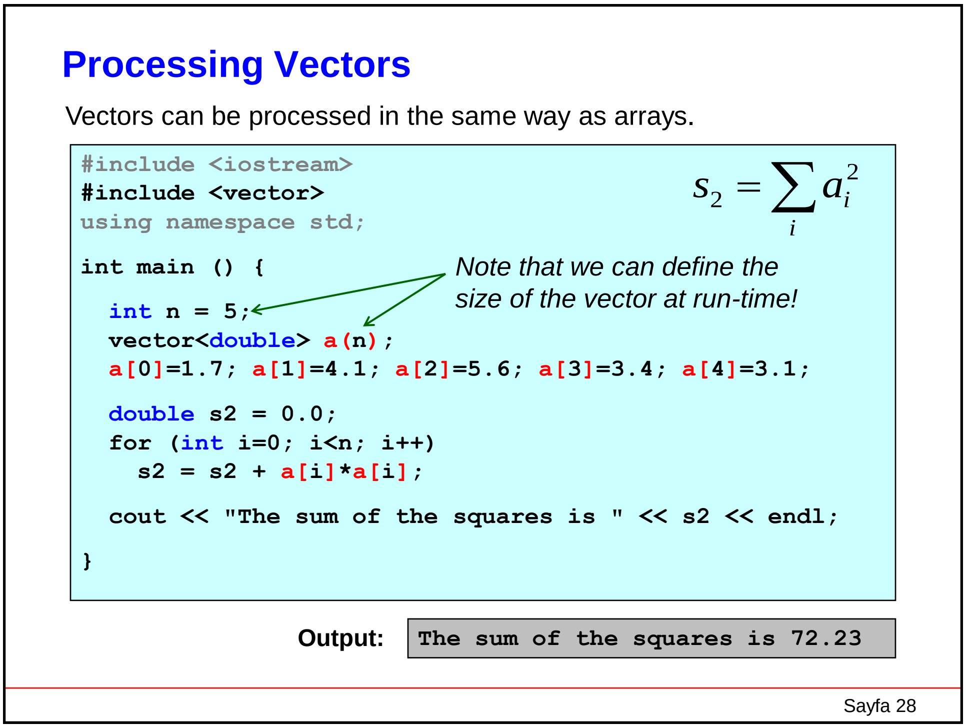


```
#include <iostream>
#include <vector>
using namespace std;

int main () {

  int n;
  cout << " Input n: "; cin >> n;
  vector<double> a(n);

  cout << "Input " << n << " real numbers:" << endl;
  for(int i=0; i<n; i++)
      cin >> a[i];

  cout << "In reverse order: " << endl;
  for(int i=n-1; i>=0; i--)
      cout << a[i] << " ";

}
```

You could also use DA arrays:

replace `vector<double> a(n);`
with `double *a = new double [n];`

```
Input n: 5
Input 5 real numbers:
1.2  3.5  -0.4  10.2  7.1
In reverse order:
7.1  10.2  -0.4  3.5  1.2
```



Processing Vectors

Vectors can be processed in the same way as arrays.

$$s_2 = \sum_i a_i^2$$

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

int main () {

  int n = 5;
  vector<double> a(n);
  a[0]=1.7; a[1]=4.1; a[2]=5.6; a[3]=3.4; a[4]=3.1;

  double s2 = 0.0;
  for (int i=0; i<n; i++)
    s2 = s2 + a[i]*a[i];

  cout << "The sum of the squares is " << s2 << endl;

}
```

Note that we can define the size of the vector at run-time!

Output:

```
The sum of the squares is 72.23
```

Dynamic Processing of Vectors

There are many powerful methods available for dynamic processing of vectors; we will look at just five of them:

`name.size();`	returns the size of vector *name*
`name.push_back(x);`	adds value ***x*** to the end of the vector (increasing the size by one)
`name.pop_back();`	removes a value from the end of the vector (decreasing the size by one)
`name.clear();`	removes all values from the vector (leaving a vector of size zero)
`name.resize(s);`	resizes the vector to size ***s***

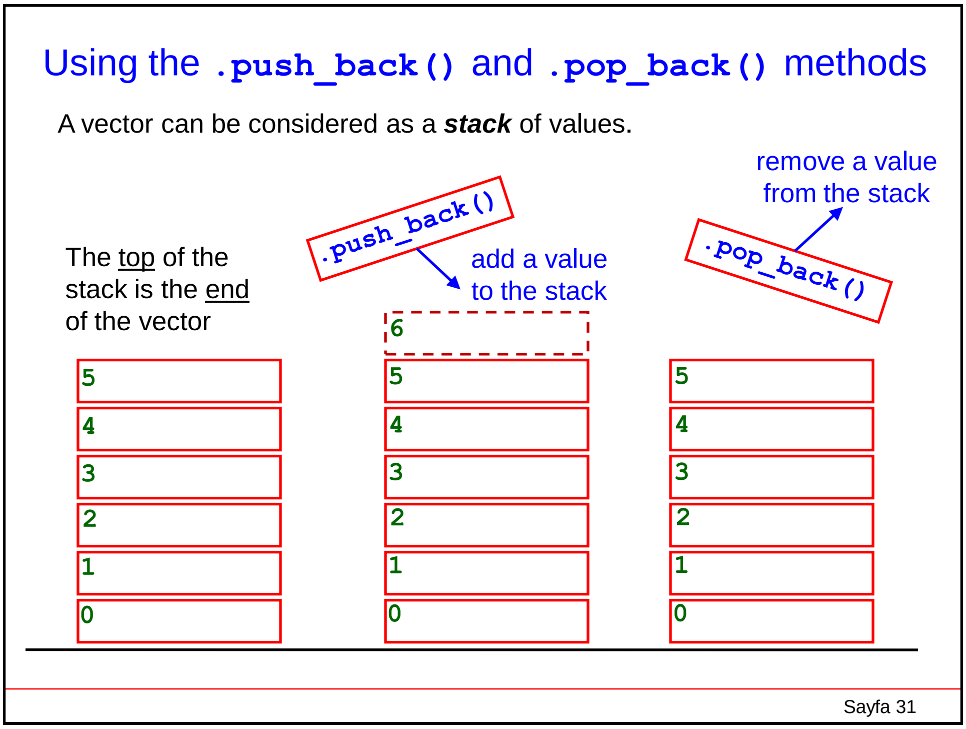
Using the `.size()` method

The `.size()` method provides a simple and consistent way to loop over all elements in a vector without the need to keep track of the vector's size:

```
vector<double> mass(5);
for (unsigned int i=0; i<mass.size(); i++) {
  mass[i] = i*i;
}
```

Note that the `.size()` method returns an **`unsigned int`** and so the counter **`i`** is also defined as type **`unsigned int`**.

In time, you will discover more uses for this method...



Using the `.push_back()` and `.pop_back()` methods

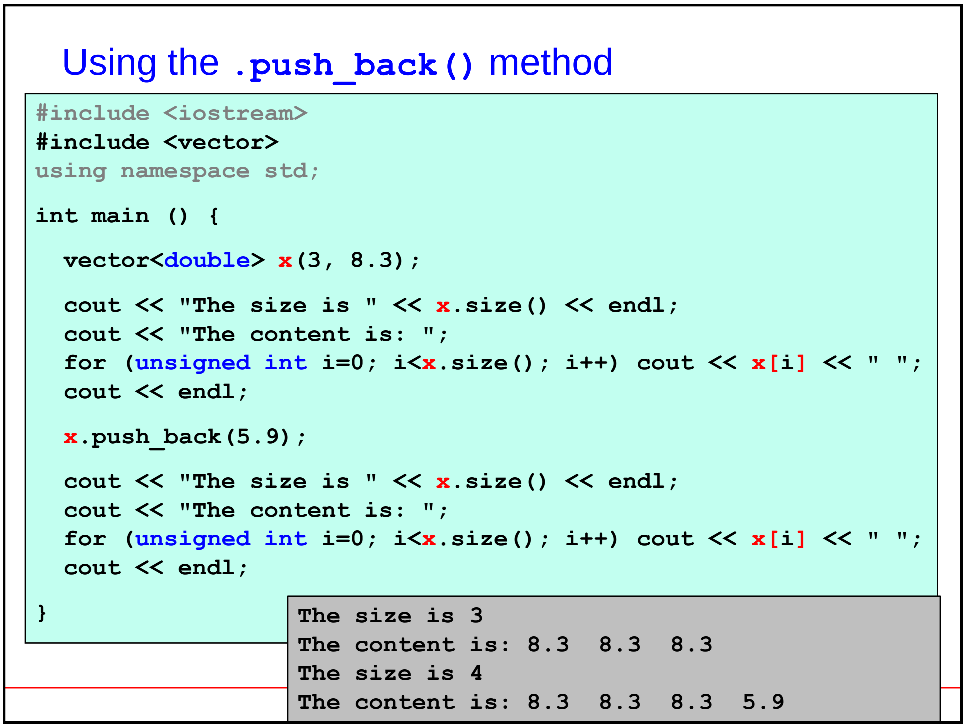
A vector can be considered as a ***stack*** of values.

The top of the stack is the end of the vector

`.push_back()` add a value to the stack

`.pop_back()` remove a value from the stack

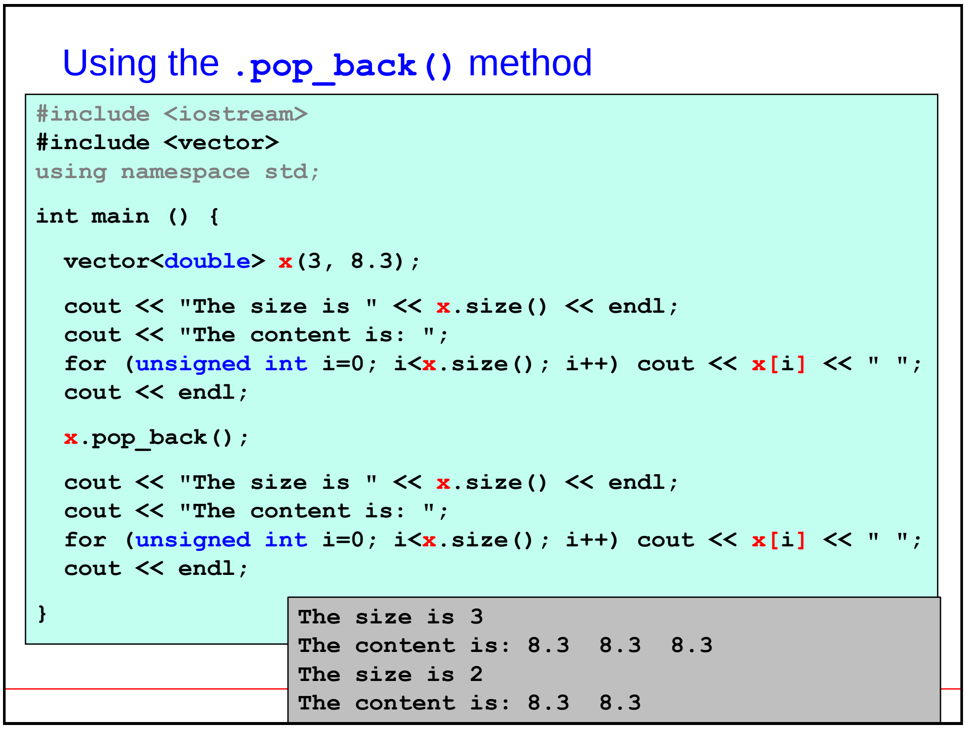
Sayfa 31



Using the `.push_back()` method

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

int main () {

  vector<double> x(3, 8.3);

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

  x.push_back(5.9);

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

}
```

