

الدرس 1 : ما هو التصحيح ؟ (Lesson #01 - What Is Debugging & Why ?)

التصحيح هو: تتبع الكود سطر سطر لإيجاد الخطأ في الكود مع رؤية القيم والمتغيرات في الذاكرة
طريقة عمل التصحيح : تشغيل الكود على (Debugging mood) يكون قبل إنشاء ملف (File.exe) ، تستطيع تتبع الكود سطر سطر مع رؤية القيم والمتغيرات في الذاكرة

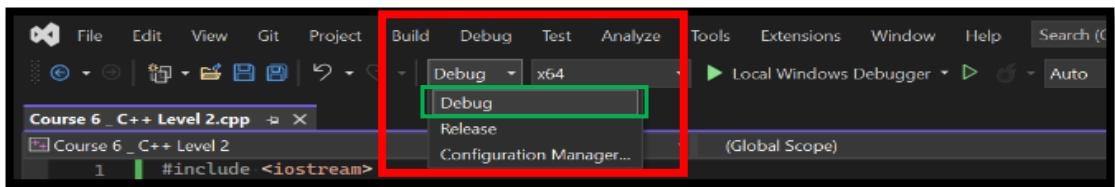
الأخطاء في البرمجة ثلاثة

١. أخطاء في قواعد الكتابة (Syntax error) وهو أسهل أنواع الأخطاء
٢. خطأ منطقي (Logical error) وهو أصعب أنواع الأخطاء ولهذا جاء التصحيح Debugging
٣. خطأ في وقت التشغيل (Run time error)

مميزات Debugging mood

- ❖ سرعة حل المشاكل
- ❖ تتبع الكود سطر سطر

الدرس 2 : نقاط التوقف والقيم في الذاكرة (Lesson #02 - Breakpoint & Memory Values)



بطيء في تشغيل البرنامج + يستخدمه مطورو البرامج **Debugging mood**

```

Course 6_C++ Level 2.cpp
1 #include <iostream>
2 using namespace std;
3 int MySum(int a, int b)
4 {
5     int s = 0;
6     s = a + b;
7     return s;
8 }
9
10 int main()
11 {
12     int arr1[5] = { 200,100,50,25,30 };
13     int a, b, c;
14
15     a = 10;
16     b = 20;
17     a++;
18     ++b;
19     c = a + b;
20     cout << a << endl;
21     cout << b << endl;
22     cout << c << endl;
23
24     for (int i = 1; i <= 5; i++)
25     {
26         cout << i << endl;
27         a = a + a * i;
28     }
29
30     c = MySum(a, b);
31
32     cout << c;
33
34     return 0;
35 }
```

Breakpoint

عندما تعمل Bilde يشتعل البرنامج الى أن يصل الى أمر التوقف ثم يتوقف ، ليكمل المطور عمله

عدم وضع **●** في سطر فاضي (سطر لا يوجد به كود)
إذا وضعتها في سطر فاضي لن يتم التوقف عند

```

Course 6_C++ Level 2.cpp
1 #include <iostream>
2 using namespace std;
3 int MySum(int a, int b)
4 {
5     int s = 0;
6     s = a + b;
7     return s;
8 }
9
10 int main()
11 {
12     int arr1[5] = { 200,100,50,25,30 };
13     int a, b, c;
14
15     a = 10;
16     b = 20;
17     a++;
18     ++b;
19     c = a + b;
20     cout << a << endl;
21     cout << b << endl;
22     cout << c << endl;
23
24     for (int i = 1; i <= 5; i++)
25     {
26         cout << i << endl;
27         a = a + a * i;
28     }
29
30     c = MySum(a, b);
31
32     cout << c;
33
34     return 0;
35 }
```

بعد أن يتوقف البرنامج عند **●** يبدأ المبرمج بتتبع الكود خطوة بخطوة **➡** (سهم لتتبع سطر الكود)

يدل على أن السهم متوقف عند الكود (لم يتم تنفيذه) **➡**
لتتنفيذ سطر الكود والنزول لسطر جديد اضغط على اختصار (F11)

عند وضع مؤشر الفأرة على اسم المتغير تظهر القيمة المخزنة فيه

إذا لم توضع قيمة مبدئية للمتغير يضع الكومبيوتر بيانات من عنده تكون غير مفيدة ، وقد ينتج عن ذلك مشاكل في البرنامج في وقت التشغيل **Run time error**

الدرس 3 : المزيد عن نقاط التوقف (Lesson #03 - More About Breakpoints)

```
#include <iostream>
using namespace std;
int MySum(int a, int b)
{
    int s = 0;
    s = a + b;
    return s;
}
int main()
{
    int arr1[5] = { 200,100,50,25,30 };
    int a, b, c;
    a = 10;
    b = 20;
    a++;
    ++b;
    c = a + b; // Breakpoint 1
    cout << a << endl;
    cout << b << endl;
    cout << c << endl;
    for (int i = 1; i <= 5; i++)
    {
        cout << i << endl;
        a = a + a * i;
    }
    c = MySum(a, b); // Breakpoint 2
    cout << c;
    return 0;
}
```

تستطيع إنشاء أكثر من واحدة في البرنامج

تستطيع الانتقال من الأولى إلى الثانية
 باستخدام اختصار التشغيل (F5)

تستطيع التنقل أيضا بإمساك السهم بالماوس
 ووضعه في أي سطر تريده

تستطيع إيقاف Debugging mood عن طريق
 الشريط العلوي

```
#include <iostream>
using namespace std;
int MySum(int a, int b)
{
    int s = 0;
    s = a + b;
    return s;
}
int main()
{
    int arr1[5] = { 200,100,50,25,30 };
    int a, b, c;
    a = 10;
    b = 20;
    a++;
    ++b;
    c = a + b;
    cout << a << endl;
    cout << b << endl;
    cout << c << endl;
    for (int i = 1; i <= 5; i++)
    {
        cout << i << endl;
        a = a + a * i;
    }
    c = MySum(a, b);
    cout << c;
    return 0;
}
```

تستطيع إلغاء عمل مع وجودها في البرنامج
 وتشغيل البرنامج كامل بدون التوقف عند

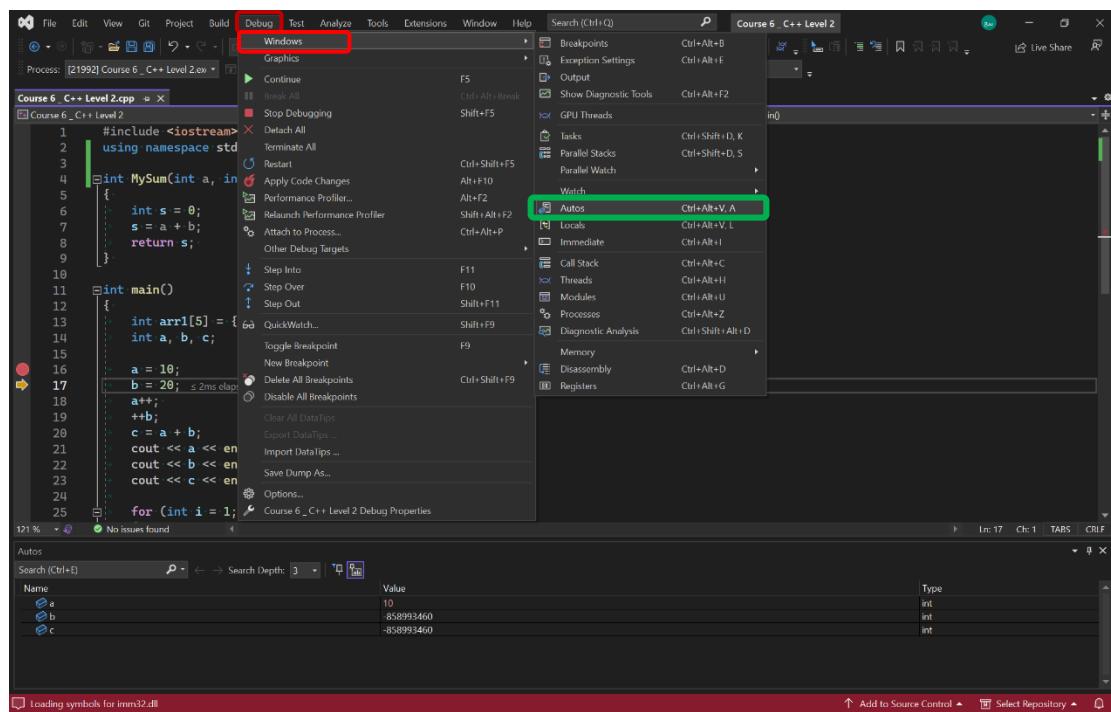
الدرس 4 : نافذة السيارات (Lesson #04 - Autos Window)

يظهر قيم المتغيرات تلقائيا في نافذة أسفل البرنامج ، بدون وضع مؤشر الفأرة على Autos المتغير

