



# Introduction to C++

*Georgia Koutsandria*

Internet of Things A.Y. 18-19

Prof. Chiara Petrioli

Dept. of Computer Science

Sapienza University of Rome



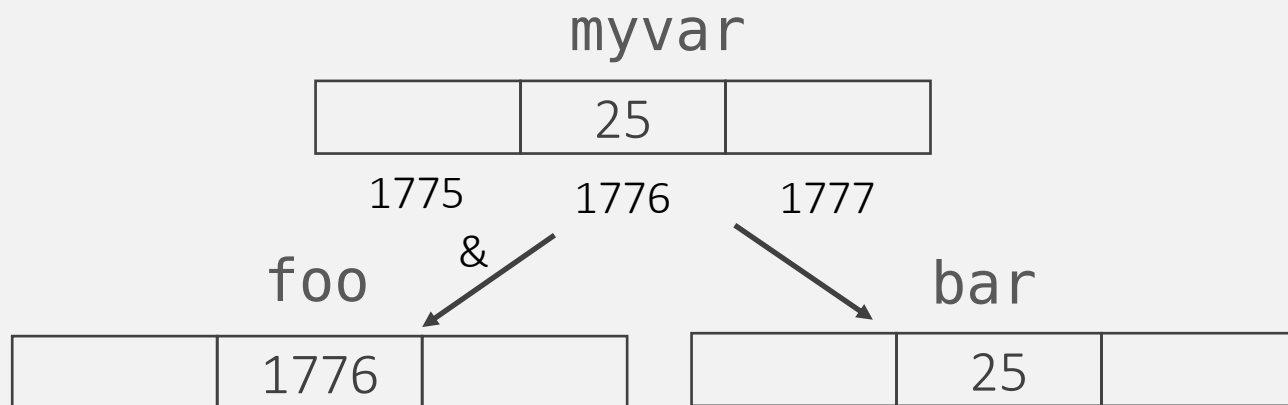
# Pointers



# Pointers



- Variables: Locations in the computer's memory which can be accessed by their identifier (their name).
- The address of a variable can be obtained by using the ampersand sign(&).
- Pointer: The variable that stores the address of another variable.

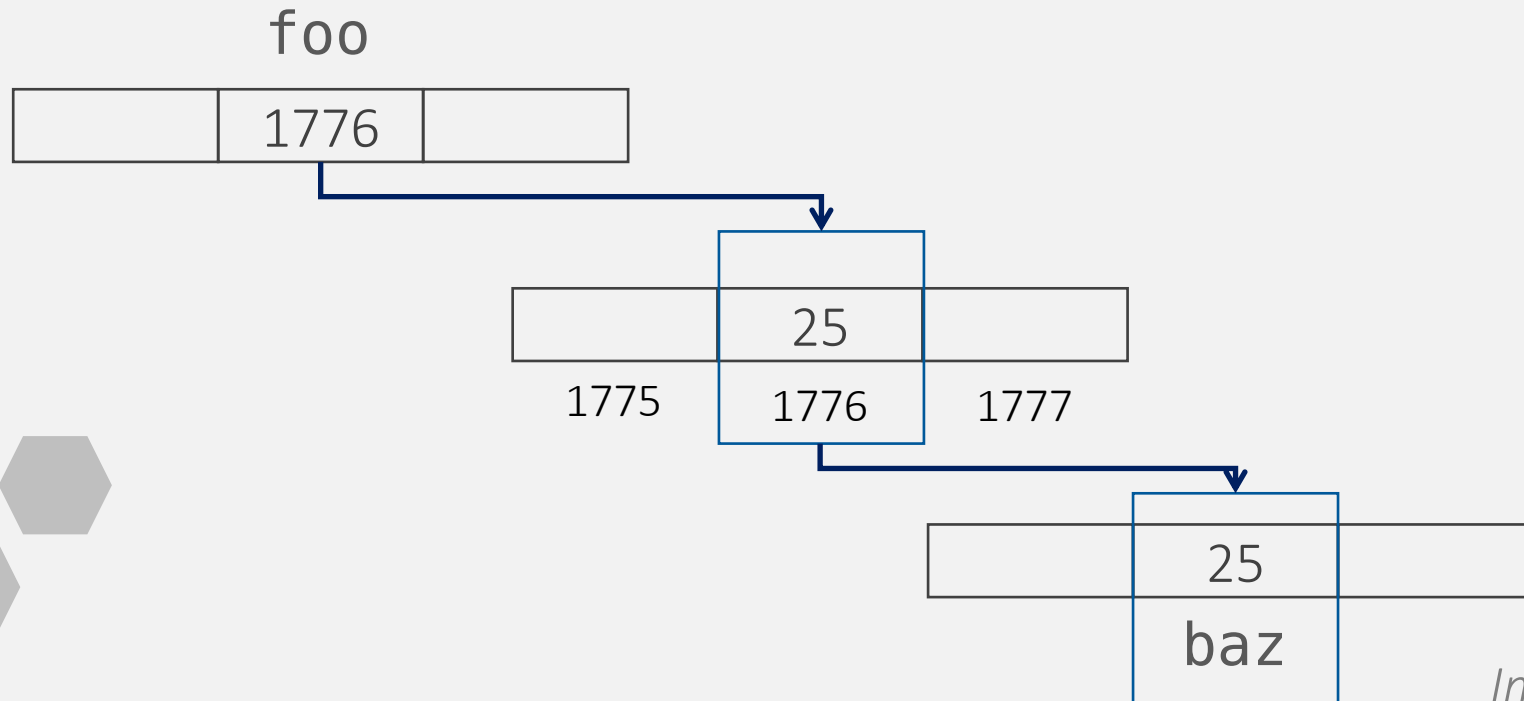


```
myvar = 25;  
foo = &myvar;  
bar = myvar;
```

