



Associative Containers in the C++ standard library (Recap++)



Associative Container Types

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









- A collection of keys.
- Useful when we want to know whether a value is present.
- It keeps the elements ordered at all times.
- E.g.: A business might define a set named bad_checks to hold the names of individuals who have written bad checks.

#include <set>



```
//empty set of ints
set<int> first;
//range
int myints[] = \{10, 20, 30, 40\};
set<int> second(myints, myints+4);
//a copy of the set named second
set<int> third(second);
//iterator
set<int> fourth(second.begin(), second.end());
```





- Several basic functions:
 - begin () returns an iterator to the first element in the set
 - end() returns an iterator to the past-the-end element in the container
 - size () Returns the number of elements in the set
 - empty () returns whether the set container is empty
 - insert()-adds a new element to the set
 - erase()-removes an element from the set
 - find() –returns the iterator to a specific element.
 - Operator = assigns new contents to set replacing the current ones.

```
// count the number of times each word occurs
// in the input
map<string, size_t> word_count; //empty map
set<string> exclude = {"Then","But","An"};
string word;
while(cin >> word){
     //count only words that are not in exclude
     if (exclude.find(word) == exlude.end())
           ++word_count[word];
```





Range-based Loop



Range-based loop

• A more readable equivalent to the traditional for loop operating over a range of values, such as all elements in a container (array, vector, map, set, etc.).



- For observing elements in a container. i.e., read-only:
 - 1. If the objects are cheap to copy (capture by value)

for (auto elem : container_name)

2. Capture by const reference

for (const auto& elem : container_name)

- When modifying the elements in the container:
 - Capture by non-const reference

for (auto& elem : container_name)



Loop through Map

```
map<string,int>::iterator it;

for(it=myMap.begin();it!=myMap.end();it++)
{
    cout << it->first << ": "
        << it->second
        << endl;
}</pre>
```



auto: Tells the compiler to deduce the type of a declared variable from its initialization expression.

Loop through Set



```
for(auto elem : mySet)
    cout << elem << " , ";</pre>
```

auto: Tells the compiler to deduce the type of a declared variable from its initialization expression.



Exercise 1



Write a program that initializes a set which contains 5 integers and prints the contents of the set container. Use two different ways to loop through the set: 1) Using an iterator; 2) Using type inference (auto).



Exercise 1: Solution

```
#include <iostream>
#include <set>
using namespace std;
int main(){
    int mynumbers[] = \{23, 10, 45, 5, 3\};
    set<int> myset (mynumbers, mynumbers + 5);
    cout << "myset contains: ";</pre>
    for(set<int>::iterator iter = myset.begin() ;iter != myset.end();iter++)
       cout << " " << *iter;</pre>
    cout << endl;</pre>
    cout << "myset contains: ";</pre>
    for(auto elem : myset)
       cout << " " << elem;
    cout << endl;</pre>
    return 0;
```





- A group of data elements grouped together under one name.
- These data elements are called members which can have different types and lengths.

```
struct type_name{
   member_type1 member_name1;
   member_type2 member_name2;
   member_type3 member_name3;
}
```



- type_name: The name of the structure type.
- member_name: The name of the data member.
- object_names: A set of valid identifiers for objects that have the type of this structure.

```
struct type_name{
   member_type1 member_name1;
   member_type2 member_name2;
   member_type3 member_name3;
}
```





It declares a structure type, called product

```
struct product{
    int weight;
    double price;
};
```

Two members, each of a different type

```
product apple; _____ Three objects of structure product melon, orange; _____ type are declared.
```



Data Structures (Alternative option)



```
struct product{
    int weight;
    double price;
};

product apple;
product melon, orange;
```

```
struct product{
    int weight;
    double price;
} apple, melon, orange;
```

Name objects can be used to directly declare objects of the structure type.



Accessing the members

- Once a member is declared, it can be accessed directly.
- Syntax: Insert a dot (.) between the object name and the member name.
- E.g.: Each of the data type corresponds to the member it refers to.
 - apple.weight
 - apple.price
 - melon.weight
 - melon.price
 - orange.weight
 - orange.price



Data Structures: An example

```
#include <iostream>
                                                int main(){
#include <string>
                                                  string mystr;
#include <sstream>
                                                  mine.title = "Goodbye Bafana";
using namespace std;
                                                  mine year = 2007;
struct movies t{
                                                  cout << "Enter a title: ";</pre>
  int year;
                                                  getline(cin, yours.title);
  string title;
                                                  cout << "Enter year: ";</pre>
}mine, yours;
                                                  getline(cin, mystr);
                                                  stringstream(mystr) >> yours.year;
void printmovie (movies_t movie){
                                                  cout << "My favorite movie is: ";</pre>
       cout << movie.title;</pre>
                                                  printmovie(mine);
       cout << " (" << movie.year << ")"</pre>
                                                  cout << "Your favorite movie is: ";</pre>
             << endl;
                                                  printmovie(yours);
                                                  return 0;
```