نوافذ السيارات قد تشتعل تلقائيا

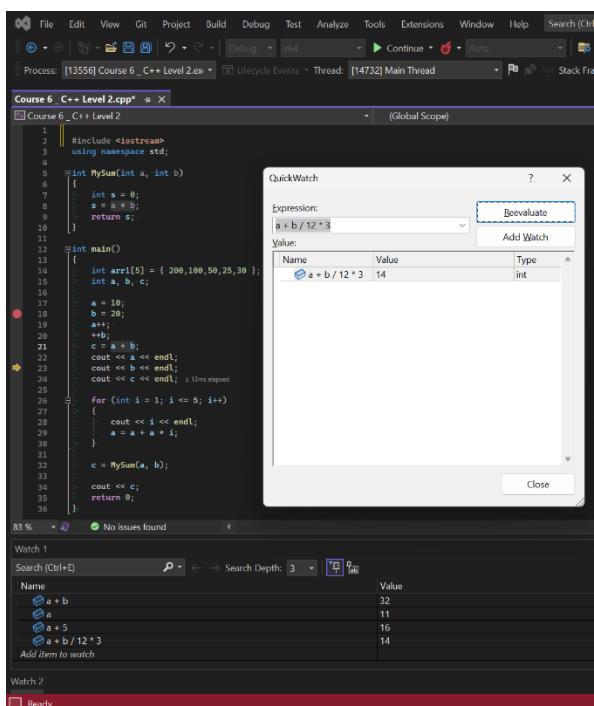
وإذا لم تشتعل تلقائيا اتبع الخطوات التالية

١. شغل البرنامج على Debugging mood
٢. اختر من الشريط العلوي (Debug)
٣. ثم اختر (Windows)
٤. ثم اختر (Autos)



الدروس 5 : نافذة المراقبة السريعة (Lesson #05 - Quick Watch Window)

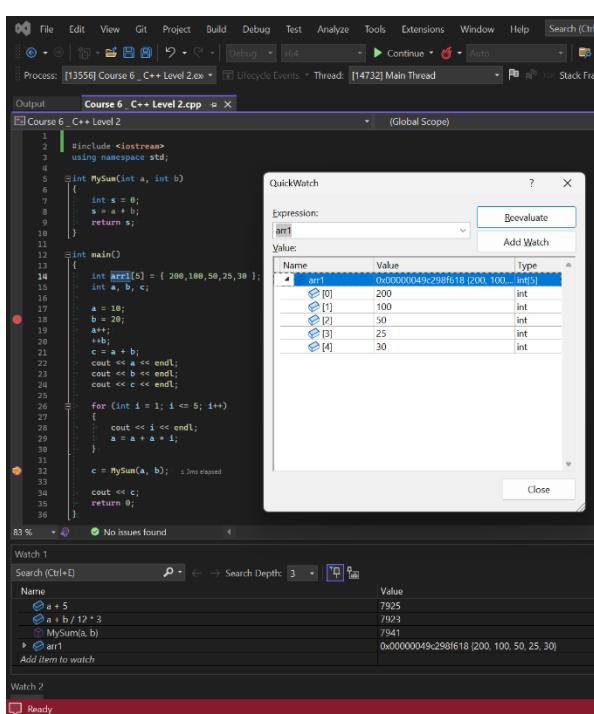
تخصيص متغير معين لمراقبته في البرنامج ، ويمكن إضافته لقائمة المشاهدة (من النافذة المنبثقة Add Watch) ومشاهدته في نافذة أسفل البرنامج)



لمراقبة متغير معين

١. شغل البرنامج على Debugging mood
٢. حدد على المتغير لمراقبته
٣. ثم اضغط على (Shift + F9)
٤. ستظهر نافذة (QuickWatch)
٥. لمراقبة المتغير في أسفل الشاشة اضغط على (Add Watch)

تستطيع إضافة معادلات في QuickWatch بدون مراقبتها في أسفل الشاشة باستخدام Reevaluate بعد كتابة المعادلة



تستطيع مراقبة كل شيء في QuickWatch سواء (Variable , Array , Function , Expression)

المتغير arr1 قيمته هي عنوان موقعه في الذاكرة (Function) : لو استدعيت arr1 في arr1 سيرسل Function فقط وليس نسخة من arr1 أي عند استدعاء arr1 في arr1 سيدهب نسخة الى عنوان arr1 وليس arr1 سترسل Pointers (ستدرس لاحقا في Function)

لمراقبة Function يتم التحديد على اسم Function مع الأقواس – وما بداخل الأقواس – MySum(a, b)

الدرس 6 : تغيير القيم في وضع التصحيح (Lesson #06 - Changing Values In Debugging Mode)

لتحريك قيمة المتغير أثناء التصحيح **Debugging mood** - بدون تحريك قيمته الأصلية في البرنامج -

The screenshot shows a C++ IDE interface with the following details:

- Code Editor:** Displays the file "Course 6_C++ Level 2.cpp" containing the following code:

```
16 a = 10;
17 b = 20;
18 a++;
19 ++b;
20 c = a + b;
21 cout << a << endl;
22 cout << b << endl;
23 cout << c << endl;
24
25
26 for (int i = 1; i <= 5; i++)
27 {
28     cout << i << endl;
29     a = a + a * i;
30 }
31
32 c = MySum(a, b);
33
34 cout << c;
35 return 0;
36 }
```
- Output Window:** Shows the output of the program: 10, 20, 30, 40, 50.
- Autos View:** Shows variable values:

Name	Value	Type
a	34	int
i	5	int
- Watch View:** Shows variable values:

Name	Value	Type
i	3	int

١. شغل البرنامج على **Debugging mood**

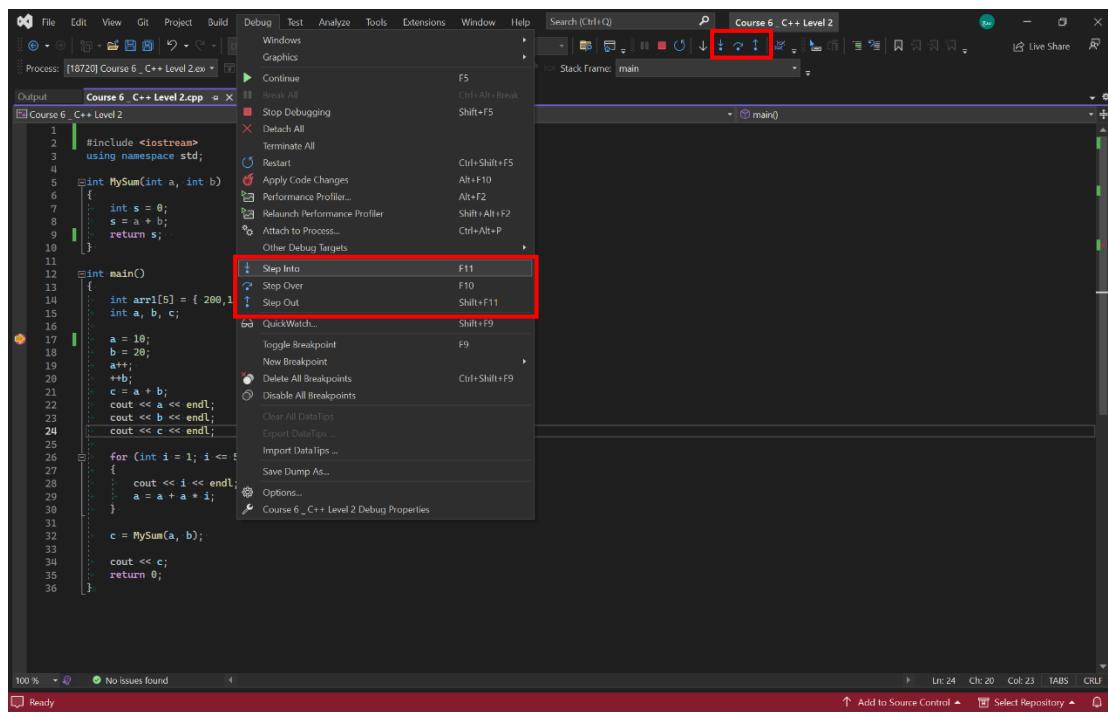
٢. غير قيمة المتغير باستخدام

- مؤشر الفأرة** : عند وضع مؤشر الفأرة على اسم المتغير تظهر القيمة المخزنة فيه اضغط مررتين على قيمة المتغير ثم ضع القيمة
- اضغط مررتين على قيمة المتغير **Autos** ثم ضع القيمة
- اضغط مررتين على قيمة المتغير **QuickWatch** ثم ضع القيمة

الدرس 7 : خطوة الى : أسفل / خروج / تخطي (Lesson #07 - Step Into/Over/Out)

أزرار التنقل في Debugging mood

١. (F11) للنزول الى سطر سطر (Step Into)
٢. (F10) لتخطي الدخول في Function (Step Over)
٣. (Shift + F11) للخروج من داخل Function (Step Out)



في أي البرنامج تستخدم Function & Procedures قد تعيد استخدامها في نفس لبرنامج أو في برنامج آخر ،
لذا وجدت المكتبات : لاختصار كتابة الكود - في المشروع - ، ضعها في مكتبة واستدعائها في وقت الحاجة