# Pointers



- They can be used to access the variable they point to directly.
- A dereference operator (\*) is preceding the pointer's name.
- Pointer: The variable that stores the address of another variable.



```
myvar = 25;  
foo = &myvar;  
baz = *foo;
```

# Declaring Pointers



- They have different properties when they point to a `char` than when they point to an `int` or `float`.
- Their declaration needs to include the data type are going to point to.
- Syntax: `type * name;`
- The asterisk means that a pointer is declared which should not be confused with the dereference operator.



# Pointers- An example



```
#include <iostream>
using namespace std;
int main(){
    int firstvalue, secondvalue;
    int * mypointer;

    mypointer = &firstvalue;
    *mypointer = 10;
    mypointer = &secondvalue;
    *mypointer = 20;
    cout << "firstvalue is " << firstvalue << endl;
    cout << "secondvalue is " << secondvalue << endl;

    return 0;
}
```





# Pointers and Arrays



- An array can always be implicitly converted to a pointer of a proper type.
- Pointers and arrays support the same set of operations.
- **Exception:** Pointers can be assigned a new address, while arrays cannot.
- The name of an array can be used like a pointer to its first element.



# Pointers and Arrays-An example



```
#include <iostream>
using namespace std;
int main(){
    int numbers[5];
    int * p;
    p = numbers; *p = 10; p++; *p = 20;
    p = &numbers[2]; *p = 30;
    p = numbers + 3; *p = 40;
    p = numbers; *(p+4) = 50;
    for (int n=0;n<5;n++)
        cout << numbers[n] << ", ";
    return 0;
}
```

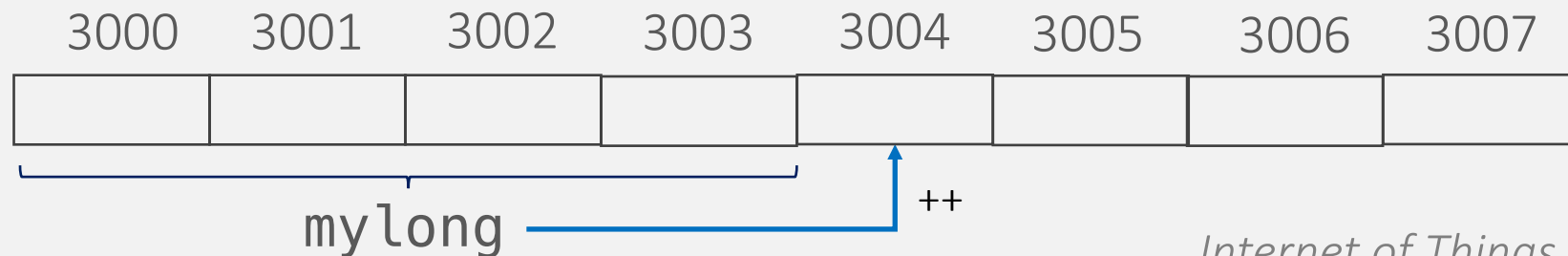
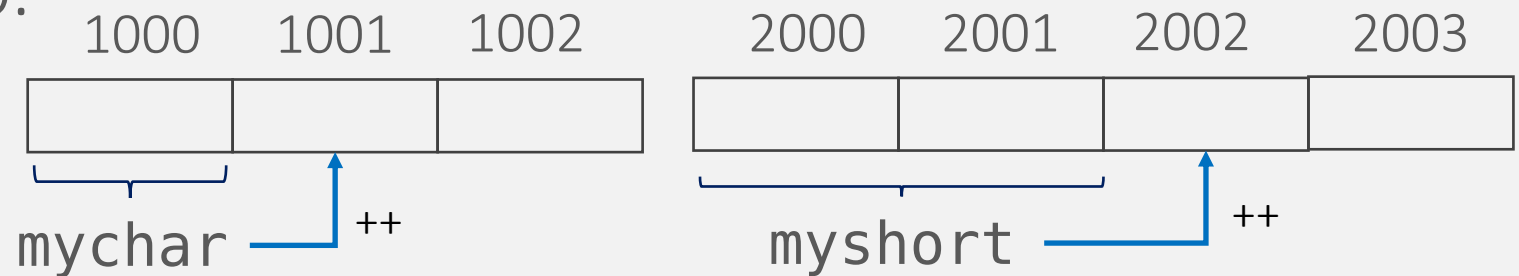


