محاضرات في مادة:

# الخوارزميات وهياكل البيانات

# Algorithms and Data Structures

**د/ قسم السيد إبراهيم**

**جامعة أفريقيا العالمية**

# Course Contents:

(1)IntroductiontoC++

(2)PointersandDynamicObjects

(3)LinkedLists

(4)Stacks

(5)Queues

(6)Trees

(7)Sorting

(8)Searching

(9)Graphs

# References:


- "DataStructuresandAlgorithmAnalysisinC++",byMarkAllenWeiss, Addison-Wesley,2006.
- "DataStructuresandAlgorithmsinC++",byAdamDrozdek,2nded., 2008.
- "C++PlusDataStructures",byNellDale,JonesandBartlettPublishers, 1999.

# Lecture# 1

## Introductionto C++

# Outline

- C++basicfeatures
  - Statementsyntax.
- Classdefinitions
  - Data members,methods,constructor,destructor.
  - Pointers,arrays,andstrings.
  - Parameterpassinginfunctions.

# BasicC++

- InheritallC syntax
  - Primitivedatatypes
    - Supported data types: **`int`**, **`long`**, **`short`**, **`float`**, **`double`**, **`char`**, and **`bool`**.
  - Basic expressionsyntax
    - Definingtheusualarithmeticandlogical operations suchas: **`+`**, **`-`**, **`*`**, **`/`**, **`%`**, **`&&`**, **`||`**, and **`!`**.
  - Basic statementsyntax
    - **`If-else`**, **`switch`**, **`for`**, **`while`**, and **`do-while`**.

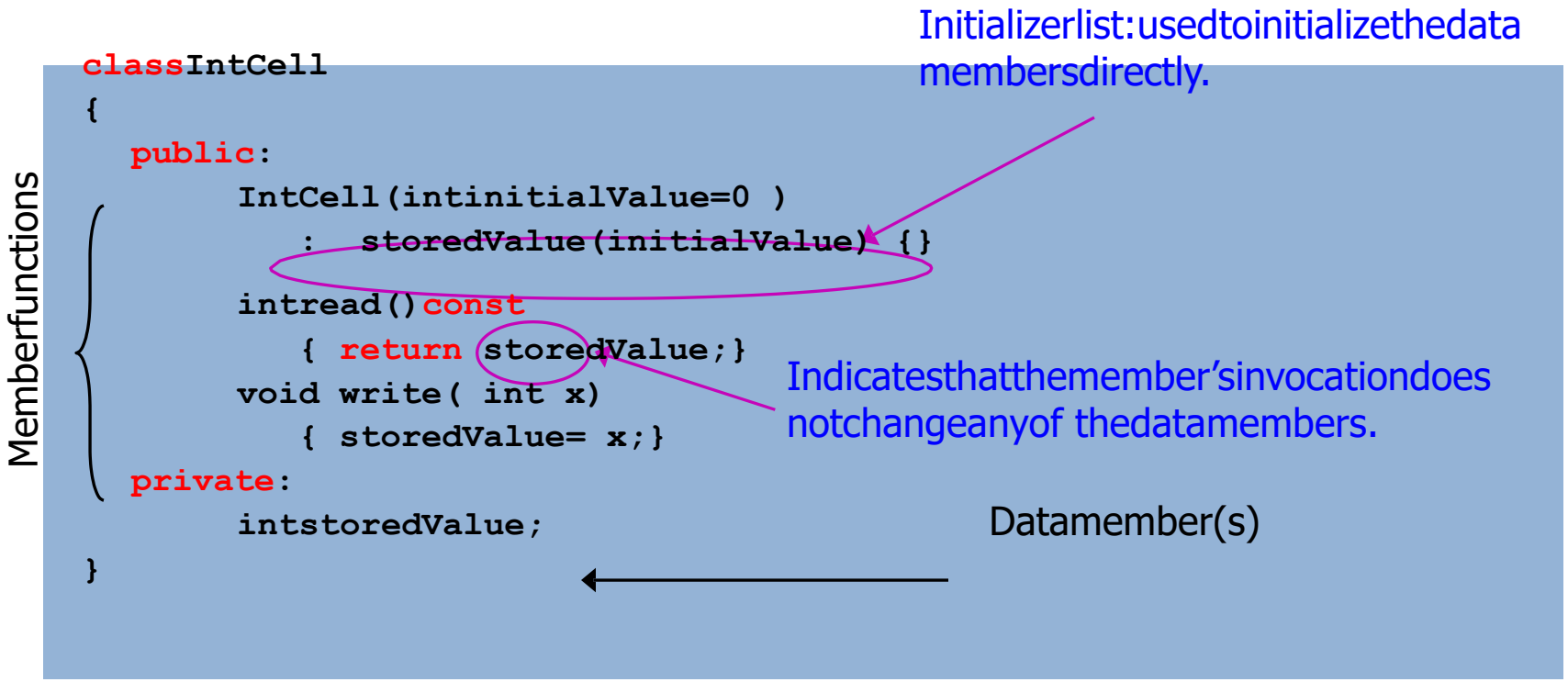
# BasicC++(cont…)