بعض المكتبات الموجودة في C++

- #include <iostream>
- #include <string>
- #include <cmath>

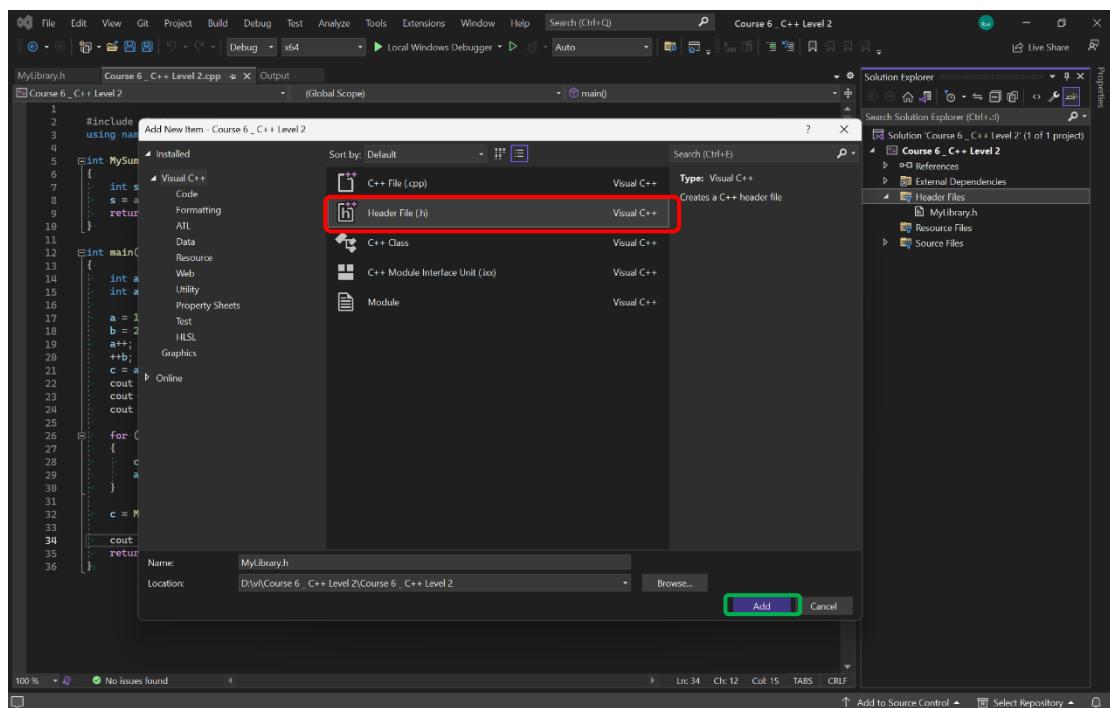
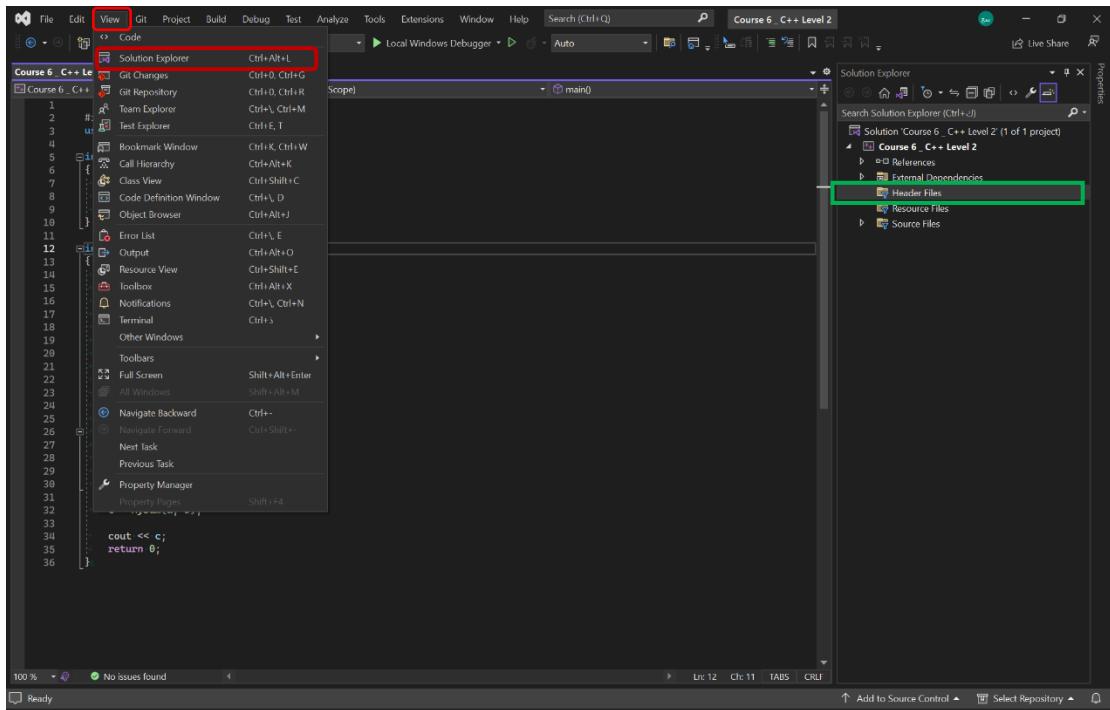
اجعل لكل مكتبة اسم يعبر عن محتواها

مميزات المكتبات

- تنظم الكود
- تختصر إعادة كتابة الكود
- تقلل عدد الأسطر

طريقة إنشاء مكتبة محلية – داخل مشروع - (Library Local)

١. اختر من الشريط العلوي (View)
٢. ثم اختر (Solution Explorer) ستظهر نافذة
٣. ثم اختر من النافذة (Header Files) زر الفأرة الأيمن
٤. ثم اختر (Add)
٥. ثم اختر (New Item..) ستظهر نافذة
٦. ثم اختر (Header File (.h))
٧. أنشئ اسم للمكتبة
٨. ثم اختر (Add) لإضافة المكتبة



في مكتبةك تستطيع إنشاء **namespace** ثم تحت **Function & Procedures** لكن تحت **namespace** أي اسم خاص بها ; ثم { **وضع فيها Function & Procedures**

تستطيع إستدعاء مكتبات - سواء موجودة في C++ أو أنت أنشأتها - داخل مكتبةك مثل
`#include <iostream>`

كيف تستطيع استدعاء مكتبةك في المشروع (الملف الرئيسي)

١. استدعاء مكتبةك `#include "MyLibrary.h"`
٢. داخل () استدعاء اسم `int main()`

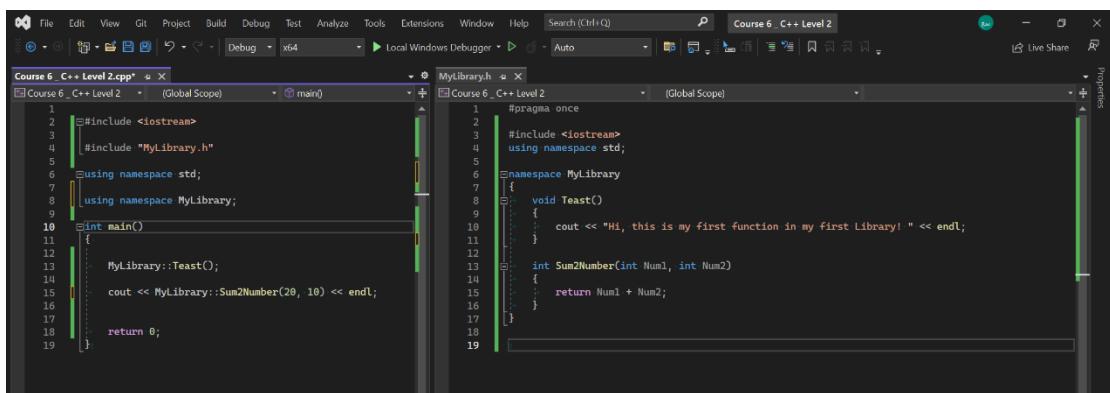
مثل
`MyLibrary::Teast();`

`cout << MyLibrary::Sum2Number(20, 10) << endl;`

تستطيع حذف `using namespace MyLibrary;` باستخدام `MyLibrary::`

لكن يفضل عدم استخدام هذا الاختصار

- قد يتتشابه الاسم في مكتبة أخرى فتحدث بعض الأخطاء
- سهولة قراءة الكود
- معرفة موقع **Function & Procedures** في أي مكتبة



```

Course 6_C++ Level 2.cpp*  (Global Scope)  main()
1 #include <iostream>
2
3 #include "MyLibrary.h"
4
5 using namespace std;
6
7 using namespace MyLibrary;
8
9
10 int main()
11 {
12     MyLibrary::Teast();
13     cout << MyLibrary::Sum2Number(20, 10) << endl;
14
15     return 0;
16 }
17
18
19

MyLibrary.h  (Global Scope)
1 #pragma once
2
3 #include <iostream>
4 using namespace std;
5
6 namespace MyLibrary
7 {
8     void Teast()
9     {
10         cout << "Hi, this is my first function in my first Library! " << endl;
11     }
12
13     int Sum2Number(int Num1, int Num2)
14     {
15         return Num1 + Num2;
16     }
17
18
19

```

الدرس 9 : اختصار الجمل الشرطية (Lesson #09 Ternary Operator: Short Hand If)

- اختصار كتابة الكود **Ternary Operator**
 - ليس كل شرط يعمل **Ternary Operator**
 - **If else Ternary Operator** يرجع نتيجة عكس الشرط + علامة الاستفهام ؟
 - إذا تحقق الشرط : & إذا لم يتحقق الشرط ; تستطيع عمل الآتي : مثال
- ```

Result = "PASS"; ○
cout << "PASS"; ○
Function & Procedures ○
إضافة شرط جديد (Mark >= 50) ? ○