# Pointers arithmetics



- Only addition/subtraction operations are allowed.
- Operations depend on the size of the data type to which they point.
- E.g.: In a given system, a `char` takes 1 byte, a `short` takes 2 bytes, and `long` takes 4 bytes. 3 pointers that point to memory locations 1000, 2000, and 3000.

```
char * mychar;  
short * myshort;  
long * mylong;
```



# Pointers arithmetics



- The increment/decrement operators can be used as either prefix or suffix of an expression.
- The increment/decrement operator has a higher precedence than the \*.

```
//increment pointer, and dereference unincremented address
*p++; //same as *(p++);
//increment pointer, and dereference incremented address
*++p; //same as *(++p);
//dereference pointer, and increment the value it points to
++*p; //same as ++(*p);
//dereference pointer, and post-increment the value it
points to
(*p)++;
```

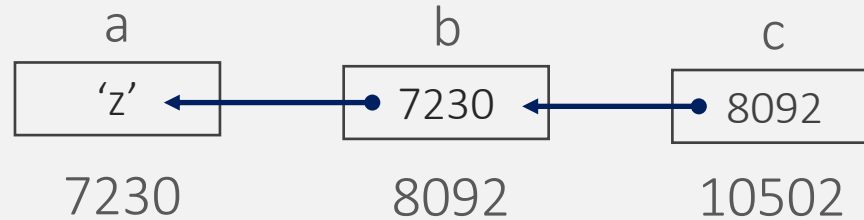


# Pointers to Pointers



- The syntax requires an asterisk (\*) for each level of indirection in the declaration of the pointer.

```
char a;  
char * b;  
char ** c;  
a = 'z';  
b = &a;  
c = &b;
```



- Variable c can be used in three different levels of indirection
  - c is of type char\*\* and a value of 8092.
  - \*c is of type char\* and a value of 7230.
  - \*\*c is of type char and a value of 'z'.



# Exercise 1



Write a program with a function that swaps (exchanges the values of) two integer numbers. You should pass the arguments to the function by reference using pointers. Display the values of the two numbers before and after swapping them.



# Exercise 1-Solution



```
#include <iostream>
using namespace std;
void swap(int* n1, int* n2) {
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
int main(){
    int a = 10, b = 20;
    cout << "Before swapping: ";
    cout << "a = " << a << ", b = " << b << endl;
    swap(&a, &b);
    cout << "After swapping: ";
    cout << "a = " << a << ", b = " << b << endl;
    return 0;
}
```



## Exercise 2



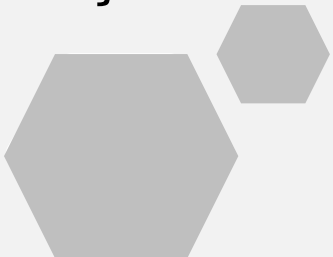
Write a program to print the elements of an array in reverse order using pointers. Print the elements of the array before and after reversing it.



