Section 3

Classes

- كما هو معلوم فإن لغة C++ تشتمل على مجموعة من الـData types التي تستخدم في إنشاء المتغيرات.
- إن نوع المتغير يخبرك عن بعض السمات الخاصة بالمتغير، فإذا أعلنت عن المتغيرين (height, width) على أنهما متغيرين من النوع (unsigned integers) فانك ستعلم بان كل واحد منهما يمكن ان يحمل قيمة عددية تتراوح مابين (0..65,535)، وان محاولة تحميل المتغير باي شيء اخر غير مدى هذه القيم سيؤدي الى خطأ واضح، لذا فانك لاتتمكن من تخزين اسمك مثلاً في هذه المتغيرات.
- عليه فإن الـData types الخاصة بالمتغيرات تخبرك عن:
  1) حجم المتغيرات في الذاكرة
  2) ماهو نوع ومدى القيم التي ستخزن في المتغيرات
  3) وما هي العمليات التي يمكن إجاؤها على المتغيرات
- وبشكل عام يمكننا القول أن الـData type هو صنف (Class) للبيانات أو لأي شئ آخر.
- فعلى سبيل المثال تعتبر كل من (السيارة، الدار، الشخص، الفاكهة، الشكل، وغيرهم) هم بمثابة أنواع جديدة ولكنها ليست من أنواع البيانات.
- فأساس البرامج المكتوبة باللغة C++ هو الكائنات (Objects) التي يتم إنشاؤها بواسطة صنف (Class) يستعمل كقالب أو إطار عام لتلك الكائنات.
- فعندما يكون هنالك الكثير من الكائنات المتطابقة في البرنامج لا يكون منطقياً وصف كل واحد منها على حدة ، من الأفضل تطوير مواصفات واحدة لكل من هذه الكائنات وبعد تحديد تلك المواصفات يمكن استخدامها لإنشاء قدر ما نحتاج إليه من الكائنات
- تسمى هذه المواصفات بالـ Class.
- ويطلق على هذه البرمجة التي تقوم باستخدام الكائنات في كتابة الكود بالبرمجة الكائنية (OOP)

Class Basics

- الـClass هي عبارة عن **بنية برمجية** تجمع «**تغلف**" البيانات (الخصائص) والمهام (السلوكيات) لمجموعة من الكائنات المتشابهة.
- وعند تعريف أصناف في برنامج++C، يكون المبرمج قد أنشأ أنواعاً جديدة (New data types)، لذلك تصف الـClasses في سياق آخر بأنها عبارة عن أنواع بيانات معرفة من قبل المبرمج (User-defined data types).
- الصنف هو نوع بيانات يعرّفه المستخدم، ويُسبق بالكلمة المفتاحية class
- ويتألف الصنف من أعضاء يمكن أن تكون أيًا مما يلي:
  1) أعضاء بيانية (Data members) وتسمى كذلك بالمتغيرات العضوية
  2) دوالاً أعضاء (Member functions) وتسمى كذلك بالدوال التابعة
- **مثال:**
  - ✓ يُمكن التفكير في **السيارة** على أنها مجموعة من **الأبواب**، و**المقاعد**، و**النوافذ**، وغيرها
  - ✓ وبجانب آخر أخرى يُمكن التفكير في السيارة من خلال معرفة ما يُمكنها عمله: يُمكنها أن **تتحرك**، و**زيادة سرعتها**، و**تقليلها**، و**التوقف**، وغيرها.
  - ✓ تُتيح لك الـClass كبسلة أو تجميع هذه الأجزاء والإجراءات في مجموعة واحدة.

Class Basics

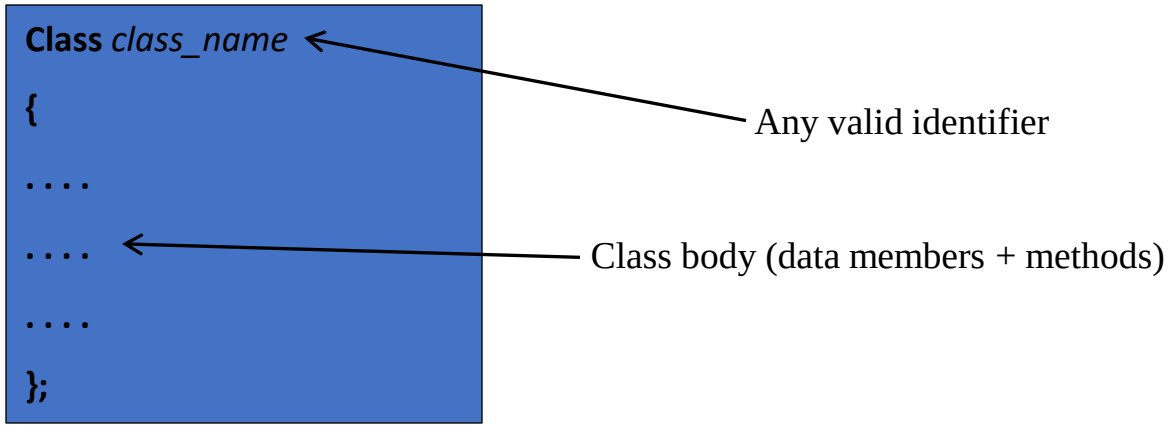
Class Definition

- يشير الـClass definition الى الكود الذي يمثل الـClass في برنامج الـC++
- يتألف الـClass definition من الكلمة المحجوزة class يليها اسم الصنف ثم جسم الصنف بين قوسين حاصرين { } ويجب أن ينهي تعريف الصنف بفاصلة منقوطة أو عبارة إعلان عن كائنات تنتمي إلى الصنف

```
class class_name {  /* class body*/  };
```

أو

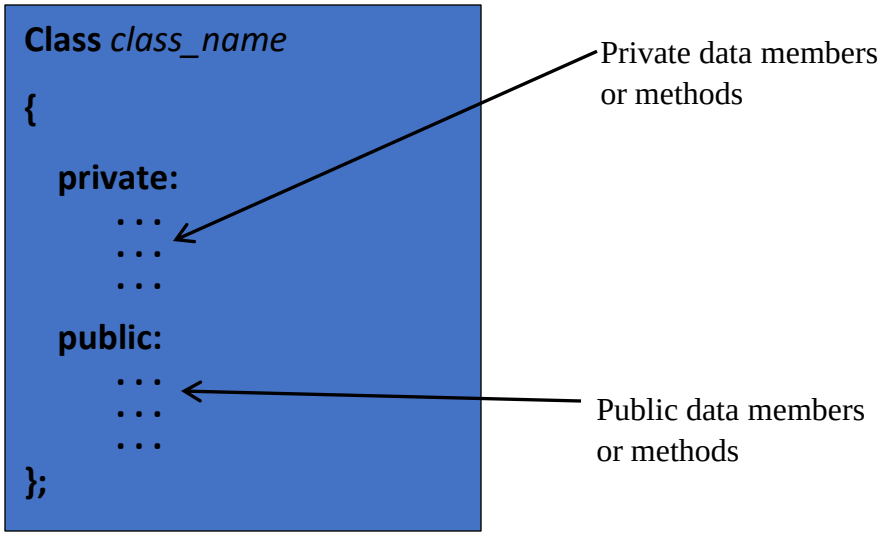
```
class class_name { /* class body */ }  obj1, obj2;
```