```

### الطريقة القصيرة Syntax

| مثال                                                                                                                                           |         | الشرط                | إذا تحقق الشرط  |
|------------------------------------------------------------------------------------------------------------------------------------------------|---------|----------------------|-----------------|
| <code>int Mark = 90;<br/>string Result;<br/>Result = (Mark &gt;= 50) ? "PASS" :<br/>"FAIL";<br/>cout &lt;&lt; Result &lt;&lt;<br/>endl;</code> | ?       | ( إضافة شرط ) ؟      | { تنفيذ الشرط } |
|                                                                                                                                                | True :  | إذا تحقق الشرط :     | { تنفيذ الشرط } |
|                                                                                                                                                | False ; | إذا لم يتحقق الشرط ; | { تنفيذ الشرط } |

### الطريقة الطويلة Syntax

| مثال                                                                                                                                                                         |         | الشرط                                             | إذا تحقق الشرط  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|---------------------------------------------------|-----------------|
| <code>int Mark = 90;<br/>string Result;<br/>if (Mark &gt;= 50)<br/>{ Result = "PASS"; }<br/>else<br/>{ Result = "FAIL"; }<br/>cout &lt;&lt; Result &lt;&lt;<br/>endl;</code> | if      | ( إضافة شرط )                                     | { تنفيذ الشرط } |
|                                                                                                                                                                              | else if | ( إضافة شرط جديد ، إذا لم يتحقق الشرط الذي قبله ) | { تنفيذ الشرط } |
|                                                                                                                                                                              | else    | إذا لم يتحقق أي شرط                               | { تنفيذ الشرط } |

## # الدرس 10 : نطاق التكرار (Lesson #10 - Ranged Loop)

❖ ليست بديل من العادلة **for Loop** **Ranged Loop**

❖ نوع النطاق (int , bool , string ...) لابد أن يكون مثل نوع **Collection**

❖ **Ranged Loop** تستخدم مثلاً مع (Array , Victor , Object )

❖ يبدأ النطاق من أول عنصر {1} ثم العنصر الذي يليه {2} ... مثل

```
int Array1[] = { 1,2,3,4} ;{Set}
for (int n : Array1)
{ Cout << n << endl; }
```

( Array , Victor , Object ) تستخدم مع أشياء ديناميكية مثلاً ( Collection of things

### Syntax

| مثال                                                                                            |     | تسمية المتغير<br>وتحديد نوعه                                |   | مجموعة من الأمور<br>Collection |                         |
|-------------------------------------------------------------------------------------------------|-----|-------------------------------------------------------------|---|--------------------------------|-------------------------|
| <code>int Array1[] = { 1,2,3,4}; for (int n : Array1) { Cout &lt;&lt; n &lt;&lt; endl; }</code> | for | (Range Declaration                                          | : | Range Expression )             | { Code }                |
|                                                                                                 | for | ( تسمية النطاق لابد أن يكون نوع النطاق مثل نوع ( Collection | : | النطاق المراد السير فيه        | { تكرار تنفيذ الأوامر } |

## # الدرس 11 : التحقق من رقم فقط (Lesson #11 - Validate Number)

❖ التتحقق أن المدخل رقم فقط Validate  
❖ Function ( ستدرس لاحقا OOP ) هي cin وليس OOP  
❖ هل المدخل خطأ ؟ cin.fail()

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

int ReadNumber()
{
 int Number = 0;

 cout << "Please enter a number ? \n";
 cin >> Number;

 while (cin.fail())
 // يعنى هل المدخل (خطأ) ليس رقم ؟
 // نعم (خطأ) المدخل ليس رقم : أدخل في حلقة التكرار
 {
 // user didn't input a number
 cin.clear(); // تجاوز عن الخطأ المدخل

 // تخطى عن كل الأشياء المدخلة الى '\n'
 cin.ignore(std::numeric_limits<std::streamsize>::max(),
 '\n');

 cout << "Invalid Number , Enter a valid one : " << endl;
 }

 return Number;
}

int main()
{
 cout << "\n Your number is : " << ReadNumber() << endl;
}
```

## # الدرس 12 : أحدى المعامل & (Lesson #12 - Bitwise & Operator)

- ❖ أحدى المعامل **Bitwise** تستخدم إشارة واحدة فقط **&**
- ❖ في المنطق Logic AND تستخدم إشارتين من **&&**
- ❖ يتم تحويل الأرقام إلى **Binary** & **Bitwise**
- ❖ ثم يتم المقارنة بين كل Bit مع مقابلها (Logic AND **&&**)
- ❖ الناتج يتم تحويله إلى Decimal

| ( 25 & 12 ); |   |   |   |   |   | Decimal |
|--------------|---|---|---|---|---|---------|
| Binary       | 0 | 1 | 1 | 0 | 0 | 12      |
| Binary       | 1 | 1 | 0 | 0 | 1 | 25      |
| الناتج &     | 0 | 1 | 0 | 0 | 0 | 8       |

## # الدرس 13 : أحدى المعامل | (Lesson #13 - Bitwise | Operator)

- ❖ أحدى المعامل **Bitwise** تستخدم إشارة واحدة فقط |
- ❖ في المنطق Logic OR تستخدم إشارتين من **||**
- ❖ يتم تحويل الأرقام إلى **Binary** | **Bitwise**
- ❖ ثم يتم المقارنة بين كل Bit مع مقابلها (|)
- ❖ الناتج يتم تحويله إلى Decimal

| ( 25   12 ); |   |   |   |   |   | Decimal |
|--------------|---|---|---|---|---|---------|
| Binary       | 0 | 1 | 1 | 0 | 0 | 12      |
| Binary       | 1 | 1 | 0 | 0 | 1 | 25      |
| الناتج       | 1 | 1 | 1 | 0 | 1 | 29      |

## # الدرس 14 : Declaration Vs Definition (Lesson #14 - Declaration Vs Definition)

❖ يتم استخدامه عند مناداة Function أعلى ل Declaration Vs Definition ❖  
❖ أسفل Function

```
// Function declaration
// يكون عنوان التعريف سطر واحد
void add(int, int);

int main()
{
 استدعاء Function من الأسفل//
 add(10, 20);
}

// Function definition
// main() بعد Function تعریف
void add(int a, int b)
{
 cout << a + b << endl;
}
```

## # الدرس 15 : فشل البارامتر (اختياري) (Lesson #15 - Default Parameters)

❖ فشل البارامتر : وضع قيمة مبدئية في حال فشل البارامتر (يأخذ القيمة المعرفة = 0)  
❖ فشل البارامتر يعني بارامتر اختياري Option Parameter يعنى Default Parameters ❖

```
عند وضع قيمة في البارامتر (= 0) يصبح البارامتر اختياري
int MySum(int a, int b, int c = 0, int d = 0)
{
 return (a + b + c + d);
}

int main()
{
 // (10 + 20 + 0 + 0) = 30
 cout << MySum(10, 20) << endl;

 // (10 + 20 + 30 + 0) = 60
 cout << MySum(10, 20, 30) << endl;

 // (10 + 20 + 30 + 40) = 100
 cout << MySum(10, 20, 30, 40) << endl;
}
```

## # الدرس 16 : وظيفة التحميل الزائد ( Lesson #16 - Function Overloading )

- ❖ مجموعة من Functions تحت اسم واحد تجمعهم علاقة (الجمع)
- ❖ كل Function لابد أن يختلف عن الآخر من (نوع ، عدد البارامتر)
- ❖ تكرار اسم Function لا يحدث خطأ في البرنامج عكس المتغيرات في (C++)
- ❖ Function Overloading هي Polymorphism تعدد أشكال

```
// اختلاف نوع البارامتر
double MySum(double a, double b)
{
 return (a + b);
}

// اختلاف نوع البارامتر
int MySum(int a, int b)
{
 return (a + b);
}

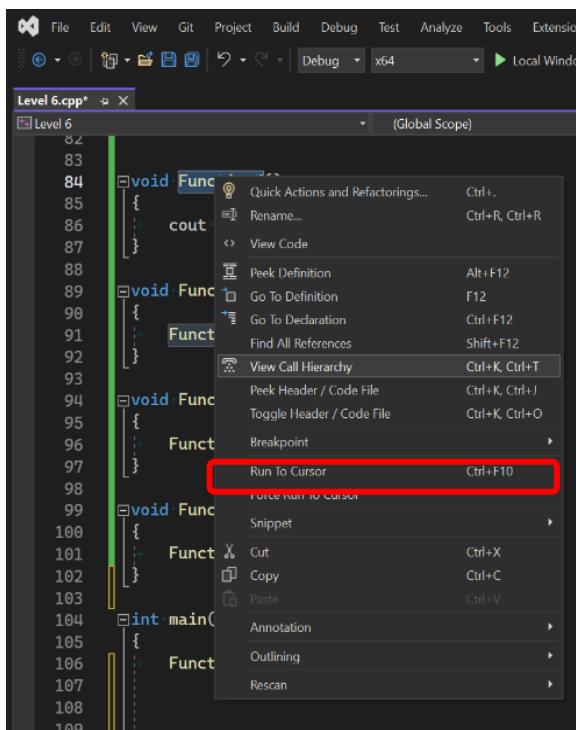
// اختلاف عدد البارامتر
int MySum(int a, int b , int c)
{
 return (a + b + c);
}