- Addanewcommentmark:
  - //…. : for1 line comment.
  - /*….*/ : for agroupofline comment.

- *const*support for constantdeclaration, justlikesC.

# ClassDefinitions

- AC++class consistsof*datamembers* and *methods*(*member functions*).



# InformationHidingin C++

- Twolabels: *public*and*private*
  - Determinevisibilityofclassmembers.
  - Amemberthat is *public*canbeaccessedbyanymethod in anyclass.
  - Amemberthat is *private*only canbeaccessedby methodsinitsclass.

- Informationhiding
  - Data membersaredeclared*private*,thusrestricting accessto internal detailsoftheclass.
  - Methodsintendedfor general usearemade*public*.

# Constructors

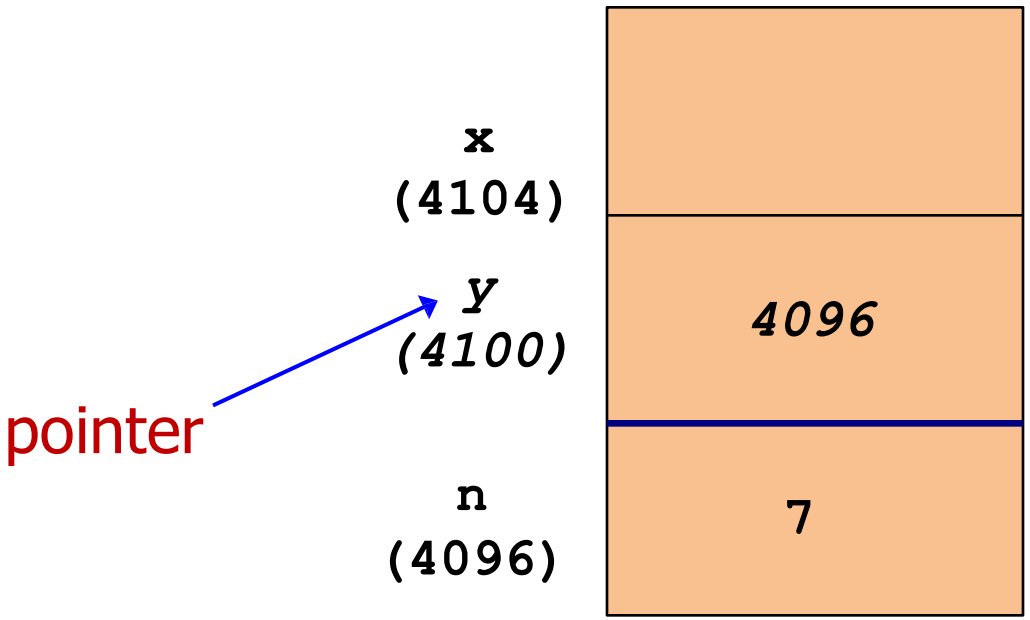
- A *constructor* is a special method that describes how an instance of the class (called *object*) is constructed.
- Whenever an instance of the class is created, its constructor is called.
- C++ provides a *default constructor* for each class, which is a constructor with no parameters.
  - But, one can define multiple constructors for the same class, and may even redefine the default constructor.
  - How to distinguish them?