Class Basics

Class Definition

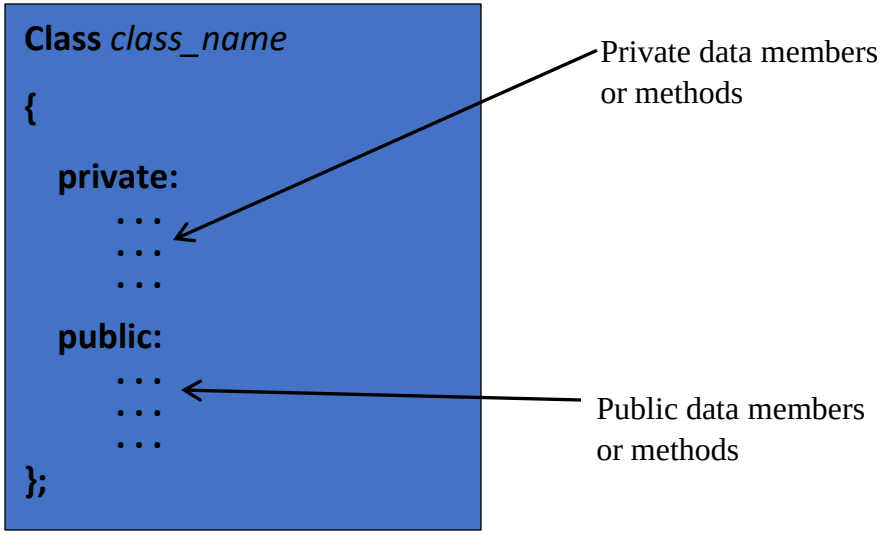
- غالباً ما يكتب الصنف في برنامج الـ C++ على النحو التالي:



- ✓ يشتمل جسم الصنف (Class body) على مجموعة من الأعضاء (Members)، ويمثل كلاً من هذه الأعضاء إما عضو بياتات (Data member) أو دله عضو (Member function)
- ✓ تكتب الأعضاء في جسم الصنف بصورة منظمة في شكل أجزاء كتابية (Sections)، وتقسم هذه الأجزاء الى ثلاثة أنواع هي:
  1. **private:**
  2. **public:**
  3. **protected:**
- ✓ هذه الأجزاء كتابتها بجسم الصنف إختيارية ويمكن أن تكرر عدة مرات كما يمكن كتابتها بأي ترتيب