Pointers to Structures



Pointers to Structures

• A structure can be pointed to by its own type of pointers.



```
struct movies_t{
                        An object of structure
  int year;
                            type movies t
  string title;
};
                              A pointer that points to
                              objects of structure type
movies_t amovie;
movies_t * pmovie;
                                     movies t
pmovie = &amovie;
                           The value of the pointer
                           pmovie is assigned the
                          address of object amovie.
```

Pointers to Structures

• The arrow operator (->) is a dereference operator that is used exclusively with pointers to objects that have members; It allows access to the member of an object directly from its address.

Expression	What is evaluated	Equivalent
a.b	Member b of object a	
a->b	Member b of object pointed to by a	(*a) . b
*a.b	Value pointed to by member b of object a	*(a.b)

Pointers to Structures: An example

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
struct movies t{
  int year;
  string title;
int main(){
  string mystr;
  movies t amovie;
  movies_t * pmovie;
  pmovie = &amovie;
  cout << "Enter a title: ";</pre>
  getline(cin, pmovie->title);
  cout << "Enter year: ";</pre>
  getline(cin, mystr);
  stringstream(mystr) >> pmovie->year;
  cout << "You have entered: " << pmovie->title;
  cout << " (" << pmovie->year << ")" << endl;</pre>
  return 0;
```





Nesting Structures



Nesting Structures

• Structures can be nested in such a way that an element of a structure is itself another structure.



```
struct movies_t{
  int year;
  string title;
};
struct friends_t{
  int year;
  string name;
  string email;
  movies_t favorite_movie;
}gina, gabriele;
friends_t * pfriends = &gina;
```

gina.name
gabriele.favorite_movie.title
gina.favorite_movie.year
Pfriends->favorite_movie.year

Exercise 1

Write a program that implements a structure **array** to construct a database for the products of a supermarket. Your program should take as input the name and the price of 5 products (from the keyboard/user) and it should display them on the screen in a table manner.

- product: a data structure.
- pr: an array structure/object of size 5.
- name: member to store the name of the product.
- price: member to store the price of the product.



Enter the name of product 1: Milk Enter the price of product 1: 0.9

Enter the name of product 2:Shampoo Enter the price of product 2:5.23

Enter the name of product 3:Water Enter the price of product 3:0.5

Enter the name of product 4:Bread Enter the price of product 4:1.99

Enter the name of product 5:Sugar Enter the price of product 5:2.34

Product Name	Price (Euro)
Milk	0.9
Shampoo	5.23
Water	0.5
Bread	1.99
Sugar	2.34

Exercise 1-Solution

```
#include <iostream>
using namespace std;
struct product{
    char name[20];
    float price;
} pr[5];
int main(){
   for(int i=0;i<5;i++){</pre>
       cout << "Enter the name of product " << i+1 << ":";</pre>
       cin >> pr[i].name;
       cout << "Enter the price of product " << i+1 << ":";
       cin >> pr[i].price;
       cout << endl;</pre>
    cout << "Product Name" << "\t \t" << "Price (Euro)" << endl;</pre>
    for(int i=0;i<5;i++)
       cout << pr[i].name << "\t \t" << pr[i].price << endl;</pre>
    return 0;
```





Classes



Classes

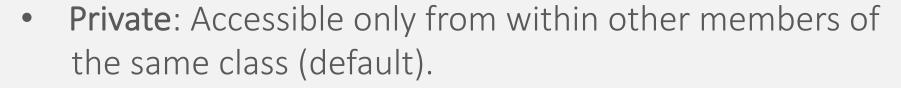
- class_name: A valid identifier for the class.
- object_names: An optional list of names for objects;
 An object is an instantiation of a class.
- members: Contained in the body of the declaration; can be data or function declarations.
- access_specifiers: Modify the access rights for the members of the class (optional).