# Destructor

- Typically, the *destructor* is used to free up any resources that were allocated dedduring the use of the object.
- A *destructor* is called when an object is deleted either implicitly, or explicitly (using the *delete* operation).
  - The destructor is called whenever an object goes out of scope or is subjected to a *delete*.
- C++ provides a *default destructor* for each class.
  - The default simply applies the destructor on each data member.
  - But we can redefine the destructor of a class.
- A C++ class can have only one destructor.

# Pointers

- A *pointer* is a variable which contains address of other variable.
- Accessing the data at the contained address is called "dereferencing a pointer" or "following a pointer".



# APointer Example

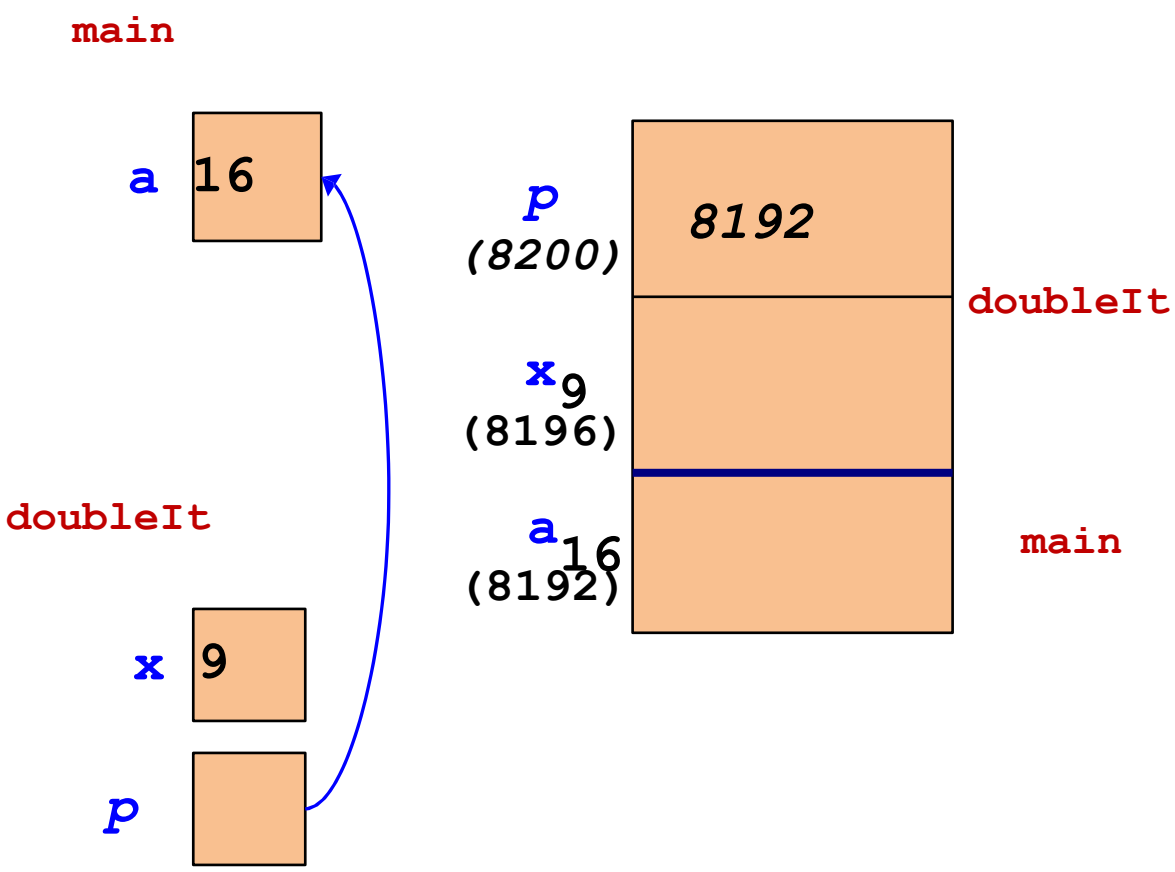
## Thecode

```
voiddoubleIt(int x, int *p)
{
   *p= 2 * x;
}
intmain()
{
   inta = 16;
   doubleIt(9,&a);
   cout<<"a= "<< a;
   return0;
}
```