Class Basics

Class Definition

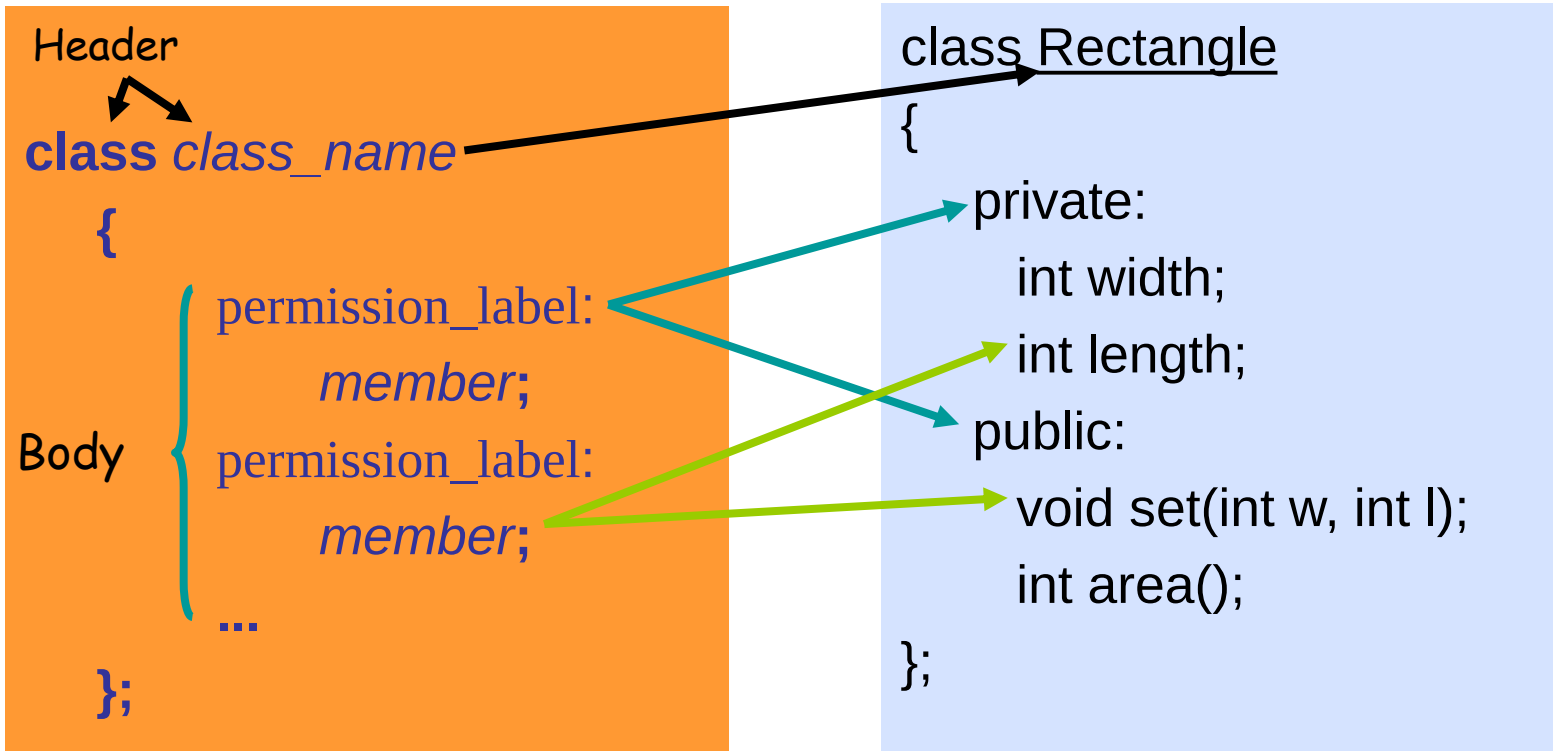


- ✓ تحدد هذه الأجزاء طريقة الوصول (Accessing) الى الأعضاء.
- ✓ فالأعضاء الذين يتم تضمينهم في الجزء public يمكن الوصول اليهم والتعامل معهم من أي مكان بالبرنامج.
- ✓ أما الأعضاء الذين يتم تضمينهم في الجزء private لايمكن الوصول اليهم أو التعامل معهم إلا من داخل الصنف المحتوي على الأعضاء.
- ✓ والأعضاء الذين يتم تضمينهم في الجزء protected فإنهم فقط يتم التعامل معهم من داخل الصنف المحتوي للأعضاء أو بواسطة أصدقائه.

Class Basics

Class Definition

المثال التالي يوضح التعريف (Class definition) للصنف Rectangle



- ✓ يشتمل الـDefinition على اثنين من الـData members واثنين من الـMember functions
- ✓ بين ماهي إمكانية الوصول لكلعضومنهذه الأعضاء؟

Class Basics

Class Definition

المثال التالي يوضح التعريف (Class definition) للصنف MyClass

```
class MyClass {        // The class
  public:              // Access specifier
    int myNum;         // Attribute (int variable)
    string myString;   // Attribute (string variable)
};
```

- بين ماهو إسم الClass؟
- أذكر ما هي أعضاء هذا الـClass وذلك مع تبيين النوع وأمكانية الوصول لكل من هذه الأعضاء

Class Basics

Object Declaration

- عند إنشاء كائن لصنف معين فإن هذا الأمر يشبه المثال التالي:
  - عندما نشتري كتاب طبخ، فإن **الوصفة** تمثل **الـClass** بينما **الطبق** الذي تنتجه هذه الوصفة يمثل **الـObject**
  - لذلك فإننا نقول أن هذه **الطبخة** من تلك **الوصفة** وهذا **الكائن** من ذاك **الصنف**
- يتكون العالم الحقيقي من كائنات (Objects)
  - ✓ بعضها يكون ملموس- مثلا انت كشخ، دفتر محاضراتك، سيارتك.
  - ✓ وبعضها غير ملموس- مثلا حسابك في البنك ، الكورس الذي تدرسه بالجامعة
- في العالم الحقيقي، فإن الكائن هوعبارة شئ له مواصفات (Attributes) ويتصرف بطريقة معينه (behaviors)
  - ✓ لذلك فإن مفهوم الكائن في لغة C++ هو عبارة عن تغليف (encapsulation) او حزم للبيانات والطرق في نموذج برمجي،
  - ✓ وبهذا فان **الكائن البرمجي** يوفر تمثيلا او تجريدا لكائنات العالم الحقيقي.
- العبارة التالية تبين الصورة العامة للإعلان عن الـObjects في لغة الـC++

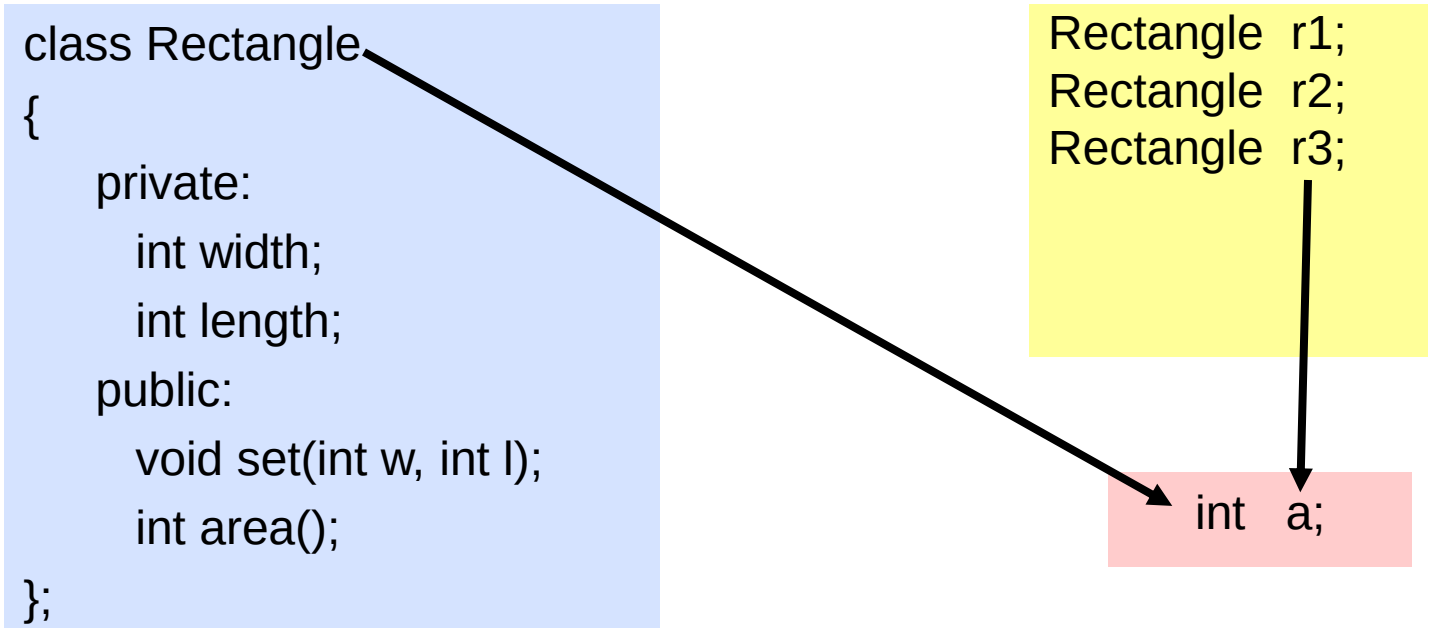
```
Class_name object_name;
```

- طريقة الإعلان عن الكائن(Object declaration) ببرنامج الـC++ هي نفس طريقة الإعلان المتغير.
- الرسم التالي يوضح أمثلة للإعلان عن بعض الكائنات

Class Basics

Object Declaration

- الرسم التالي يوضح أمثلة للإعلان عن بعض الكائنات



- عاةً أن الإعلان هو ما يحتاجه الـCompiler للإشارة لذلك الشيء المعلن،عليه فأنه ببساطة عندما يتم الإعلان عن Object سيتم حجز مكان في الذاكرة لذلك الكائن وتحفظ في هذا المكان كل أعضاء الكائن التي تم تعريفها في صنف الكائن.