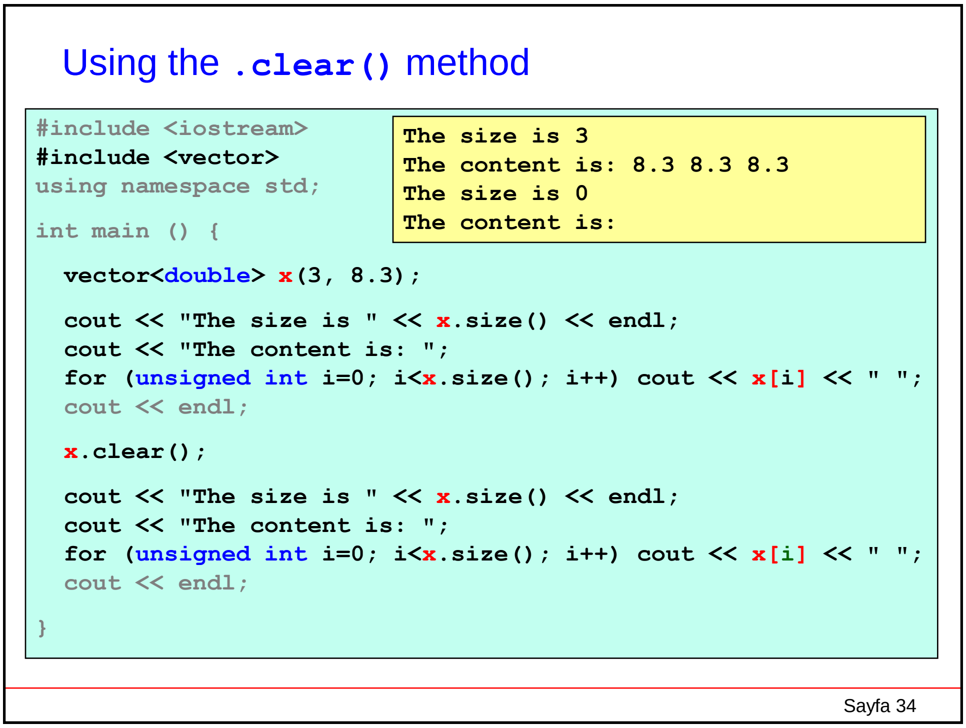
```
The size is 3
The content is: 8.3  8.3  8.3
The size is 4
The content is: 8.3  8.3  8.3  5.9
```



Using the `.pop_back()` method

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

int main () {

  vector<double> x(3, 8.3);

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

  x.pop_back();

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

}
```

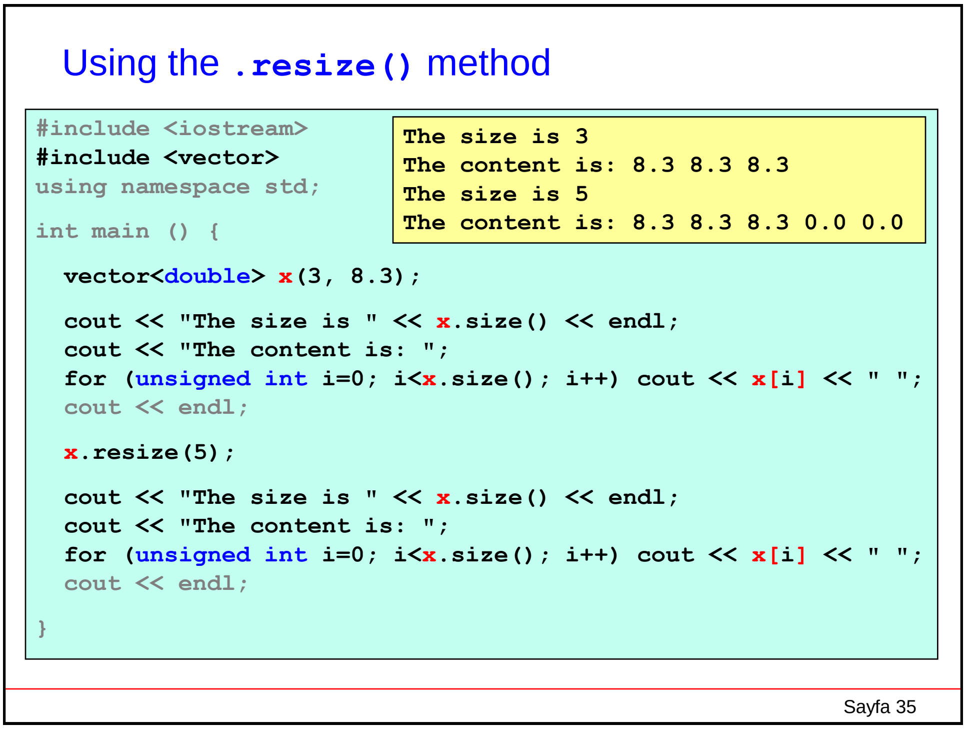
```
The size is 3
The content is: 8.3  8.3  8.3
The size is 2
The content is: 8.3  8.3
```



Using the `.clear()` method

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

int main () {

  vector<double> x(3, 8.3);

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

  x.clear();

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

}
```

```
The size is 3
The content is: 8.3 8.3 8.3
The size is 0
The content is:
```