// اختلاف عدد البارامتر
int MySum(int a, int b, int c , int d)
{
 return (a + b + c + d);
}

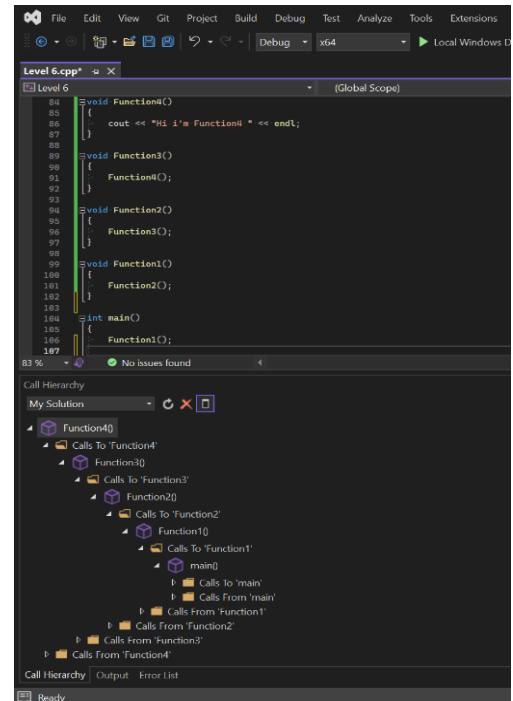
int main()
{
 cout << MySum(10, 20) << endl;
 cout << MySum(10.1, 20.2) << endl;
 cout << MySum(10, 20 , 30) << endl;
 cout << MySum(10, 20 , 30 , 40) << endl;
}
```

## # الدروس 17 : تدرج الاستدعاء ( Lesson #17 - Call Stack / Call Hierarchy )

- ❖ **Call Stack OR Call Hierarchy** هو نفس الشيء (اختلاف أسماء فقط والفعل واحد)
- ❖ سلسلة مناداة Function لـ Function **Call Stack / Call Hierarchy**
- ❖ يصل لآخر Function يتم استدعاؤه ( تدرج استدعاء Function في البرنامج )
- ❖ يمثل الإطار النشط في الذاكرة **Active Frame**
- ❖ يكون واحد فقط نشط **Active Frame**
- ❖ أول Function يتم استدعاؤه في البرنامج هو **int main()** وآخر Function إزالتة
- ❖ أول عنصر يدخل في Stack هو آخر عنصر يخرج من Stack ( والعكس بالعكس )
- ❖ إضافة عنصر إلى Stack يسمى **Push** ويكون هو Active Frame
- ❖ إزالة عنصر من Stack يسمى **Pop** ويكون الذي قبله هو Active Frame



١. حدد Function
٢. اضغط على زر الفأرة الأيمن فتظهر نافذة
٣. اختر (View Call Hierarchy)
٤. ستظهر نافذة (Call Hierarchy)



٥. لمعرفة Function الذي استدعاه اضغط على Function بجانب اسم Function
٦. ثم اضغط على ( Call To ) بجانب اسم " Function "

## بعض ( اختصارات Visual Studio )

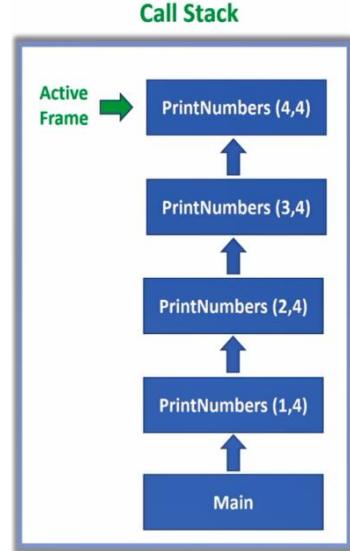
١. تصغير Function بجانب نوع Function توجد إشارة ناقص —
٢. للذهاب الى **Function Definition** ( F12 )
٣. للذهاب الى **Function Declaration** ( Ctrl + F12 )
٤. البحث عن Function مع جميع استدعاءاته ( Shift + F12 ) Find All References
٥. نظرة خاطفة ل Function مع عدم الذهاب إليه ( Alt + F12 ) Peek Definition
٦. لتغيير اسم Function مع جميع استدعاءاته دفعه واحدة ( Ctrl + R ) Rename...

- ❖ هو استدعاء Function نفسه داخل Function **Recursion**
- ❖ كل Function يتم استدعاؤه يتم حجز مكان له في **Call Stack** ( ولو استدعي نفسه Function )
- ❖ له حد أقصى للتخزين في الذاكرة ( إذا تجاوز الحد البرنامج لا يشتعل )
- ❖ لا يستحسن استخدامه إذا توفر البديل Loop ( for , while , do ) **Recursion**
- ❖ يستخدم إذا كنت تعلم أنه لن يتم تجاوز الحد الأقصى للتخزين **Recursion**
- ❖ في Python له مساحة كبيرة ، أما في C++ له مساحة أصغر من Call Stack

```
void PrintNumbers(int N, int M)
{
 if (N <= M)
 {
 cout << N << endl;
 // Print (1 , 2 , 3 , 4)

 // استدعاء نفسه Function
 PrintNumbers(N + 1, M);
 // 1 + 1 = 2 | 2 + 1 = 3 | 3 + 1 = 4 ;
 }
}

int main()
{
 PrintNumbers(1 , 4);
}
```



## # الدرس 20 : تثبيت المتغير (Lesson #20 - Static Variables)

- ❖ دورة حياة المتغير Variable العادي في Function هي بانتهاء Function ( بمجرد الانتهاء من Function يدمر نطاق Variable تلقائيا – نطاق محلي – )
- ❖ عند استدعاء Function نفسه مرة أخرى تكون قيمة المتغير هي نفسها (مثل قيمة Function الأولى )
- ❖ تثبيت قيمة المتغير بعد الخروج من Function (حياته بانتهاء كامل البرنامج)
- ❖ متغير يتم الاحتفاظ بقيمة في البرنامج بأكمله Static

```
void MyFunction()
{
 // Function تنتهي حياة المتغير بانتهاء
 // يبدأ من جديد مع كل استدعاء وهي القيمة 1

 int Number = 1;

 cout << "Value of Number : " << Number << endl;

 Number++;
}

void MyFunctionStatic()
{
 //Function لا تنتهي حياة المتغير مع Static بانتهاء
 //يحافظ على القيمة السابقة لاستدعائه

 static int Number = 1; // 2 | 3 | 4

 cout << "Static Variable of Number : " << Number << endl;

 Number++; // 2 | 3 | 4
}

int main()
{
 MyFunction(); // Print 1
 MyFunction(); // Print 1
 MyFunction(); // Print 1

 MyFunctionStatic(); // Print 1
 MyFunctionStatic(); // Print 2
 MyFunctionStatic(); // Print 3
}
```

## # الدرس 21 : المتغير التلقائي (Lesson #21 - Automatic Variables)

- ❖ الأفضل في تعريف أنواع المتغيرات هي استخدام نوع المتغير المراد استخدامه مثل ( int , string , float .... ) يكون أسرع للبرنامج
- ❖ الأفضل عدم استخدام المتغير التلقائي **Automatic Variables**

```
int main()
{
 auto a = 10; // Type Integer
 auto y = 12.5; // Type Double
 auto z = "Mohammed Abu-Hadhoud"; // Type String

 cout << a << endl;
 cout << y << endl;
 cout << z << endl;
}
```

## # الدرس 22 : Register Variable (Lesson #22 - Register Variable)

### تم إلغاؤه من C++ 11 فما فوق

- ❖ المتغيرات يتم تخزينها بشكل تلقائي في RAM
- ❖ كانت تستخدم مع السرعات العالية جداً جداً

أسرع وحدات التخزين من حيث تلقي الأوامر من CPU بالترتيب

- ١.Registers البيانات المتأتية على الفور (أقرب وحدة لCPU)
٢. Cache Memory الوصول الى البيانات بشكل مماثل أبطأ من الأول
٣. RAM Primary Memory الوصول الى البيانات بشكل أسرع أبطأ من الثاني
٤. Hard disk Primary Memory (الذاكرة الدائمة) الوصول الى البيانات بشكل بطيء

## # الدرس 23 : تنسيق طباعة الأرقام (Printf) (Lesson #23 - Integer Format (Printf))

❖ يتم استبدال كل **%d** في string بالمتغير رقم صحيح **Variable int** بعد الفاصلة ،  
(بالترتيب ) (أول **%d** = مع أول متغير بعد الفاصلة ... )

```
int main()
{
 int Page = 1, TotalPages = 10;

 // print string and int Variable
 printf("The page number = %d \n", Page);
 printf("You are in page %d of %d \n", Page , TotalPages);

 // Width Specification
 // مواصفات العرض

 // 2 = Two Digit
 // يوجد متغير واحد (رقم) عوض عن المتغير (رقم) الثاني ب 0 صفر
 // %0*d, 2
 printf("The page number = %0*d \n", 2, Page); // Print 01
 printf("The page number = %0*d \n", 3, Page); // Print 001
 printf("The page number = %0*d \n", 4, Page); // Print 0001
 printf("The page number = %0*d \n", 5, Page); // Print 00001

 int Number1 = 20, Number2 = 30;
 printf("The Result of %d + %d = %d \n", Number1, Number2 , Number1 +
Number2);
}
```

## موجود على أغلب اللغات Format

- ❖ **%.3f** ، إظهار أول 3 أرقام من الخانات العشرية بعد الفاصلة .
- ❖ مع تقرير الرقم الأخير بعد الفاصلة 3 أرقام فقط **f3.%**
- ❖ يتم استبدال 3 بمتغير **Variable** (PI) كامل ، يتم التعديل على الشاشة فقط