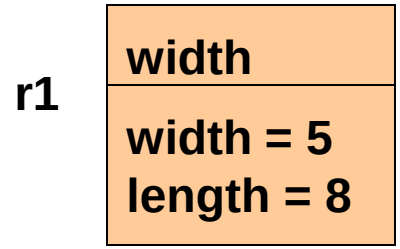
Class Basics

Object Declaration

- **r1** is statically allocated

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

```
main()
{
    Rectangle r1;
    r1.set(5, 8);
}
```



Class Basics

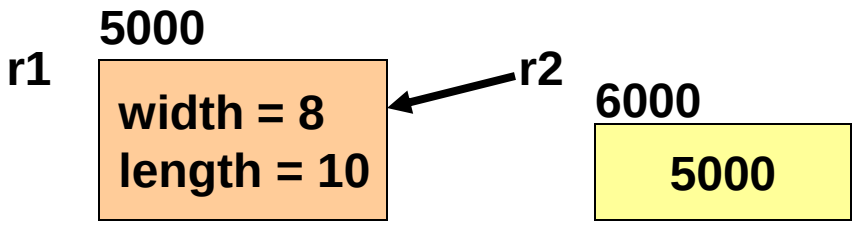
Object Declaration

- **r1** is a pointer to a Rectangle object

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

```
main()
{
    Rectangle r1;          //dot notation
    r1.set(5, 8);

    Rectangle *r2;
    r2 = &r1;              //arrow notation
    r2->set(8,10);
}
```



Class Basics

Object Declaration

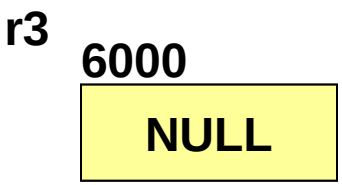
- **r3** is dynamically allocated

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

```
main()
{
      Rectangle *r3;
      r3 = new Rectangle();

      r3->set(80,100);

      delete r3;
      r3 = NULL;
}
```



Class Basics

Object Declaration

- فيما يلي مثالاً لكود يشتمل على بعض الأخطاء

```
Class X
{
        void f();
        int m;
};

void user(X x, X* px)
{
        m=1;            //error: there is no m in the scope
        x.m=1;          //OK
        x->m=1;         //error: x is not a pointer
        px->m=1;        //OK
        px.m=1;         //error: px is a pointer
}
```

Class Basics

Data Members (الأعضاء البيانية)

- يتم الإعلان عن الأعضاء البيانية في الـClass بنفس الطريقة التي يتم بها الإعلان عن المتغيرات باستثناء أنه لا يمكننا تمهيد الأعضاء البيانية عند الإعلان عنها.
- يمكن أن تكون الأعضاء البيانية من أي نوع بيانات في الـ C ++
- يمكن أن تكون أعضاء البيانات privateكما يمكن أن تكون public، ولكنها في الغالب تكون private.
- **مثال:**

```
class stack {
private:
    int s_elements[SIZE];
    int top;
public:
    void init ( );
    void push(int i);
    int pop ( );
};
```

- يحتوى الصنف stackعلى إثنين من أعضاء البيانات هما المصفوفة s_elements والتي عناصرها من النوع int والمتغير top من النوع int أيضا
- لاحظ أن هذه التعريفات لا تعطى المتغيرات أي قيمة، فهي فقط تعطيها اسماً وتحدد أنها تتطلب مساحة معينة من الذاكرة حيث يتم تخصيص مساحة الذاكرة بعد إنشاء الكائنات

Class Basics

Data Members (الأعضاء البيانية)

- بعد تعريف الصنف، يُضاف نوع جديد إلى برنامجك، ومن الممكن استنساخ كائنات من هذا الصنف على النحو التالي:

```
stack my_stack;
```

- تقع الأعضاء البيانية والدوال الأعضاء للـClass ضمن مجال رؤية الـClass.
- وضمن مجال رؤية الـClass يمكن الوصول مباشرة إلى أعضاء الـClass من قبل كافة الدوال الأعضاء التابعة للـClass وذلك فقط بذكرإسم العضو، أما في خارج المجال فيمكن الوصول إلى أعضاء الـClass العامة فقط من خلال اسم الكائن.
- من خارج هذا المجال يتم الوصول إلى أعضاء الصّنف بواسطة نقطة (.) تسمى بمعامل الوصول (**dot operator**).

```
my_stack. top = 10;
my_stack. s_elements[top] = 2;
```

- كيفية الوصول من خارج الصنف إلى الأعضاء العامة في الصنف:
  1) إستخدام إسم كائن للصنف يليه معامل النقطة (.)
  2) إستخدام مرجع (Reference) إلى كائن الصنف وعامل النقطة
  3) إستخدام مرجع (Reference) إلى كائن للصنف يليه المعامل (->)

- فيما يلي برنامج++C يشتمل على Class definition وإعلان عن object لهذا الـClass، إضافة الي بعض العبارات التي تنجز أعمالها عبر الوصول الى أعضاء الـClass

Class Basics

Data Members (الأعضاء البيانية)

- فيما يلي برنـامج++C يشتمل على Class definition وإعلان عن object لهذا الـClass، إضافة الي بعض العبارات التي تنجز أعمالها عبر الوصول الى أعضاء الـClass

```
class MyClass {        // The class
  public:              // Access specifier
    int myNum;         // Attribute (int variable)
    string myString;   // Attribute (string variable)
};

int main() {
  MyClass myObj;  // Create an object of MyClass

  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";

  // Print attribute values
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
}
```

Access Modifiers (محددات الوصول)

- إخفاء البيانات (Information hiding) هو جزء أساسي من البرمجة بلغة C++. لإخفاء البيانات هتالك ثلاثة أنواع من حماية الوصول المحددة في لغةC++
- يتم تحديد إمكانية الوصول إلى أعضاء الصنف (بيانات ، أعضاء دالية) في الـC++ باستخدام ثلاث كلمات محجوزة، هي:

public (عام), private (خاص), protected (محمي)