# Exercise 2-Solution



```
#include <iostream>
using namespace std;
int main(){
    int array[4] = {1, 2, 3, 4};
    int *p = array;
    cout << "Array before reversing: " << endl;
    for (int i=0;i<4;i++)
        cout << *(p+i) << endl;
    cout << "These reversed array is: " << endl;
    for (int j=3;j>=0;j--)
        cout << *(p+j) << endl;
    return 0;
}
```







# Sequential Containers in the C++ standard library



# Sequential Containers



- Standard library includes several container types, e.g., `array`, `vector`, `string`, `list`, `forward_list`, `deque`.
- The order of the elements corresponds to the positions in which the elements are added to the container.
- Built-in functions, e.g., sorting and ordering.



# Sequential Containers



- Array: A fixed-size container.
- Other containers: We can add or remove elements, growing and shrinking the size of the container.
- Containers offer different performance trade-offs relative to
  - The cost to add or delete elements
  - The cost to perform nonsequential access to the elements.



# Sequential Containers



- Array:
  - Fixed-size
  - Fast random access
  - Cannot add/delete elements.
- String and Vector:
  - Hold their elements in contiguous memory
  - Fast random access
  - Fast insert/delete at the back
  - Insert/delete other than the back may be slow.



# Sequential Containers



- Deque:
  - Supports fast random access
  - Adding/removing elements in the middle of a deque is a (potentially) expensive operation.
  - Adding/removing elements at either end is a fast operation.
- List and forward\_list:
  - Fast insert/delete at any point of the list
  - Do not support random access to elements; access is done by iterating through the container
  - Substantial memory overhead.



# Which sequential container to use?



- Unless you have a reason to use another container, use a `vector`.
- Lots of small elements and space overhead matters, don't use `list` or `forward_list`.
- Random access to elements: `vector` or `deque`.
- Insert/delete elements in the middle of the container: `list` or `forward_list`.
- Insert/delete elements at the front and the back (not in the middle): `deque`.



# Which sequential container to use?



The predominant operation of the application (whether it does more access or more insertion or deletion) will determine the choice of the container type.





# Array



- A fixed-size sequence container; No memory management.
- Holds a specific number of elements ordered in a strict linear sequence.
- Appropriate header: `#include <array>`

```
//array holds 2 objects of type int; initialized  
array<int, 2> myarray = {2, 8};  
//array holds 2 objects of type int; initialized  
array<int, 2> myarray{2, 8};  
//10 objects of type int, each initialized to 10  
array<int, 10 > myarray;
```



# Arrays: An example



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

int main(){
    array<int,4> myarray = {1, 2, 3, 4};
    cout << "Element of myarray at position 1 is: "
         << myarray[1] << endl;
    return 0;
}
```



# Built-in vs. Library Arrays



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

```
int main(){
    int myarray[3] = {10,20,30};
    for(int i=0;i<3;i++)
        ++myarray[i];
    for(int elem:myarray)
        cout<<elem<<endl;
    return 0;
}
```

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

```
int main(){
    array<int,3> myarray{10, 20, 30};
    for(int i=0;i<myarray.size();i++)
        ++myarray[i];
    for(int elem:myarray)
        cout<<elem<<endl;
    return 0;
}
```



# Using arrays



- `begin ()` - returns an iterator to the first element in the array
- `end()` - returns an iterator to the past-the-end element
- `size ()` - Returns the number of elements in the array
- `empty ()` - returns whether the array is empty
- `at ()`-returns a reference to the element at a specific position
- `front()`-returns a reference to the first element in the array
- `back()`-returns a reference to the last element in the array
  - `fill()`-sets a value for all elements in the array
  - Operator `[]`-returns a reference to the element at a specified position in the array.



# Vectors



- A collection of objects which have the same type.
- Every object has an associated index which allows access to that object.
- Efficient and flexible memory management.
- Appropriate header:

```
#include <vector>
```



# Vectors



```
//vector vec holds objects of type T; vec is empty  
vector<T> vec;
```

```
vector<int> vec(4); //vec holds 4 objects of type int
```

```
//4 objects of type int, each initialized to 10  
vector<int> vec(4, 10);
```

```
vector<string> vec(4, "hi!"); //4 strings, all initialized to "hi!"
```

```
vector<vector<int>> vec //vector whose objects are vectors
```



# Vectors: An example



```
//include vector header
#include <vector>
using namespace std;

int main(){
    vector<int> vec(4);
    vec[1]=5;
    return 0;
}
```





# Using vectors



- `begin ()` - returns an iterator to the first element in the vector
- `end()` - returns an iterator to the past-the-end element
- `size ()` - Returns the number of elements in the vector
- `empty ()` - returns whether the vector is empty
- `at ()`-returns a reference to the element at a specific position
- `insert()`-inserts a new element at a specified position
- `erase()`-removes either a single element or a range of elements
  - `push_back()`-adds a new element at the end of the vector
  - `pop_back()`-removes the last element in the vector
  - `clear()`-removes all elements from the vector
  - Operator `[]`-returns a reference to the element at a specified position



# Using vectors