```

int main()
{
 float PI = 3.14159265;

 // Precision Specification
 // مواصفات الدقة
 // %.3f,3 Round
 // %*c 2 , 1
 // (*) هي لإظهار عدد الأرقام بعد الفاصلة على حسب الأعداد
 printf("Precision Specification of %.3f \n", 1, PI);
 printf("Precision Specification of %.3f \n", 2, PI);
 printf("Precision Specification of %.3f \n", 3, PI);
 // Print (3.142 تم تقرير 1 الى 2 لأن الرقم الذي يعده 5)
 printf("Precision Specification of %.3f \n", 4, PI);

 // %.3f Round
 float x = 7.0439, y = 9.0;
 printf("\n The float division is : %.3f / %.3f = %.3f \n\n", x, y,
x / y);
 // 7.044 / 9.000 = 0.783

 double d = 12.45;
 printf("The double value is : %.3f \n", d);
 printf("The double value is : %.4f \n", d);
 // يتم تعويض الخانات الناقصة بأصفار
}

```

## Char Array الى String ، يتم تحويل لا يتعامل مع String ♦

```

int main()
{
 // string لا يتعامل مع printf
 // char Array الى string تحويل

 char Name[] = "Mohammed Abu-Hadhoud";
 char SchoolName[] = "Programming Advices";

 printf("Dear %s, How are you ? \n\n", Name);
 printf("Welcome to %s school \n\n", SchoolName);

 char c = 'S';

 // %*c هي للفراغات على حسب الأرقام 1 2 (*)
 printf("Setting the width of c :%*c \n", 1, c);
 printf("Setting the width of c :%*c \n", 2, c);
 printf("Setting the width of c :%*c \n", 3, c);
 printf("Setting the width of c :%*c \n", 4, c);
 printf("Setting the width of c :%*c \n", 5, c);
}

```

| printf ( " % " );                                   |                                  |
|-----------------------------------------------------|----------------------------------|
| printf (" int and<br>short = %d OR<br>%0*d ");      | <b>int<br/>short</b>             |
| printf (" float<br>and double =<br>%.*f OR %.5f "); | <b>float<br/>double</b>          |
| printf (" char<br>array = %s ");                    | <b>char array<br/>( string )</b> |
| printf (" char<br>array = %*c ");                   | <b>char</b>                      |

```

#include <iostream>
#include <iomanip> // this library stored the std::setw
// هذه المكتبة لجز الفراغات
using namespace std;

int main()
{
 cout << "-----|-----|-----| " <<
endl; cout << " Code | Name | Mark | " <<
endl; cout << "-----|-----|-----| " <<
endl;

 // #include <iomanip> استدعاء مكتبة
 // setw(9) (حجز عدد المسافات - لتخزين الاسم فيها - لا ينطوى الاسم هذا العدد)
 cout << setw(9) << "C101" << "|" << setw(32) << "introduction to
programming 1" << "|" << setw(9) << "95" << "|" << endl;
 cout << setw(9) << "C102" << "|" << setw(32) << "Computer Hardware"
<< "|" << setw(9) << "88" << "|" << endl;
 cout << setw(9) << "C1035243" << "|" << setw(32) << "Network" << "|
<< setw(9) << "75" << "|" << endl;

 cout << "-----|-----|-----| " <<
endl;
}

```

### الناتج على الشاشة

| Code     | Name                          | Mark |
|----------|-------------------------------|------|
| C101     | introduction to programming 1 | 95   |
| C101     | Computer Hardware             | 88   |
| C1035243 | Network                       | 75   |

الأول [للسقوف] الثاني [للأعمدة] **Two Dimensional Arrays** ♦♦♦

```
int main()
{
 // int x[Rows][Cols]
 int x[3][4] =
 {
 {1,2,3,4},
 {5,6,7,8},
 {9,10,11,12}
 };

 // index = 0
 for (int i = 0; i < 3; i++)
 {
 for (int y = 0; y < 4; y++)
 {
 cout << x[i][y] << " ";
 // [0][0] = 1 | [0][1] = 2 .. | [2][3] = 12
 }
 cout << endl;
 }
}
```

- هي عبارة عن **Array** لكن ديناميكية - تحجز المساحة المناسبة لحجمها ♦  
 تستطيع إضافة العناصر في – Run Time - وإزالتها
- ♦ تحدد حجمها قبل استخدامها - لو تم حجز **Array[100]** واستخدمت 5 خمس من 100 ستضيف مساحة (95) من الذاكرة على الفاضي == البرنامج سيصبح بطيء
- ♦ في C++ تستطيع أن تجعل مساحة **Array** ديناميكية - باستخدام **Pointer**
- ♦ عند استخدام **#include <vector>** استدعى مكتبة **Vectors**

### Syntax

| <b>Vectors</b> | <b>&lt; Type &gt;</b> | <b>Name</b> | <b>Initial Value</b> |
|----------------|-----------------------|-------------|----------------------|
| vector         | < int >               | vNumbers    | = { 10 , 20 };       |

```
#include <iostream>
#include <vector> // مكتبة - ديناميكيا المخزنة القيم حسب على الذاكرة في مساحة لحجز
using namespace std;

int main()
{
 vector <int> vNumbers = { 10, 20, 30, 40, 50 };
 cout << "Numbers vector = ";

 // Ranged Loop Vector هي
 // vector <int> لا بد أن يكون مثل int Number
 // ينسخ العنصر الأول من vector إلى int Number : vNumbers
 // عملية النسخ تأخذ وقت وحجم = برنامج بطيء
 for (int & Number : vNumbers)
 {
 // إشارة & ، عملها الذهاب إلى موقع العنصر في الذاكرة
 // إشارة & : عدم نسخ العناصر//
 cout << Number << " ";
 }

 cout << endl;
}
```

- ❖ في **Vector** يوجد اسمه **push\_back** لإضافة العناصر
- ❖ عند إضافة عنصر ل **Stack** يتم إضافتها في **Stack**
- ❖ **Stack** يستخدم نوع من أنواع **Vector** وهي **Data Structure**
- ❖ أول عنصر يدخل في **Stack** هو آخر عنصر يخرج من **Stack** ( والعكس بالعكس )
- ❖ إضافة عنصر إلى **Stack** يسمى **Push**
- ❖ إزالة عنصر من **Stack** يسمى **Pop**
- ❖ في **Array** هي **Parameters** By Reference & By Value
- ❖ في **Vector** هي **Parameters** By Reference & By Value
- ❖ عند إنشاء **Vector** من نوع **Parameters** يفضل دائماً وضع إشارة &
- ❖ ( للتعديل على **Vector** الرئيسي )
- ❖ يفضل دائماً وضع إشارة & في **Address** ( للذهاب إلى **Ranged Loop** ) وعدم نسخه

```
#include <iostream>
#include <vector> // مكتبة - ديناميكيا المخزنة القيم حسب على الذاكرة في مساحة لجز
using namespace std;

int main()
{
 vector <int> vNumbers ;
 لإضافة عناصر الى vector وأخذ مساحة له بشكل أوتوماتيكي //
 vNumbers.push_back(10);
 vNumbers.push_back(20);
 vNumbers.push_back(30);
 vNumbers.push_back(40);
 vNumbers.push_back(50);

 cout << "Numbers vector : \n\n";
 // Ranged Loop هي Vector من الطرق لطباعة
 // vector <int> لا بد أن يكون مثل int Number
 // ... int Number من vector ينسخ العنصر الأول من vector الى int Number : vNumbers
 // عملية النسخ تأخذ وقت وحجم = برنامج بطيء
 for (int & Number : vNumbers)
 {
 // إشارة & ، عملها الذهاب الى موقع العنصر في الذاكرة
 // الى عدم نسخ العناصر/
 cout << Number << endl;
 }
 cout << "\n\n";
}
```

## Homework Solution

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

عدم وضع إشارة & للباراميتير يعني //

عمل نسخة من Vector وتعينتها ثم بعد الخروج من Function يتم تدمير هذه النسخة -  
// 1- لا يتم تعديل أو تعينة على Vector الرئيسي الموجود في ()

```
void ReadNumber (vector <int> & vNumbers)
{
 char ReadMore = 'Y';
 int Number;

 while (ReadMore == 'y' || ReadMore == 'Y')
 {
 cout << "Pleas enter a Number ? ";
 cin >> Number;

 vNumbers.push_back(Number);

 cout << "\nDo you want to more numbers ? Y / N ? ";
 cin >> ReadMore;
 }
}
```

عند وضع إشارة & للباراميتير //

الذهاب للموقع - العنوان - الموجود في الذاكرة ( يتم نسخ العنوان فقط // 1- Address

```
void PrintVectorNumber(vector <int> & vNumbers)
{
 cout << "Numbers Vector : \n\n";

 // Ranged Loop
 for (int & Number : vNumbers)
 {
 cout << Number << endl;
 }
}

int main()
{
 vector <int> vNumbers;