```
class class_name{
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
}object_names;
```



Access specifier

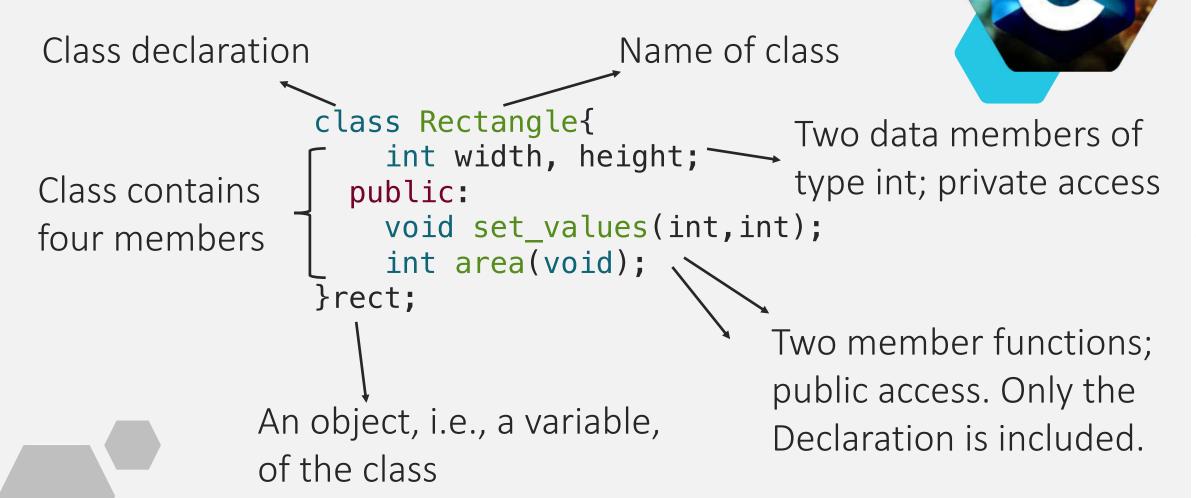




- **Protected**: Accessible from other members of the same class and also from members of their derived classes.
- **Public**: Accessible from anywherhe where the object is visible.



An example



Class vs. Object name

- Rectangle: The class name
- rect: An object of type Rectangle



The type name The variable name (the class) (the object)



```
class Rectangle
int width, height;
  public:
    void set_values(int,int);
    int area(void);
}rect;
```



Accessing public members of a class

 Public objects can be accessed as if they were normal functions or variables.



• Use of dot (.) between object name and member name.

```
• E.g.: rect.set_values(3,4);

myarea = rect.area();
```



```
class Rectangle{
int width, height;
  public:
    void set_values(int,int);
    int area(void);
}rect;
```

Accessing members of a class



```
class Rectangle{
    int width, height;
    public:
    void set_values(int,int);
    int area(void);
}rect;
```

Members with private access cannot be accessed from outside of the class.

They can only be referred to from within other members of the same class.





Q: What would happen if your program tries to access a private data member from outside of a class?



An example

```
#include <iostream>
using namespace std;
class MyClass{
      int var1, var2;
};
int main()
 MyClass mc;
  mc.var1 = 10;
  cout << "var1: " << mc.var1 << endl;</pre>
  return 0;
```





Q: What would happen if your program tries to access a private data member from outside of a class?



A: Compilation will fail! You will get the following error:



Defining a member function

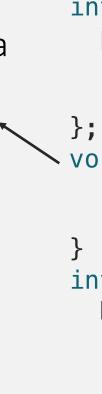


- 1. <u>Within the class definition:</u> Function is automatically considered an inline member function by the compiler.
- 2. <u>Include declaration and define it later outside the class:</u> A normal (not-inline) class member function.



An example

The scope operator (::) is used in the definition of a class member to define a member of class outside the class itself.



```
#include <iostream>
using namespace std;
class Rectangle{
int width, height;
  public:
    void set_values(int,int);
    int area(){return width*height;}
void Rectangle::set_values(int x, int y){
      width = x;
      height = y;
int main(){
  Rectangle rect;
  rect.set_values(3,4);
  cout << "area: " << rect.area() << endl;</pre>
  return 0;
```

The scope operator(::)



- It specifies the class to which the member being defined belongs.
- It grands exactly the same scope properties as if this function definition was directly included within the class definition.



Multiple object declaration

```
#include <iostream>
using namespace std;
class Rectangle{
int width, height;
  public:
    void set_values(int,int);
    int area(){return width*height;}
};
void Rectangle::set_values(int x, int y){
       width = x:
       height = y;}
                                       Two instances
int main(){
  Rectangle rect, rectb;
                                          (objects)
  rect.set_values(3,4);
  rectb.set_values(5,6);
  cout << "area: " << rect.area() << endl;</pre>
  cout << "areab: " << rectb.area() << endl;</pre>
  return 0;
```





Q: What would happen in the previous example if we called the member function area before having called set_values?



Q: What would happen in the previous example if we called the member function area before having called set_values?



A: An undetermined result, since the members width and height had never been assigned a value.



Q: What would happen in the previous example if we called the member function area before having called set_values?