```
#include <vector>
using namespace std;
int main(){
    vector<float> ivec(10);

    for(int i=0; i<ivec.size(); ++i)
        ivec.at(i) = 5.0f*float(i);

    return 0;
}
```

Built-in function to  
get the size of a  
vector

access element of  
vector ivec at  
position i



# Using vectors



```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  int main(){
5      vector<int> myvector;
6      vector<int>::iterator it;
7
8      it = myvector.begin();
9      myvector.insert (it,10);
10     it = myvector.end();
11     myvector.insert (it,2,30);
12     cout << "myvector contains:";
13     for (it=myvector.begin(); it<myvector.end(); it++)
14         cout << ' ' << *it;
15     cout << endl;
16     cout << "element at position 0 is: " << myvector.at(0) << endl;
17     cout << "size of vector is: " << myvector.size() << endl;
18     myvector.erase (myvector.begin()+2);
19     cout << "size of vector is: " << myvector.size() << endl;
20     myvector.push_back(50);
21     cout << "myvector contains:";
22     for (int i=0; i<myvector.size(); i++)
23         cout << ' ' << myvector[i];
24     cout << endl;
25     myvector.clear();
26     cout << "size of vector is: " << myvector.size() << endl;
27     return 0;
28 }
```

# Exercise 1



Write a program to implement the various functions of a vector.

- A vector initially has two integer elements initialized to 50.
- Print the contents and the size of the vector.
- Insert at the beginning of the vector elements from 1 to 10 (10 9 8 7 6 5 4 3 2 1 ).
- Print the contents and the size of the vector.
- Remove the last element of the vector and print again the size.



# Exercise 1-Solution



```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  int main(){
5      vector<int> vec(2,50);
6      vector<int>::iterator it;
7      cout << "vec has size " << vec.size() << " and contains: ";
8      for(int i=0; i<vec.size(); i++)
9          cout << ' ' << vec[i];
10     cout << endl;
11
12     it = vec.begin();
13     int j=0;
14     for(j=1;j<11;j++){
15         it = vec.begin();
16         vec.insert(it,j);
17     }
18     cout << "vec has size " << vec.size() << " and contains: ";
19     for(int i=0; i<vec.size(); i++)
20         cout << ' ' << vec[i];
21     cout << endl;
22     vec.pop_back();
23     cout << "vec has size " << vec.size() << endl;
24     return 0;
25 }
```



# Associative Containers in the C++ standard library



# Associative Containers



- Elements are stored and retrieved by a key.
- Two primary associative container types: map and set.
- The C++ library provides eight associative containers.





# Associative Container Types



Container Type	
<code>map</code>	Holds key-value pairs
<code>set</code>	The key is the value
<code>multimap</code>	A key can appear multiple times
<code>multiset</code>	A key can appear multiple times
<code>unordered_map</code>	Organized by a hash function
<code>unordered_set</code>	Organized by a has function
<code>unordered_multimap</code>	Hashed map; keys can appear multiple times
<code>unordered_multiset</code>	Hashed set; keys can appear multiple times





# The map associative container



- A collection of (key, value) pairs; often referred to as an associative array.
- Values are found by a key rather than by their position (as in arrays).
- E.g.: Mapping names to phone numbers; Each pair contains a person's name as a key and a phone number as its value.

```
#include <map>
```

```
map<key, value> name;
```



# The map associative container



- E.g.: `map<string, int> words;`
  - Key = Word (string)
  - Value = Word's frequency count (int)
- Several basic functions:
  - `begin ()` - returns an iterator to the first element in the map
  - `end()` - returns an iterator to the theoretical element that follows the last element in the map
  - `size ()` - Returns the number of elements in the map
  - `empty ()` - returns whether the map is empty
  - `at ()` - returns the reference to the element associated with the key.



# The map associative container



- Several basic functions (cont.):
  - `insert(key,value)`-adds a new element to the map
  - `erase(iterator position)`-removes the element at that position
  - `erase(const g)`-removes the key value 'g' from the map
  - `clear()`-removes all elements
  - `find()` –returns the iterator to the entry having key equal to given key.
- Operator `[]`-used to reference the element present at position given inside the operator.
  - Operator = assigns contents of a container to a different container.



# map : An example