 ReadNumber(vNumbers);
 PrintVectorNumber(vNumbers);
}
```

- ❖ عدم وضع إشارة **& Reference** : يتم إنشاء نسخة ثانية من المتغير
- ❖ النسخ = حجز مساحة أخرى لا داعي لها ، ووقت لعملية النسخ == برنامج بطيء
- ❖ وضع إشارة **& Reference** : يتم نسخ عنوان المتغير للتعديل على القيمة

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

struct stEmployee
{
 string FirstName;
 string LastName;
 int Salary;
};

int main()
{
 vector < stEmployee > vEmployee;

 stEmployee tempEmployee;

 // vEmployee ثم إضافتها إلى stEmployee تعبئته

 tempEmployee.FirstName = "Mohammed";
 tempEmployee.LastName = "Abu-Hadhoud";
 tempEmployee.Salary = 5000;
 vEmployee.push_back(tempEmployee);

 tempEmployee.FirstName = "Ali";
 tempEmployee.LastName = "Maher";
 tempEmployee.Salary = 300;
 vEmployee.push_back(tempEmployee);

 tempEmployee.FirstName = "Aya";
 tempEmployee.LastName = "Omran";
 tempEmployee.Salary = 1000;
 vEmployee.push_back(tempEmployee);

 cout << "Employees Vector : \n\n";

 // Ranged Loop
 // vEmployee : يتم إنشاء نسخة ثانية من
 // النسخ = حجز مساحة أخرى لا داعي لها ووقت لعملية النسخ == برنامج بطيء
 // وضع إشارة & : يتم نسخ عنوان vEmployee للذهاب إليه
 for (stEmployee & Employee : vEmployee)
 {
 cout << "FirstName : " << Employee.FirstName << endl;
 cout << "LastName : " << Employee.LastName << endl;
 cout << "Salary : " << Employee.Salary << endl;

 cout << endl;
 }
 cout << endl;
}
```

## # الدرس 31 : إزالة العناصر (Lesson # 31 - Remove elements)

- ❖ أول عنصر يدخل في **Stack** هو آخر عنصر يخرج من **Stack** ( والعكس بالعكس )
- ❖ إضافة عنصر الى **Stack** يسمى **Push**
- ❖ إزالة عنصر من **Stack** يسمى **Pop** ( يزيل آخر عنصر دخل الى **stack** أولاً )
- ❖ عبارة عن **Stack** هو ( يستخدم لتخزين البيانات والتعامل معها بطريقة معينة )

```
vector <int> vNumbers ;

vNumbers.push_back(10);
vNumbers.push_back(20);
vNumbers.push_back(30);
vNumbers.push_back(40);
vNumbers.push_back(50);

طباعة حجم (عدد) العناصر الموجودة في Stack - بعد تعبئته عناصرها -
cout << "Stack Size : " << vNumbers.size() << endl;

//... (50) Stack إزالة آخر عنصر دخل
vNumbers.pop_back();
vNumbers.pop_back();
vNumbers.pop_back();
vNumbers.pop_back();
vNumbers.pop_back();

طباعة حجم (عدد) العناصر الموجودة في Stack بعد إزالة عناصرها //
cout << "Stack Size : " << vNumbers.size() << endl;

إذا استخدمت () Stack و pop_back() لا يوجد به عناصر : البرنامج لا يستغل
للتتأكد من أن Stack فاضي استخدم //

// 1- إذا كان Stack vNumbers ليس فارغ نفذ الشرط
if (! vNumbers.empty())
 vNumbers.pop_back();

إذا كان عدد العناصر في vNumbers داخل Stack أكبر من 0 نفذ الشرط 2-
if (vNumbers.size() > 0)
 vNumbers.pop_back();

لإزالة جميع العناصر من Stack دفعه واحدة //
vNumbers.clear();

إزالة جميع العناصر باستخدام Ranged Loop
for (int& Number : vNumbers)
{
 vNumbers.pop_back();
}

cout << "Numbers Vector : \n\n";

for (int& Number : vNumbers)
{
 cout << Number << endl;
}
cout << endl;
```

```

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

int main()
{
 vector <int> vNumbers5 ;

 vNumbers5.push_back(10);
 vNumbers5.push_back(20);
 vNumbers5.push_back(30);
 vNumbers5.push_back(40);
 vNumbers5.push_back(50);

 vNumbers5.clear();
 // error الخطأ يسمى استثناء أو اعتراف
 // exception
 // empty() لذا يفضل وضع شرط لأخذ بيانات من العناصر إما size() أو
 // empty()

 if (vNumbers5.size() > 0)
 {
 // Vector بطبع أول عنصر في
 cout << "First Element : " << vNumbers5.front() << endl;
 }

 if (!vNumbers5.empty())
 {
 // Vector بطبع آخر عنصر في
 cout << "Last Element : " << vNumbers5.back() << endl;
 }

 // Stack طباعة حجم (عدد) العناصر الموجودة في
 cout << "Size : " << vNumbers5.size() << endl;

 // Vector الحجم الكلي ل
 cout << "Capacity : " << vNumbers5.capacity() << endl;

 // إرجاع 1 إذا كان Stack لا يوجد به بيانات
 // إرجاع 0 إذا كان Stack يوجد به بيانات
 cout << "Empty : " << vNumbers5.empty() << endl;
}

```

❖ لكل متغير يتم حجز مكان له (**Slot**) في الذاكرة يحتوي على

○ اسم المتغير

○ وقيمتة

○ وعنوانه **Address** في الذاكرة

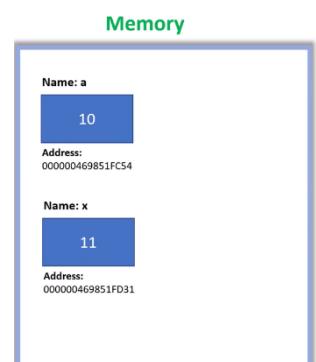
- **Reference** مكتوب بلغة **Address** ويسمى **Address** ❖

❖ هو مرجع لعنوان مكان المتغير في الذاكرة

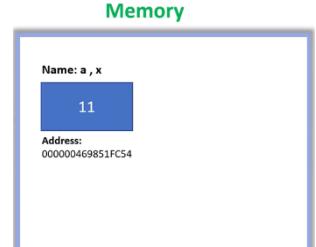
❖ **Reference & Tسمى**

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

// By Value Parameter (int x)
void Function1(int x)
{
 // main() يتم إنشاء نسخة أخرى من المتغير الرئيسي في
 // هذه النسخة لها اسم وقيمة وعنوان Address آخر مختلف
 // التغييرات داخل Function لا تؤثر على القيمة الأصلية داخل main()
 x++; // 10 + 1 = 11
}
```



```
// By Reference Parameter (int & x)
void Function2(int & x)
{
 // main() يتم نسخ عنوان Address من المتغير الرئيسي في
 // هذه لها اسم مختلف (اسمان) وعنوان Address واحد
 // التغييرات داخل Function تؤثر على القيمة الأصلية داخل main()
 x++; // 10 + 1 = 11
}
```



لكل متغير يتم حجز مكان - له في الذاكرة يحتوي على //  
 اسم المتغير ، وقيمتة وعنوانه في الذاكرة //  
 Address مكتوبة بلغة **Address** ويسمى **Address** أو **Reference** - //  
 هو مرجع لعنوان مكان المتغير في الذاكرة //

```
int main()
{
 // main() يتم حجز مكان - له في الذاكرة يحتوي على
 // اسم المتغير ، وقيمتة وعنوانه في الذاكرة
 // Address مكتوبة بلغة Address ويسمى Address أو Reference -
 // هو مرجع لعنوان مكان المتغير في الذاكرة

 int a = 10;

 // By Value Parameter (int x)
 Function1(a);
 cout << a << endl; // print (10)

 // By Reference Parameter (int & x)
 Function2(a);
 cout << a << endl; // print (11)

 cout << a << endl;
 cout << "Hexadecimal (Address OR Reference) : " << &a << endl;
}
```

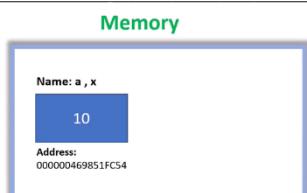
## # الدرس 34 : Creating References (Lesson #34 - Creating References)

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

```
int main()
{
 int a = 10;
```

// يتم إنشاء متغير له نفس عنوان المتغير المعرف  
// أي له اسمين مختلفين وعنوان واحد في الذاكرة Address ولهم قيمة واحدة  
int& x = a;

```
cout << &a << endl;
cout << &x << endl;
```



إذا تم تغيير قيمة أي واحد من المتغيرين - الاسمين - يتم التغيير على الجميع  
لأنهما لهما نفس العنوان // Address