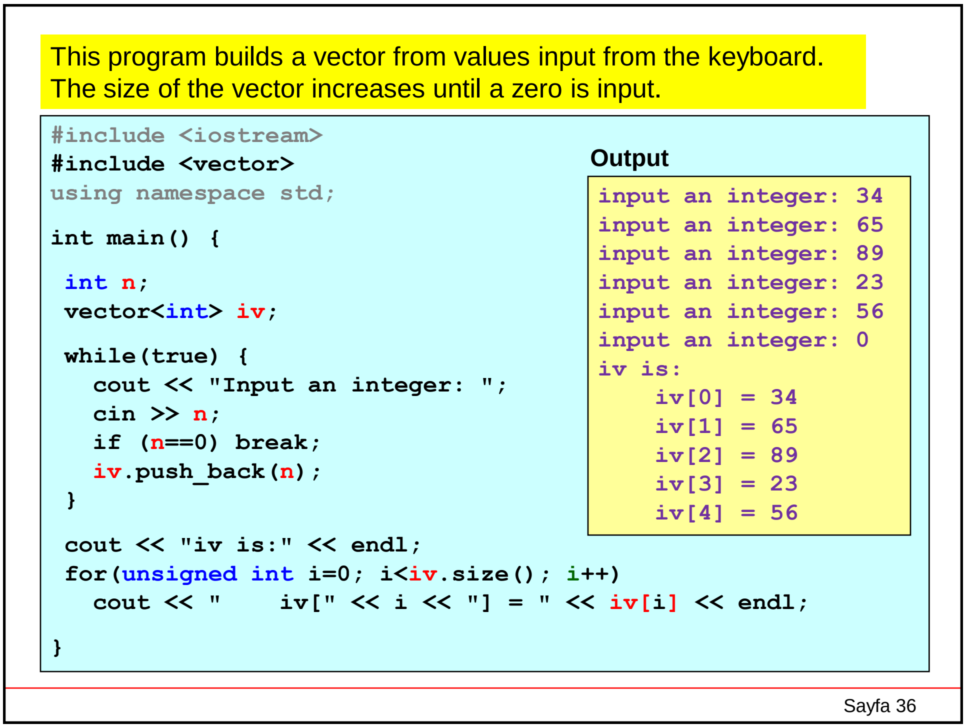


Using the `.resize()` method

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

int main () {

  vector<double> x(3, 8.3);

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

  x.resize(5);

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

}
```

```
The size is 3
The content is: 8.3 8.3 8.3
The size is 5
The content is: 8.3 8.3 8.3 0.0 0.0
```

Sayfa 35



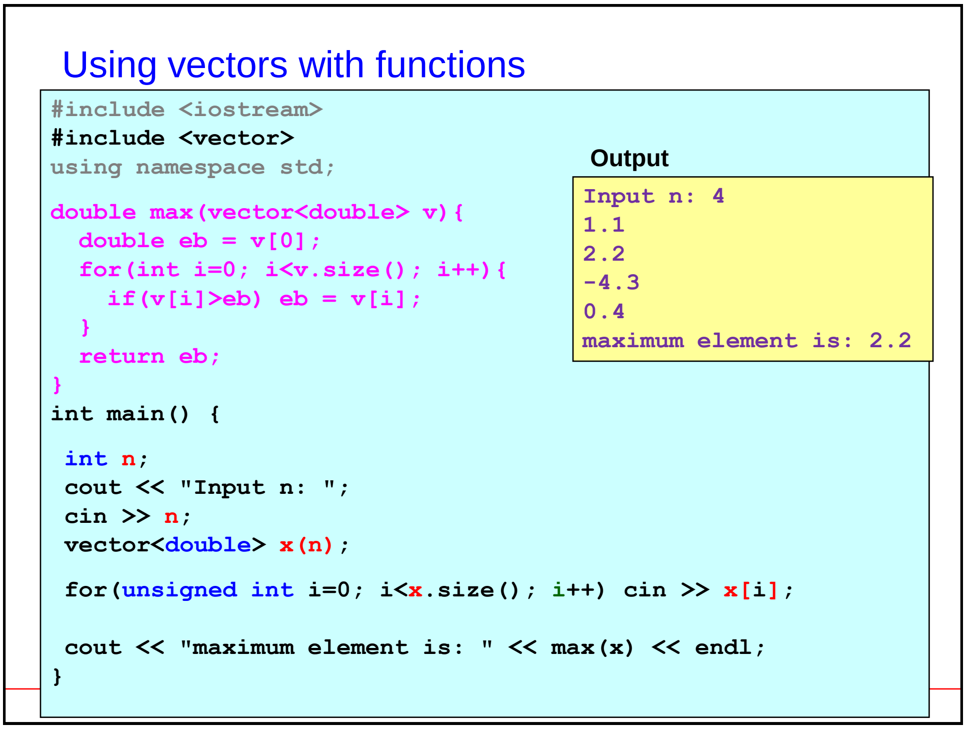
This program builds a vector from values input from the keyboard.
The size of the vector increases until a zero is input.

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

int main() {

 int n;
 vector<int> iv;

 while(true) {
   cout << "Input an integer: ";
   cin >> n;
   if (n==0) break;
   iv.push_back(n);
 }

 cout << "iv is:" << endl;
 for(unsigned int i=0; i<iv.size(); i++)
   cout << "    iv[" << i << "] = " << iv[i] << endl;

}
```