- تتم كتابة هذه الكلمات داخل جسم الصنف تليها نقطتان(:).
- يفيد تعريف الأعضاء البيانية والدوال الأعضاء بعد المحدد :public في جعل هذه الأعضاء والدوال عامة (أي يمكن الوصول لها ومتاحة للاستخدام من أي نقطة ضمن البرنامج)
- أما الأعضاء البيانية والدوال الأعضاء المصرح عنها بصنف ما بعد المحدد :private يمكن الوصول إليها وتكون متاحة فقط للاستخدام بواسطة الدوال الأعضاء المرتبطة بالصنف أو الدوال الأصدقاء.
- الأعضاء البيانية والدوال الأعضاء المصرح عنها بصنف ما بعد المحدد :protected يمكن الوصول إليها وتكون متاحة فقط للاستخدام بواسطة الدوال الأعضاء المرتبطة بالصنف أو عن طريق صنف مشتق (دالة معرفة في صنف مشتق).
- توفر الكلمات الرئيسية الخاصة (private) والمحمية (protected) مستوى حماية الوصول لإخفاء البيانات والدوال داخل الصنف.
- تشير هذه المحددات إلى رؤية الأعضاء حيث تكون الخصوصية (private) أكثر تقييدًا من المحمية (protected)
- إذا أخذنا كمثال الكود التالي

Access Modifiers (محددات الوصول)

- إذا أخذنا كمثال الكود التالي

```
class Patient {
    private:
        int patientNumber;
        string diagnosis;

    public:
      void billing() {
          // code
      }
      void makeAppointment() {
          // code
      }
};
```

- ✓ يشتمل الكود على Class definition يتضمن المتغيرين (patientNumber و diagnosis) الذين تم إخفاؤهما بواسطة المحدد private
- ✓ كذلك يشتمل الـdefinition على الدالة makeAppoinment تم تعريفها لتستدعى من أي مكان بالبرنامج بإستخدام المحدد public

Access Modifiers (محددات الوصول)

- البرنامج التالي للـC++ يتضمن Class definition لها عضوين. هذا وقد تم الوصول الى العضوين من داخل الدالة main() وذلك بفضل استخدام المحدد public

```
#include <iostream>
using namespace std;
// define a class
class Sample {

    // public elements
   public:
    int age;

    void displayAge() {
        cout << "Age = " << age << endl;
    }
};
```

```
int main() {

    // declare a class object
    Sample obj1;

    cout << "Enter your age: ";

    // store input in age of the obj1 object
    cin >> obj1.age;

    // call class function
    obj1.displayAge();

    return 0;
}
```

Output:

Enter your age: 20

Age = 20

Access Modifiers (محددات الوصول)

- تم استخدام المحدد private في البرنامج التالي لتقييد المتغير age ليكون اليه فقط من داخل الـClass definition التي أعلن فيها المتغير

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

// define a class
class Sample {

    // private elements
   private:
    int age;

    // public elements
   public:
    void displayAge(int a) {
        age = a;
        cout << "Age = " << age << endl;
    }
};
```

```
int main() {

    int ageInput;

    // declare an object
    Sample obj1;

    cout << "Enter your age: ";
    cin >> ageInput;

    // call function and pass ageInput as
    argument
    obj1.displayAge(ageInput);

    return 0;
}
```

Output:

Enter your age: 20

Age = 20

Access Modifiers (محددات الوصول)

- يُعدُّ استخدام الكلمة المفتاحيّة protected مفيدًا لقصر حق الوصول إلى بعض الMembers على الأصناف المشتقّة. فمثلًا في برنامج الـC++ التالي يكون الوصول إلى العضو age مقصورًا على الأصناف المشتقّة من الصنف sample

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

// declare parent class
class Sample {
    // protected elements
   protected:
    int age;
};

// declare child class
class SampleChild : public Sample {

   public:
    void displayAge(int a) {
        age = a;
        cout << "Age = " << age << endl;
    }

};
```

```
int main() {
    int ageInput;

    // declare object of child class
    SampleChild child;

    cout << "Enter your age: ";
    cin >> ageInput;

    // call child class function
    // pass ageInput as argument
    child.displayAge(ageInput);

    return 0;
}
```

Output:

Enter your age: 20

Age = 20

Member Functions

- التاابعة العضو (Member function) هي تلك الدالة التي يكون تعريفها (Function definition) أو راسها (Prototype/Function definition) مضمن بداخل تعريف الصنف (Class definition)
- توفر الـMember functions واجهة بينية عامة للبيانات الأعضاء الخاصة للصنف.
- عليه فإن الدالة التابعة يمكن تضمين تعريفها باداخل أو خارج الصنف (Class) التي تنتمي لها.
- كمثال أنظر الكود التالي يشتمل على Class definition يتضمن رأس للدالة **`getVolume`**

```
class Box {
   public:
      double length;          // Length of a box
      double breadth;         // Breadth of a box
      double height;          // Height of a box
      double getVolume(void);// Returns box volume
};
```

Member Functions

Definition of Member Functions

- هنالك طريقتين لتعريف الـMember functionهما:
  1. تعريف الدالة بداخل الـClass definition
  2. تعريف الدالة في خارج الـClass definition
- بداخل الـClass definition يتم تعريف تعريف الدالة بتضمينه مباشرةً في الـDefinition
- فيما يلي مثال لكود لـCass definition تم بداخله تضمين تعريف الدالة `getVolume`

```
class Box {
   public:
      double length;      // Length of a box
      double breadth;     // Breadth of a box
      double height;      // Height of a box

      double getVolume(void) {
         return length * breadth * height;
      }
};
```

Member Functions

Definition of Member Functions

- عند تعريف الـMember function بخرج الـClass definition، يتم إستخدام تسمية يظهر فيها إسم الـClass التي صرحت فيها الدالة.
- تستخدم هذه التسمية المعامل "::". ويعرف هذا المعامل **بمعامل تمييز المدى** (**scope resolution operator**)
- يحدد هذا المعامل إسم الصنف الذي تتبع له الـFunction وذلك كما هو مبين في الـCode example التالي

```
double Box::getVolume(void) {
   return length * breadth * height;
}
```

- وبنفس الطريقة المتبعة في الأعضاء البيانية، تستخدم النقطة (.)، والتي تسمى بمعامل الوصول (dot operator)، كنمط لإستدعاي الـMember functions
- فيا يلي Code example لإنشاء كائن وإستدعاء Member function تتبع لصنفه

```
Box myBox;           // Create an object

myBox.getVolume();   // Call member function for the object
```