```
#include <iostream>
using namespace std;
class Rectangle{
int width, height;
  public:
    void set_values(int,int);
    int area(){return width*height;}
void Rectangle::set_values(int x, int y){
       width = x;
       height = y;}
int main(){
  Rectangle rect, rectb;
  cout << "area: " << rect.area() << endl;</pre>
  rect.set_values(3,4);
  rectb.set_values(5,6);
  cout << "areab: " << rectb.area() << endl;</pre>
  return 0;
```



Q: What would happen in the previous example if we called the member function area before having called set_values?



A: An undetermined result, since the members width and height had never been assigned a value.

|Georgias-MacBook-Pro:C++ examples gina\$ g++ -std=c++11 rectangleError.cpp -o rectangleError |Georgias-MacBook-Pro:C++ examples gina\$./rectangleError

area: 1718552992

areab: 30



Constructor

- A special function which is automatically called whenever a new object of a class is created.
- It allows the class to initialize member variables or allocate storage.
- They are only executed once, when a new object is created.
- Declaration: like a regular member function; the name matches the class name; no return type.





Constructor - An example

```
#include <iostream>
using namespace std;
class Rectangle{
int width, height;
  public:
    Rectangle(int, int);
    int area(){return width*height; }
};
Rectangle::Rectangle(int a, int b){
       width = a;
       height = b;
                                        Constructor
int main(){
  Rectangle rect(3,4);
  Rectangle rect_b(5,6);
  cout << " rect area: " << rect.area() << endl;</pre>
  cout << " rect_b area: " << rect_b.area() << endl;</pre>
  return 0;
```



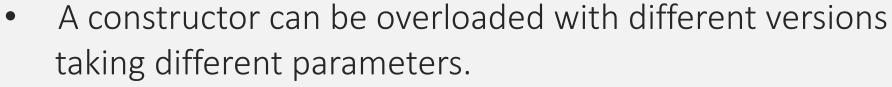




Overloading constructors



Overloading constructors





- The compiler will automatically call the one whose parameters match the arguments.
- The *default constructor*: A special kind constructor that takes no parameters. It is called when an object is declared but is not initialized with any arguments.

Rectangle rectb; // ok, default constructor called
Rectangle rectc(); // Oops!



An example

```
#include <iostream>
using namespace std;
class Rectangle{
int width, height;
  public:
    Rectangle(int, int);
                                                  int main(){
    int area(){return width*height; }
                                                    Rectangle rect(3,4);
};
                                                    Rectangle rect b;
Rectangle::Rectangle(){
                                                    cout << "rect area:" << rect.area()</pre>
       width = 5;
                                                          << endl;
        height = 5;
                                                    cout << " ect b area: " << rect b.area()</pre>
                                                          << endl;
Rectangle::Rectangle(int a, int b){
                                                    return 0;
       width = a;
        height = b;
```



Calling constructors



Calling constructors

• functional form: Enclose the arguments of the constructor in parentheses.

```
class_name object_name ( value1, value2, value3, ... )
```

Single parameter:

```
class_name object_name = initialization_value;
```

 Uniform initialization: Same as the functional form but using braces instead of parentheses. (Optional: an equal sign before the braces.)

```
class_name object_name { value1, value2, value3, ... }
```



An example

```
#include <iostream>
using namespace std;
class Circle{
    double radius;
  public:
    Circle(double r){radius = r;};
    double circum(){return 2*radius*3.14159265; }
};
int main(){
  Circle foo(10.0); //functional form
  Circle bar = 20.00; // assignment init.
  Circle baz {30.00}; // uniform init.
  Circle qux = \{40.00\}; //uniform init.
  return 0;
```





Member initialization in constructors



Member initialization

- When a constructor is used to initialize other members, these members can be initialized directly.
- Initialization is done by inserting, before the contructor's body, a colon (:) and a list of initializations for class members.