Output:

a = 18

## BoxdiagramMemoryLayout



# ObjectPointerDeclaration

- Declaration:

```
ClassName*PointerName;
```

Example:

```
IntCell*p;
//definesa pointer to an object of class IntCell
```

- The * indicates that *p* is a pointer variable; it is allowed to point at an IntCell object.
- The *value* of *p* is the address of the object that it points at.
- *p* is uninitialized at this point.

# DereferencingPointers

- Dynamic objectcreation


*p*=*new* IntCell;

InC++*new*returns a pointerto thenewly createdobject.

- Garbagecollection
  - C++does nothavegarbagecollection.
  - Whenanobjectthat**isallocated by *new***is no longer referenced, the*delete* operationmust beapplied totheobject.

delete*p;*

# DereferencingPointers(cont…)

- Usingapointer
  - Wecanaccessaclassmemberof theobjectpointedatby a pointerbyusingoperator"->".

```
p=newIntCell;
intb;
……
b=p->read();//thevalueofthedatamember
                //storedValueoftheobjectpointedat
                  //bypisassignedto b
cout<<b<<endl;
p->write(50);
cout<<p->read()<<endl;
```

Theoutputis:

0
50

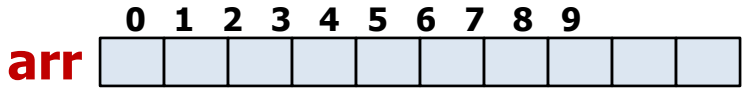
# Array of Objects

- An *array* of objects is a collection of objects with same type stored consecutively in memory.

- **Declaring static array:**

```
ClassNameArrayName[size];
```

Example:




```
IntCellarr[10];//anarrayconsistingof10IntCellobjects
```

- The size of the array must be known at compile time.
- *arr* actually is a constant pointer.
  - The value of *arr* **cannot** be changed.
- The *(i+1)*-st object in the array *arr* can be accessed either by using ***arr[i]***, or by ****(arr+i)***.

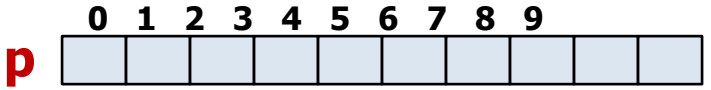
# Arrayof Objects(cont…)

- **Declaringdynamicarray:**

```
ClassName*PointerName;
PointerName=newClassName[size];
```

Example:

```
IntCell*p=newIntCell[10];
arr=p;     //invalid
```




  - The*(i+1)*-stobjectin thearray*p*canbeaccessedeither by using***p[i]***,orby***(p+i)***.

- Arraysofobjects cannotbecopiedwith„=„ .

- Arrays are*not* passedbycopy.Instead,theaddressof thefirst elementis passedto thefunction.

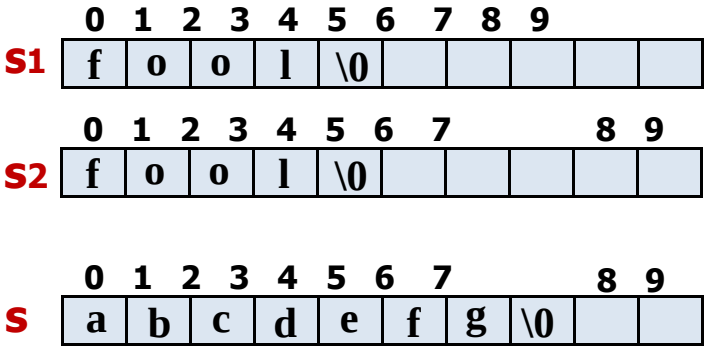
```
intsumOfArray(intvalues[],intnumValues)
```

# Strings

- Built-inC-stylestrings areimplementedas anarrayof characters.
- Each string ends withthespecial null-terminator„**\0**“.
- Commonstring functions:
  - ***strcpy***:usedto copy strings.
  - ***strcmp***:used tocompare strings.
  - ***strcat***: used tojoinstrings.
  - ***strlen***:used todeterminethelengthof strings.
- Individualcharacters canbeaccessedbythearrayindexing operator.