Member Functions

Definition of Member Functions

- الرسم التالي يشتمل على إيضاحات لتعريف الـMember function بخارج تعريف الـClassالتي تتبع له

```
class Rectangle
{
    private:
      int width, length;
    public:
      void set (int w, int l);
      int area() {return width*length; }
};
```

inline

```
void Rectangle :: set (int w, int l)
{
    width = w;
    length = l;
}
```

class name

member function name

scope operator

- فيما يلي برنامج كامل لـC++ يشتمل تعريفات لمجموعة من الـMember functions

Member Functions

Definition of Member Functions

```
#include <iostream>

using namespace std;

class Box {
   public:
      double length;         // Length of a box
      double breadth;        // Breadth of a box
      double height;         // Height of a box

      // Member functions declaration
      double getVolume(void);
      void setLength( double len );
      void setBreadth( double bre );
      void setHeight( double hei );
};

// Member functions definitions
double Box::getVolume(void) {
   return length * breadth * height;
}

void Box::setLength( double len ) {
   length = len;
}
void Box::setBreadth( double bre ) {
   breadth = bre;
}
void Box::setHeight( double hei ) {
   height = hei;
}
```

```
// Main function for the program

int main() {
   Box Box1;                // Declare Box1 of type Box
   Box Box2;                // Declare Box2 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.setLength(6.0);
   Box1.setBreadth(7.0);
   Box1.setHeight(5.0);

   // box 2 specification
   Box2.setLength(12.0);
   Box2.setBreadth(13.0);
   Box2.setHeight(10.0);

   // volume of box 1
   volume = Box1.getVolume();
   cout << "Volume of Box1 : " << volume <<endl;

   // volume of box 2
   volume = Box2.getVolume();
   cout << "Volume of Box2 : " << volume <<endl;
   return 0;
}
```

Output:

Volume of Box1 : 210

Volume of Box2 : 1560

Member Functions

Types of Member Functions

- مميزات تعريف الدوال في خارج الصنف:
  1) إضافة تعدد الأشكال للدوال المعرفة بداخل الـClass
  2) تطوير البرامج (مثل حزم برامج Office التطويرية)
- تشتمل الـMember functions على بعض الأنواع الخاصة، وبناءاً على ذلك يتم تصنيف الـ Member functionsكما يلي:
  1. Simple member functions (الدوال الأعضاء البسيطة)
  2. Static member functions (الدوال الأعضاء الساكنة)
  3. Const member functions (الدوال الأعضاء الثابتة)
  4. Inline member functions (الدوال الأعضاء المضمنة)
  5. Friend member functions (الدوال الأعضاء الصديقة)
- تعمل الـconst member functions فقط على إسترجاع أو قراءة أعضاء الكائنات تستدعى تلك الدوال، ولكنها لا تستطيع عمل أي تعديل أو تغيير في تلك الأعضاء
- يتم تعريف الـStatic member function بواسطة الكلمة الأساسية static. استدعاءات هذه الدالة تتم من دون استعمال كائن معين بل يشار إلى الدالة من خلال ربط اسمها باسم الصنف بواسطة عامل دقة المدى (::)
- يمكن لدالة ليست عضواً في صنف (Class) ما الوصول إلى الأعضاء الخاصة بذلك الصنف وذلك بجعل الدالة صديقة friend لدوال ذلك الصنف. لجعل دالة ما صديقة نكتب الإعلان عنها داخل الصنف مسبوقاً بالكلمة الأساسية friend
- الـSimple member function هي الـMember functions الأساسية، ولا يشتمل تعريف هذه الدوال على كلمات خاصة محجوزة (مثل الـstatic وغيرها)
- تُسمى الدوال المُعرّفة بالكلمة المفتاحية inline دوالًا مُضمّنة (inline functions)

Constructor and Destructor (البناء والهدم)

Constructor

- من أهم الأشياء التي عليك التفكير بها عند إنشاء كلاس جديد هي تسهيل طريقة إنشاء كائنات من هذا الكلاس لاحقاً، من هنا جاءت فكرة دالة البناء (Constructor) والتي هي عبارة عن دالة يتم إستدعائها أثناء إنشاء كائن من الكلاس لإعطاء قيم أولية للخصائص الموجودة فيه (تهيئة الكائن للإستخدام)
- يتم استدعاءهذه الدالة بشكل تلقائي عند إنشاء كائن object من الصنف.
- هذه الدالة تجعلك قادراً على تمرير قيم أولية للكائن مباشرةً في لحظة إنشائه.
- دالة البناء Constructor هي أول دالة يتم إستدعائها عند إنشاء أي كائن object من أي صنف.
- من أهم مهام هذه الدالة أنها تقوم بإعطاء قيم اولية لجميع المُتغيرات الموجودة في الكلاس.
- كل كلاس يتم إنشاؤه يحتوي على Constructor واحد على الأقل. و حتى إن لم تقم بتعريف أي Constructor، سيقوم المترجم بإنشاء واحد إفتراضي.
- القاعدة الأساسية عند تعريف Constructor هي أنه يجب أن يحمل نفس إسم الكلاس و يكون نوعه Public
- **خصائص دالة البناء (Constructor)**
  1. تقوم بإرجاع قيمة و لا حتى void
  2. إسمها يأتي نفس إسم الكلاس (فإذا كان إسم الكلاس Date فيكون إسمها أيضاً Date)
  3. يتم إستدعائها عند إنشاء أي كائن من الكلاس مباشرةً.
- في المثال التالي قمنا بتعريف كلاس إسمه Book ولم نقم بتعريف Constructor له.

```
class Book {
   }
```

Constructor and Destructor (البناء والهدم)

Constructor

- في المثال التالي قمنا بتعريف كلاس إسمه Book ولم نقم بتعريف Constructor له.

```
class Book {
   }
```

- سيقوم المترجم بإنشاء Constructor فارغ بشكل تلقائي عنا كالتالي.

```
class Book {
     public:
     // It is a default constructor which constructed by the compiler
     Book() {
     }

     }
```

- في الحقيقة أنه توجد هنالك ثلاثة أنواع من الـConstructor هي:

1) Default Constructor (دالة الهدم الإفتراضية)

   وهو ذلك الـConstructor الذي ليس له Parameters

2) Parameterized Constructor (دالة البناء التي تستخدم المعاملات)

   وهو الConstructor الذي يحتاج الى تمرير Parameters عند إستدعائه

3) Copy constructor (دالة الهدم الناسخة)

Constructor and Destructor (البناء والهدم)

Default, Parametrized, and Copy Constructors

- عند إستدعاء الـDefault constructor فإنك لا تحتاج فعل أي شيء حين تنشئ كائن من الكلاس ، ويكون ذلك كما هو مبين في المثال التالي

```
Book book;
```