```
class Rectangle{
int width, height;
  public:
    Rectangle(int, int);
    int area(){return width*height; }
};
```



```
    Rectangle::Rectangle(int a, int b) { width = a; height = b; }
    Rectangle::Rectangle(int a, int b) : width(a) { height = b; }
    Rectangle::Rectangle(int a, int b) : width(a), height(b) { }
```



Pointers to classes



Pointers to classes

Objects can be pointed to by pointers.



- The members of an object can be accessed directly from a pointer by using the arrow operator(->).
- Syntax:

class_name * pointer_name;



Operators

Expression	
*X	Pointed to by x
&x	Address of x
X . y	Member y of object x
x->y	Member y of object pointed to by x
(*x) • y	Member y of object pointed to by x
x[0]	First object pointed to by x
x[0]	Second object pointed to by x
x[n]	(n+1)th object pointed to by x





Pointers to classes: An example

```
#include <iostream>
using namespace std;
class Rectangle{
int width, height;
  public:
    Rectangle(int x, int y): width(x), height(y){};
    int area(void){return width*height; }
int main(){
  Rectangle rect(3,4);
  Rectangle * foo, * bar, * baz;
  foo = ▭
  bar = new Rectangle (5,6);
  baz = new Rectangle[2]\{\{2,5\},\{3,6\}\};
  cout << " rect's area: " << rect.area() << endl;</pre>
  cout << " *foo's area: " << foo->area() << endl;</pre>
  cout << " *bar's area: " << bar->area() << endl;</pre>
  cout << " baz[0] area: " << baz[0].area() << endl;</pre>
  cout << " baz[1] area: " << baz[1].area() << endl;</pre>
  delete bar:
  delete[] baz;
  return 0;
```





Exercise 1

Write a class (call it Student) that contains the following members: 1) First name; 2) Last name;
 3) Student ID; 4) Grade (private access).



• The class Student should also contain the following two member functions (public access): 1) storeData():Stores the details of a student (fname, lname, etc..); 2) printData():Prints the details of a student.



Your program should store the details of 3 students (given as input by the user) and then print the details of all students.

Exercise 1 (cont.)

Student # 1 ******* Enter first name: Georgia Enter last name: Koutsandria Enter the id of the student: 12345 Enter the grade of the student: 30 Student # 2 ******** Enter first name: Gabriele Enter last name: Saturni Enter the id of the student: 23456 Enter the grade of the student: 29 Student # 3 ******** Enter first name: Christian Enter last name: Cardia Enter the id of the student: 34567 Enter the grade of the student: 28 ************** F.Name L.Name Grade Georgia Koutsandria 12345 30

Saturni

Cardia

23456

34567

29

28

Gabriele

Christian





Exercise 1 - Solution

```
#include <iostream>
using namespace std;
class Student{
      string fname, lname;
      int student_id, grade;
public:
    void storeData();
    void printData();
};
void Student::storeData(){
       cout << "Enter first name: ", cin >> fname;
       cout << "Enter last name: ", cin >> lname;
       cout << "Enter the id: ", cin >> student_id;
       cout << "Enter the grade: ", cin >> grade;
       cout << endl;</pre>
void Student::printData(){
       cout << fname << " " << lname << "
           << student_id << " " << grade << endl;
                                             Internet of Things A.Y. 18-19
```



Exercise 1 – Solution(cont.)

```
int main(){
     Student students[3];
     for (auto i=0;i<3;i++){
          cout << "Student # " << i+1 << endl;</pre>
          cout << "***************** << endl;
          students[i].storeData();
     cout << "************************ << endl;</pre>
     cout << "F.Name " << " << "L.Name " << "
         << " ID " << " Grade " << endl;
     for (auto i=0;i<3;i++)
          students[i].printData();
     return 0;
```

Exercise 2



Redo exercise 1 using class and pointers.



Exercise 2 - Solution

```
#include <iostream>
using namespace std;
class Student{
      string fname, lname;
      int student_id, grade;
public:
    void storeData();
    void printData();
};
void Student::storeData(){
       cout << "Enter first name: ", cin >> fname;
       cout << "Enter last name: ", cin >> lname;
       cout << "Enter the id: ", cin >> student_id;
       cout << "Enter the grade: ", cin >> grade;
       cout << endl;</pre>
void Student::printData(){
       cout << fname << " " << lname << "
           << student_id << " " << grade << endl;
                                             Internet of Things A.Y. 18-19
```



Exercise 2 – Solution(cont.)

```
int main(){
     Student students[3];
     Student *studentsp;
     studentsp = &students[0];
     for (auto i=0;i<3;i++){
           cout << "Student # " << i+1 << endl;</pre>
           cout << "************************ << endl:
           (studentsp+i)->storeData();
     cout << "**************************** << endl;
     cout << "F.Name " << " << "L.Name " << "
         << " ID " << " Grade " << endl;
     for (auto i=0;i<3;i++)
            (studentsp+i)->printData();
     return 0;
```

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)