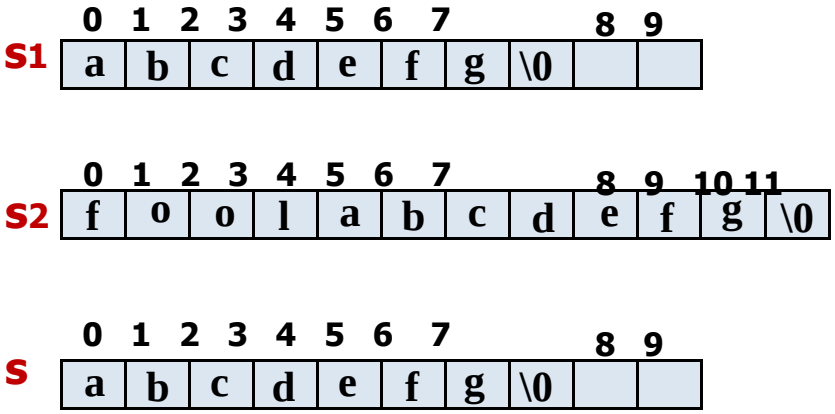
# Strings (cont…)

```
chars1[]="fool";
chars2[]="fool";
chars[]="abcdefg";
```




---

```
if(strcmp(s1,s2)==0)
    cout<<"Samestrings.";
elsecout<<"Differentstrings.";
strcpy(s1,s);//copystos1
strcat(s2,s);//addstos2
cout<<strlen(s1)<<endl;
cout<<strlen(s2)<<endl;
cout<<strlen(s)<<endl;
```



# FunctionCallbyValue

```
voidf(intx)
{  cout<<"valueof x= " <<x <<endl;
   x =4; }
main()
{ intv =  5;
  f(v);
  cout<<  "value of v=" << v<<endl;}
```

*Output:*

valueof x= 5
valueof v =5

- Whenavariable*v*is passed *byvalue*toafunction*f*, its valueis copiedto thecorrespondingvariable*x* in*f*.
- Anychanges tothevalueof*x* doesNOT affect thevalueof *v inthe mainprogram*.
- Call byvalueisthe defaultmechanismforparameter passing in C++.

# FunctionCallbyReference

```
voidf(int&x)
{cout<<   "value of x = " << x << endl;
   x=4;   }
main()
{intv=    5;
  f(v);
  cout<<  "value of v = " << v << endl;}
```

*Output:*

value of x = 5
valueof v =4

- Whena variable *v*ispassed*byreference*to a parameter *x* offunction *f*, *v* andthecorrespondingparameter*x* refer tothesame variable.
- Anychanges tothevalueof*x* DOES affect thevalue of *v*.

# FunctionCallbyConstant Reference

```
void f(constint&x)
{ cout<<"valueofx="<<x<<endl;
    x=4;    //invalid
}
main()
{intv=5;
  f(v);
  cout<<"valueofv="<<v<<endl;}
```

*Output:*

value of x =5

valueof v =5

- Passingvariable*v by constantreference*toparameter*x* of*f*will**NOT** allowany changetothevalueof*x*.
- Itisappropriateforpassinglargeobjectsthatshouldnotbe changed bythecalled function.

# Usage of ParameterPassing

- *Callbyvalue* is appropriateforsmallobjectsthat shouldnotbe changedbythefunction.
- *Callbyconstantreference*is appropriateforlargeobjectsthat shouldnotbechangedbythefunction.
- *Callbyreference*isappropriatefor allobjectsthat may be changedbythefunction.