- عند إستدعاء الـ Parametreized constructor فإنك مجبر على تمرير قيم عند إنشاء الكائن، ويكون ذلك كما هو مبين في الأمثلة التالية

```
Book book("C++ for beginners");
```

```
Book book("C++ for beginners", "Mhamad Harmush");
```

- إذا الكلاس يحتوي على Default constructor وكان يحتوي كذلك على Parameterized constructor فهنا تكون مخيّر على استدعاء الـConstructor الذي تريده (كيف يكون ذلك؟)
- المثال التالي يشتمل على C++ program يتضمن على تعريف لـDefault Constructor

Constructor and Destructor (البناء والهدم)

Default, Parametrized, and Copy Constructors

- المثال التالي يشتمل على C++ program يتضمن على تعريف لـ Default Constructor

```
#include <iostream>

using namespace std;

class creature
{
   private:
    int yearofBirth;
   public:
    creature()
    {
     cout<<"Constructor is
   called";
    }
};
```

```
// Main function for the program

int main()
{
   creature obj;
   getch();
   return 0;
}
```

Output:

Constructor is called

Constructor and Destructor (البناء والهدم)

Default, Parametrized, and Copy Constructors

- المثال التالي يشتمل على C++ program يتضمن على تعريف لـ Parametrized Constructor تم تعريفه في خارج الـClass

```
class Point{              // Declaration Point Class
  int x,y;                // Properties: x and y coordinates
  public:
   Point(int, int);     // Declaration of the constructor
   bool move(int, int); // A function to move points
   void print();        // to print coordinates on the screen
};
```

```
Point::Point(int x_first, int y_first) {
  cout << "Constructor is called..." << endl;
  if ( x_first < 0 )    // If the given value is negative
    x = 0;              // Assigns zero to x
  else
    x = x_first;
  if ( y_first < 0 )    // If the given value is negative
    y = 0;              // Assigns zero to x
  else
    y = y_first;
}
```

```
int main() {
  Point p1(20, 100), p2(-10, 45);   // Construct is called 2 times
  Point *pp = new Point(10, 50);    // Construct is called once
  Point p3;                         // ERROR! There is not a default constructor
}
```

Constructor and Destructor (البناء والهدم)

Default, Parametrized, and Copy Constructors

- كما هو مبين في المثال التالي، يمكن أن تأخذ الـParameters في الـParameterized parameters على قيم إفتراضية.

```
class Point{
  public:
    Point(int x_first = 0, int y_first = 0);
};

Point::Point(int x_first, int y_first) {
  if ( x_first < 0 )                    // If the given value is negative
    x = 0;                              // Assigns zero to x
  else x = x_first;
  if ( y_first < 0 )                    // If the given value is negative
    y = 0;                              // Assigns zero to x
  else y = y_first;
}
```

- الكود التالي يبين عملية إنشاء لكائنات يستصحبها إستدعاء إفتراضي لـParametrized constructors

```
Point p1(15,75);    // x=15, y=75
Point p2(100);      // x=100, y=0
```

- الكود التالي يبين عملية إنشاء لكائن تستصحبه إستدعاء إفتراضي لـParametrized constructor في صورة Default constructor

```
Point p3;    // x=0, y=0
```

Constructor and Destructor (البناء والهدم)

Default, Parametrized, and Copy Constructor

- الـ**Copy constructor** هي عبارة دالة بناء خاصة تستخدم لإنشاء دالة جديدة (New object) في شكل نسخة (Copy) لكائن (Object) آخر. مع ملاحظة أن كلا الكئنان هما من كلاس واحدة.
- يتم إستخدام الـCopy constructor فيما يلي:
  1) إعطاء قيم إبتدائية لكائن من كائن آخر له نفس الـClass
  2) نسخ كائن لتمريره في شكل argument الى دالة
  3) نسخ كائن لإسترجاعه من دالة
- تظهر الصورة العامة للـCopy constructor كما يلي:

```
classname (const classname &obj) {
   // body of constructor
}
```

– الـobj في هذه الصورة العامة هو عبارة عن مرجع (Reference) للكائن الذي تم إستخدامه في إنشاء الكائن المستنسخ

- الـCopy constructor نفسه يتم تصنيفه الى نوعين:
  1) Default Copy Constructor
  2) User-defined Copy Constructor

Constructor and Destructor (البناء والهدم)

Default, Parametrized, and Copy Constructor

- الـ**Default copy constructor** لايتم تعريفه من قبل المبرمج، إنما يقوم المترجم (Compiler) بتقديمه وإستدعائه تلقائياً عندما يتم نسخ قيم لكائن الى كائن آخر جديد.
- قيما يلي مثال لـC++ program يتضمن إستدعاء غير صريح لـDefault copy constructor

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

class A {
    int x, y;

public:
    A(int i, int j)
    {
        x = i;
        y = j;
    }
    int getX() { return x; }
    int getY() { return y; }
};

int main()
{
    A ob1(10, 46);
    A ob2 = ob1;
     // 1
    cout << "x = " << ob2.getX() << " y = " <<
     ob2.getY();
    return 0;
}
```

Constructor and Destructor (البناء والهدم)

Default, Parametrized, and Copy Constructor

- أما في حالة الـ**User-defined copy constructor** فيتم نسخ قيم معاملات الكائن (Object parameters) الى الكائن الجديد، ويتم تعريف بصورة صريحة الـcopy constructor بواسطة المبرمج.
- وتنجذ عملية النسخ هذه في شكل عبارات صريحة تضمن في تعريف الـCopy constructor
- قيما يلي مثال لـ C++ program يتضمن تعريف لـUser-defined copy constructor

```
#include <iostream>
using namespace std;
class Example
{
   public:
    int a;
    Example(int x)        // parameterized constructor
    {
      a=x;
    }
    Example(Example &ob)      // copy constructor
    {
        a = ob.a;
    }
};
int main()
{
  Example e1(36);      // Calling the parameterized constructor
  Example e2(e1);      //  Calling the copy constructor
  cout<<e2.a;
  return 0;
}
```

- فيما يلي مثال شامل لبرنامج حول التعامل مع الConstructor في الـC++

Constructor and Destructor (البناء والهدم)

Default, Parametrized, and Copy Constructor

```
#include <iostream>

using namespace std;

class Line {

   public:
    int getLength( void );
    Line( int len ); // simple constructor
    Line( const Line &obj); // copy constructor
    ~Line();                // destructor

   private:
    int *ptr;
};