Output

```
input an integer: 34
input an integer: 65
input an integer: 89
input an integer: 23
input an integer: 56
input an integer: 0
iv is:
    iv[0] = 34
    iv[1] = 65
    iv[2] = 89
    iv[3] = 23
    iv[4] = 56
```

Sayfa 36

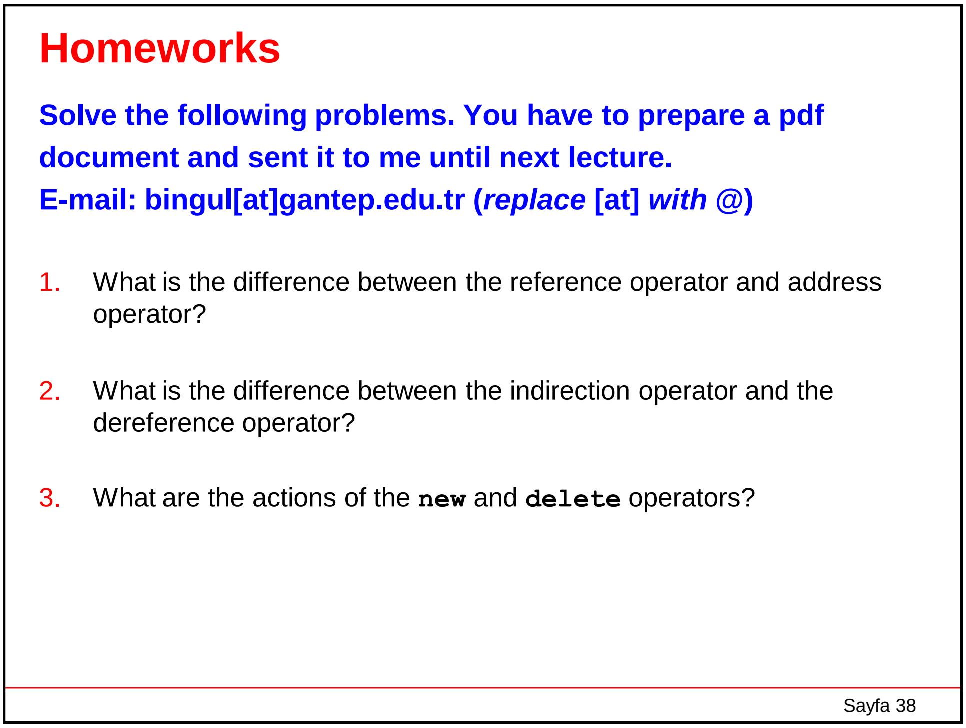


Using vectors with functions

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

double max(vector<double> v){
  double eb = v[0];
  for(int i=0; i<v.size(); i++){
    if(v[i]>eb) eb = v[i];
  }
  return eb;
}
int main() {

 int n;
 cout << "Input n: ";
 cin >> n;
 vector<double> x(n);

 for(unsigned int i=0; i<x.size(); i++) cin >> x[i];