```
1  #include <iostream>
2  #include <map>
3  #include <string>
4  using namespace std;
5  int main (){
6      map<string,string> car{{"Gabriele","Fiat"}, {"Georgia", "Audi"}};
7      map<string,string> car_new;
8      map<string,string>::iterator i, iter;
9      for (i=car.begin();i!=car.end();i++)
10         cout << "Name: " << i->first << ", car: " << i->second << endl;
11      car.insert(pair<string,string>("Daniele","Renault"));
12      cout << "Name: Daniele" << ", car: " << car["Daniele"] << endl;
13      iter = car.find("Georgia");
14      if (iter!=car.end())
15         car.erase(iter);
16      cout << "Elements in car:" << endl;
17      for (i=car.begin();i!=car.end();i++)
18         cout << "Name: " << i->first << ", car: " << i->second << endl;
19      cout << "Size of car: " << car.size() << endl;
20      car_new = car;
21      cout << "Size of car new: " << car_new.size() << endl;
22      car_new.at("Gabriele") = "Ford";
23      for (i=car_new.begin();i!=car_new.end();i++)
24         cout << "Name: " << i->first << ", car: " << i->second << endl;
25      while (!car.empty()){
26         cout << car.begin()->first << " ==> " << car.begin()->second << endl;
27         car.erase(car.begin());}
28      cout << "Size of car: " << car.size() << endl;}
```

# Exercise 1



Write a program to print the grades of the students based on the following data (use the map container). Then you should only print the grades of students with name = Gabriele and name = Christian.

<u>Name</u>	<u>Grade</u>
Georgia	29
Gabriele	26
Chiara	30
Christian	23

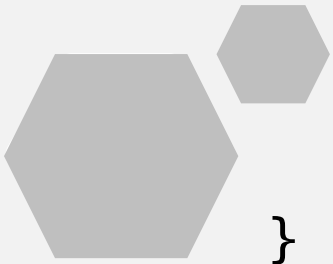


# Exercise 1-Solution



```
#include <iostream>
#include <map>
using namespace std;
int main(){
    map<string,int>grades{{"Georgia",29},{ "Gabriele",26},
                        {"Chiara",30},{ "Christian",23}};
    map<string,int>::iterator i;
    cout << "The grades of all students are: ";
    for (i=grades.begin();i!=grades.end();i++)
        cout << i->second << " ";
    cout << endl;
    cout << "The grade of student Gabriele is: "
        << grades["Gabriele"] << endl;
    cout << "The grade of student Christian is: "
        << grades["Christian"] << endl;

    return 0;
}
```





## Exercise 2



Change the program of Exercise 1 as follows:

- Insert two new elements in the map
- Print the element of the map with key = "Christian"
- Print the size of the map
- Erase the element with key= "Chiara"
- Print all the elements of map
- Reprint the size of the map.



# Exercise 2-Solution



```
1  #include <iostream>
2  #include <map>
3  using namespace std;
4  int main(){
5      map<string,int>grades{{"Georgia",29},{ "Gabriele",26},
6                          {"Chiara",30},{ "Christian",23}};
7      map<string,int>::iterator i, iter, iter1;
8      cout << "The grades of all students are: ";
9      for (i=grades.begin();i!=grades.end();i++)
10         cout << i->second << " ";
11     cout << endl;
12     cout << "The grade of student Gabriele is: "
13         << grades["Gabriele"] << endl;
14     cout << "The grade of student Christian is: "
15         << grades["Christian"] << endl;
16
17     grades.insert(pair<string,int>("Edoardo",25));
18     grades.insert(pair<string,int>("Daniele", 19));
19     iter = grades.find("Christian");
20     if (iter!=grades.end())
21         cout << "Grade of student Christian is: " << iter->second << endl;
22     cout << "Size of grades is: " << grades.size() << endl;
23     iter1 = grades.find("Chiara");
24     if(iter1!=grades.end())
25         grades.erase(iter1);
26     cout << "The grades of all students are: ";
27     for (i=grades.begin();i!=grades.end();i++)
28         cout << i->second << " ";
29     cout << endl;
30     cout << "Size of grades is: " << grades.size() << endl;
31     return 0;
32 }
```



# Additional Resources



- <http://www.cplusplus.com/doc/tutorial/>
- <https://en.cppreference.com/w/>
- Programming: Principles and Practice Using C++, Bjarne Stroustrup (Updated for C++11/C++14)
- C++ Primer, Stanley Lippman, Josée Lajoie, and Barbara E. Moo (Updated for C++11)