//Member functions definitions including constructor

Line::Line(int len) {
   cout << "Normal constructor allocating ptr" <<
    endl;

   // allocate memory for the pointer;
   ptr = new int;
   *ptr = len;
}
```

```
Line::Line(const Line &obj) {
   cout << "Copy constructor allocating ptr." <<
    endl;
   ptr = new int;
   *ptr = *obj.ptr; // copy the value
}

Line::~Line(void) {
   cout << "Freeing memory!" << endl;
   delete ptr;
}

int Line::getLength( void ) {
   return *ptr;
}

void display(Line obj) {
   cout << "Length of line : " << obj.getLength()
    <<endl;
}

// Main function for the program
int main() {
   Line line(10);

   display(line);

   return 0;
}
```

Output:
Normal constructor allocating ptr
Copy constructor allocating ptr.
Length of line : 10
Freeing memory!
Freeing memory!

Constructor and Destructor (البناء والهدم)

Destructors

- دالة الهدم (Destructor) هي عبارة عن دالة تقوم بإزالة الكائنات من الذاكرة (إلغاء الكائنات من الذاكرة).
- خصائص دالة الهدم:
  1) لها نفس اسم الصنف
  2) يسبق اسم دالة الهدم العLامة (~)
  3) لا توجد لها مدخلات (Parameters) ولا تقوم بإرجاع بشيء
  4) تقوم بإلغاء المساحات المحجوزة للكائنات في الذاكرة
  5) لا يمكن إنشاء أكثر من دالة هدم واحدة
  6) يتم تنفيذها عند إلغاء الكائن أو انتهاء البرنامج
- المدمرات مهمة جدا وتوجد في كل لغة برمجية دوال مدمرة، يمكنك ملاحظة دالة التدمير الخاصة ببيئة C++ عند ظهور الرسالة " Press any key to continue" في نهاية تنفيذ البرنامج.
  - ✓ تقوم بيئة C ++ بتدمير الكائن مباشرةً عند انتهاء البرنامج ولكن إذا أردنا التحكم بحذف الكائن فإننا يجب أن ننشئ دالة هدم خاصة بنا.
  - ✓ عند إغلاق البرنامج يجب حذف جميع المتغيرات التي يستخدمها من الذاكرة، وإلا فإنها ستبقى حتى تملأها وبالتالي سنحتاج إلى إضافة ذاكرة إضافية والتي بدورها ستمتلئ، يمكنك أن تلاحظ أنه عند إعطاء المتغير x قيمة معينة ثم إعادة تشغيل البرنامج وطباعة x فإنها لا تعيد شيئاً.
  - ✓ تقوم دالة الهدم الخاصة بنا بتدمير الكائن قبل دالة الهدم الخاصة بالبيئة
- فيما يلي Code example يشتمل على تعريفات لصنف ودالة بناء (Constructor) ودالة هدم (Destructor) للصنف

Constructor and Destructor (البناء والهدم)

Destructors

```
class String{
  int size;                  // Length (number of chars) of the string
  char *contents;            // Contents of the string
  public:
    String(const char *);    // Constructor
    void print();            // An ordinary member function
    ~String();               // Destructor
};
```

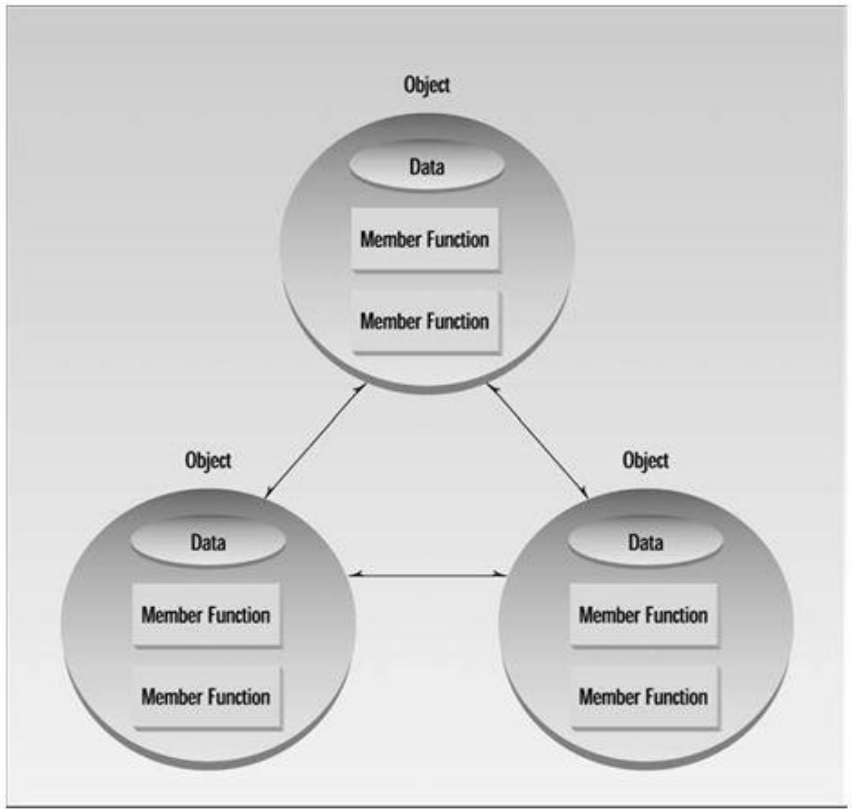
```
String::String(const char *in_data) {
  cout<< "Constructor has been invoked" << endl;
  size = strlen(in_data);                // strlen is a function of the cstring library
  contents = new char[size +1];          // +1 for null ( '\0' ) character
  strcpy(contents, in_data);             // input_data is copied to the contents
}

void String::print() {
  cout << contents << " " << size << endl;
}

// Destructor: Memory pointed by contents is given back
String::~String() {
  cout << "Destructor has been invoked" << endl;
  delete[] contents;
}
```

Communication between Objects (التواصل بين الكائنات)

- يتكون برنامج C++عادةً من عدد من الكائنات التي تتواصل communicate مع بعضها البعض عن طريق استدعاء الدوال الأعضاء المضمنة في هذه الكائنات. يوضح الشكل التالي تنظيم برنامج C++



- يشار إلى استدعاء دالة عضو لكائن بأنه إرسال رسالة (sending a message) إلى الكائن.
- يجب أن نذكر أن ما يسمى "دوال العضو" member functions في لغة C++ تسمى" الأساليب" methods في بعض اللغات الأخرى . ويشار إلى عناصر البيانات data itemsأيضًا على أنها سمات attributesأو متغيرات مثيل instance variables