 cout << "maximum element is: " << max(x) << endl;
}
```

Output

```
Input n: 4
1.1
2.2
-4.3
0.4
maximum element is: 2.2
```



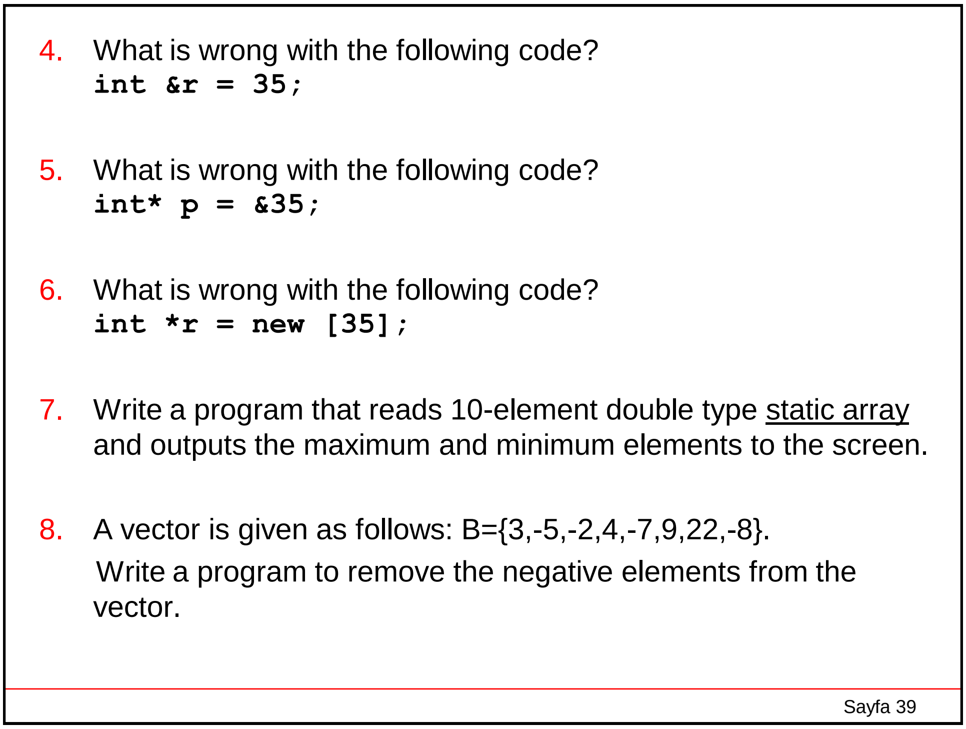
Homeworks

Solve the following problems. You have to prepare a pdf document and sent it to me until next lecture.
E-mail: bingul[at]gantep.edu.tr (*replace* [at] *with* @)

1. What is the difference between the reference operator and address operator?

2. What is the difference between the indirection operator and the dereference operator?

3. What are the actions of the `new` and `delete` operators?



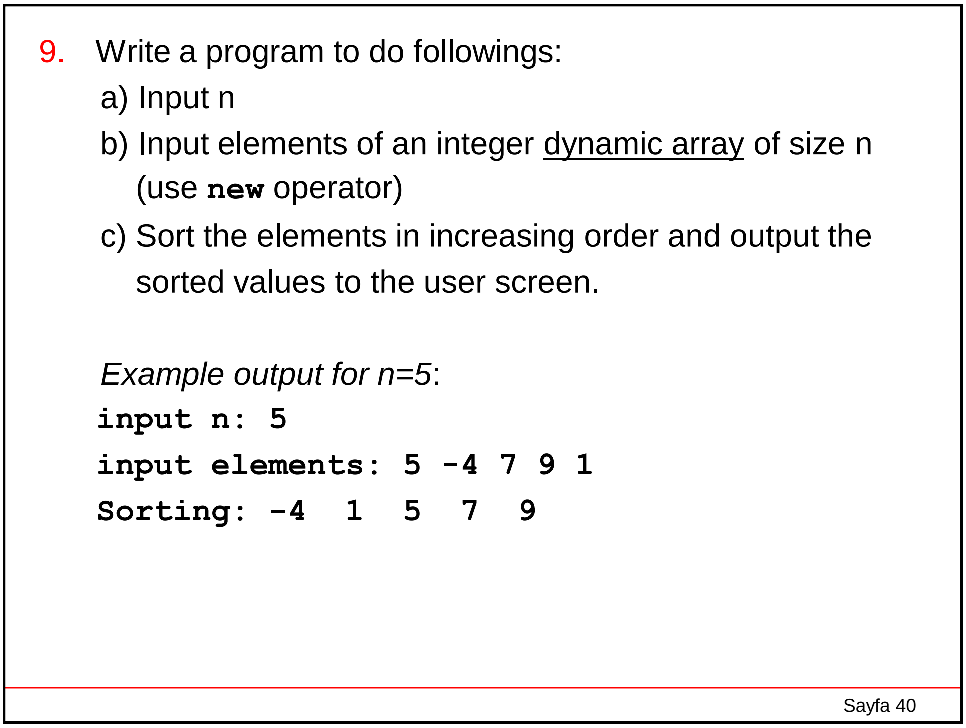
4. What is wrong with the following code?
   `int &r = 35;`

5. What is wrong with the following code?
   `int* p = &35;`

6. What is wrong with the following code?
   `int *r = new [35];`

7. Write a program that reads 10-element double type static array and outputs the maximum and minimum elements to the screen.

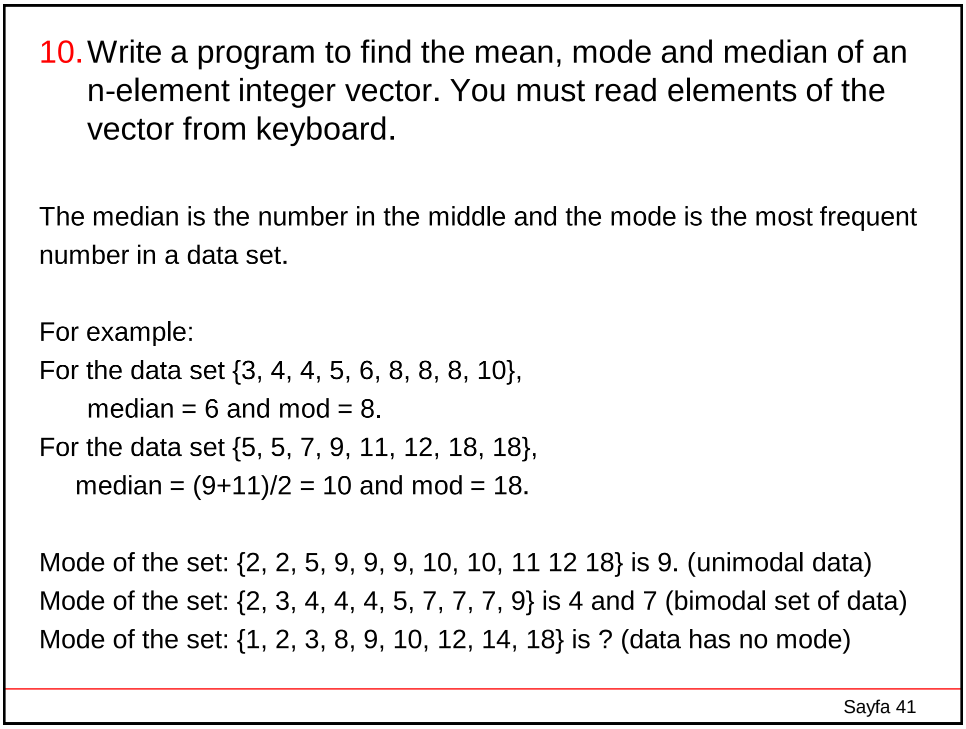
8. A vector is given as follows: B={3,-5,-2,4,-7,9,22,-8}.
   Write a program to remove the negative elements from the vector.



9. Write a program to do followings:
   a) Input n
   b) Input elements of an integer dynamic array of size n (use **new** operator)
   c) Sort the elements in increasing order and output the sorted values to the user screen.

   Example output for n=5:

```
input n:  5
input elements:  5 -4 7 9 1
Sorting: -4   1   5   7   9
```



10. Write a program to find the mean, mode and median of an n-element integer vector. You must read elements of the vector from keyboard.

The median is the number in the middle and the mode is the most frequent number in a data set.

For example:
For the data set {3, 4, 4, 5, 6, 8, 8, 8, 10},
median = 6 and mod = 8.
For the data set {5, 5, 7, 9, 11, 12, 18, 18},
median = (9+11)/2 = 10 and mod = 18.

Mode of the set: {2, 2, 5, 9, 9, 9, 10, 10, 11 12 18} is 9. (unimodal data)
Mode of the set: {2, 3, 4, 4, 4, 5, 7, 7, 7, 9} is 4 and 7 (bimodal set of data)
Mode of the set: {1, 2, 3, 8, 9, 10, 12, 14, 18} is ? (data has no mode)