```
x = 20;
```

```
cout << a << endl; // print (20)
cout << x << endl; // print (20)
```

```
}
```

## # الدروس 35 : المؤشر 1 ( Lesson #35 - What is Pointer ? )

- ❖ ما هو المؤشر **Pointer** ؟ هو متغير يخزن فيه عنوان **Address** لـ **Address** أو أي شيء له ( Function , Variable , OOP , Array , struct ... )
- ❖ خاص بلغة **C , C++** **Pointer**
- ❖ يعطي تحكم للمطور Developer كامل لإدارة الذاكرة **Pointer**
- ❖ هو متغير يخزن فيه فقط عنوان **Address** ( لا يخزن قيمة ) **Pointer**
- ❖ له عنوان **Address** آخر مختلف عن عنوان **Address** المتغير المخزن
- ❖ أي تعديل على قيمة **Pointer** يتم التعديل على قيمة المتغير الرئيسي المخزن
- ❖ نوع **Pointer** لابد أن يطابق نوع المتغير المراد تخزين **Address Type** فيه

### Syntax

| Type | Pointer إشارة | Name | أي شيء له Address |
|------|---------------|------|-------------------|
| int  | *             | P =  | &a                |

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

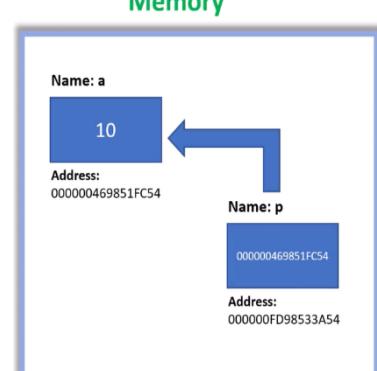
int main()
{
 int a = 10;
 int b = 20;

 cout << "a Value = " << a << endl;
 // Address طباعة
 cout << "a Address = " << &a << endl;

 // Pointer كتابة طريقة
 int* p;
 p = &a;

 // تغيير Address الى Pointer آخر
 p = &b;

 cout << "Pointer value = " << p << endl; // Print &b
 cout << endl;
}
```



## # الدرس 36 : المؤشر 2 (Lesson #36 - Dereferencing Pointer)

(**\*Name**) Pointer تعني : الوصول الى محتويات المتغير من **Dereference** ♦♦♦

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

int main()
{
 int a = 10;

 cout << "a Value = " << a << endl;
 cout << "a Address = " << &a << endl;

 int * p;
 p = &a;

 cout << "Pointer Value = " << p << endl;

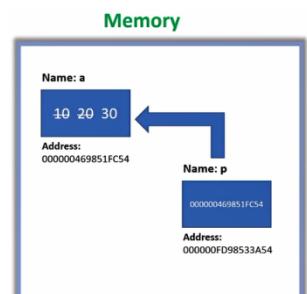
 // *Name = Pointer طباعة القيمة من
 cout << "Value of the address that p is pointing to is = " << *p <<
endl;

 استخدام إشارة * اخرى ل Pointer مؤشر مسبقا تعني
 // الوصول الى القيمة من المؤشر الى المتغير - يمكنك من التعديل عليها من
 *p = 20;

 cout << a << endl; // 20
 cout << *p << endl; // 20

 // *Name = Pointer تستطيع تغيير القيمة إما من المتغير الرئيسي أو من
 a = 30;
 cout << a << endl; // 30
 cout << *p << endl; // 30

 cout << endl;
}
```



## أخطاء شائعة يجب تجنبها في Pointer

```
int x = 10 , * p;
// خطأ
// Value لا يخزن قيمة P = Pointer
p = x;
p = 50;

// الصحيح
// Address يخزن فقط عنوان P = Pointer
p = &x;

// خطأ
// Dereference تعني الوصول الى القيمة
// لا يتم تخزين عنوان في قيمة Value
*p = &x;

// الصحيح
// Value تعدل قيمة
*p = x;
*p = 50;
```

## # الدروس 38 : المؤشر 4 (Lesson #38 - Pointers vs References)

| References &                         | Pointer *                       |
|--------------------------------------|---------------------------------|
| اسم آخر للمتغير                      | متغير آخر يتم تخزين Address فيه |
| لا يتم حجز له مكان في الذاكرة        | حجز مكان آخر له في الذاكرة      |
| لا يتم تحويله الى متغير آخر ( جامد ) | يتم تحويله الى متغير آخر ( من ) |

```

int a = 10;
int& x = a;

// Print Address
cout << &a << endl;
cout << &x << endl;

// Print Value
cout << a << endl;
cout << x << endl;

int* p = &a;

cout << p << endl;
cout << *p << endl;

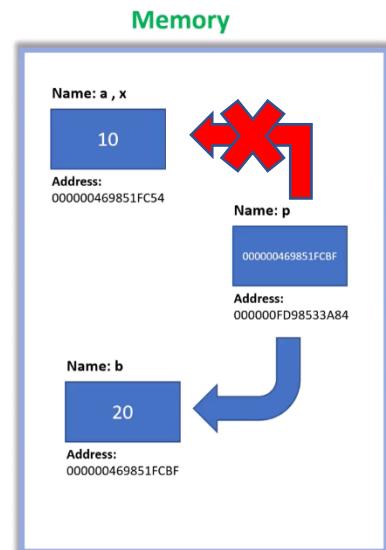
int b = 20;

// تحويل المؤشر * الى متغير آخر
p = &b;

cout << p << endl;
cout << *p << endl;

// خطأ
// تحويل الى متغير آخر By Reference &
&x = b;

```



## # الدرس 39 : المؤشر 5 (Lesson #39 - Call by Reference: Using pointers )

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

void swap_Ref(int& n1, int& n2)
{
 int temp;
 temp = n1;
 n1 = n2;
 n2 = temp;
}

void swap_Poin(int* n1, int* n2)
{
 int temp;
 temp = *n1;
 *n1 = *n2;
 *n2 = temp;
}

int main()
{
 int a = 1;
 int b = 2;

 cout << "Before swapping" << endl;
 cout << "a = " << a << endl;
 cout << "b = " << b << endl;

 swap_Ref(a, b);

 cout << "Before swapping" << endl;
 cout << "a = " << a << endl;
 cout << "b = " << b << endl;

 int s = 4; int r = 5;

 cout << "Before swapping" << endl;
 cout << "s = " << s << endl;
 cout << "r = " << r << endl;

 // Address * Pointer فيحتاج من نوع الباراميتر
 swap_Poin(&s, &r);

 cout << "Before swapping" << endl;
 cout << "a = " << s << endl;
 cout << "b = " << r << endl;
}
```

**Variable** : مجموعة من المتغيرات **Array** ♦♦♦

كل عنوان **Variable** له **Address** ♦♦♦

\* ( Name Pointer + Index ) = **Array** في **Value** للوصول للقيمة ♦♦♦

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

int main()
{
 int arr[4] = { 10,20,30,40 };
 int* ptr;
 // يخزن عنوان أول قيمة - بشكل آوتوماتيك
 ptr = arr;

 cout << "Address array : \n";

 // &arr[0] يخزن عنوان لأول عنصر في Address ptr
 cout << ptr << endl;

 // &arr[1] يخزن عنوان لثاني عنصر في Address ptr + 1
 cout << ptr + 1 << endl;

 // &arr[2] يخزن عنوان ثالث عنصر في Address ptr + 2
 cout << ptr + 2 << endl;
 // &arr[3] يخزن عنوان رابع عنصر في Address ptr + 3
 cout << ptr + 3 << endl;

 cout << "\n Value array : \n";

 cout << *(ptr) << endl; // 10
 cout << *(ptr + 1) << endl; // 20
 cout << *(ptr + 2) << endl; // 30
 cout << *(ptr + 3) << endl; // 40

 cout << endl;

 for (int i = 0 ; i < 4 ; i++)
 {
 cout << *(ptr + i) << endl;
 }

 cout << endl;
}
```

❖ **Structure** = يدل على أنه **Pointer** من نوع **Name Pointer ->** ❖

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

struct stEmployee
{
 string FirstName;
 string LastName;
 int Salary;
};

int main()
{
 stEmployee Employee1, * ptr;

 Employee1.FirstName = "Mohammed Abu-Hadhoude";
 Employee1.Salary = 5000;

 cout << Employee1.FirstName << endl;
 cout << Employee1.Salary << endl;

 ptr = &Employee1;

 // ptr-> بعد النقطة تظهر التغييرات ثم بعد الضغط على إحداها يتم تحويله إلى cout << "\n Using Pointer : \n" ;
 cout << ptr->FirstName << endl;
 cout << ptr->Salary << endl;
}
```

- ❖ تستطيع تحويل ( Variable , Function , Array ) الى متغير آخر سواء ( int , float , double , .... ) **Type** لكن بشرط أن يكون من نفس النوع
- ❖ إذا كنت لا تعرف على ماذا تؤشر في Run Time تستطيع تعريف Pointer من نوع ( تؤشر على كل شيء له Address ) Generic - **void**
- ❖ إذا عرفت من نوع **void** لا تستطيع تعديل أو طباعة القيمة **Value** إلا باستخدام الأمر التالي

### Syntax

|                                                         |
|---------------------------------------------------------|
| <b>*( static_cast &lt; Type * &gt; ( NamePointer ))</b> |
| <b>*( static_cast &lt; float * &gt; ( ptr ))</b>        |

```

void* ptr;

float f1 = 10.5;
ptr = &f1;

// Address طباعة
cout << "Address f1 : " << ptr << endl;

// خطأ
// void أو تعديل القيمة ل Pointer من نوع
cout << *ptr << endl;
*ptr = 33.2;
// الصحيح
// void أو تعديل القيمة ل Pointer من نوع
cout << *(static_cast < float *> (ptr)) << endl;

(static_cast < float> (ptr)) = 22.8;
cout << *(static_cast < float*> (ptr)) << endl;

int x2 = 50;
// تحويل إلى Pointer الى متغير آخر
ptr = &x2;

// Address طباعة
cout << "Address x2 : " << ptr << endl;

// خطأ
// void أو تعديل القيمة ل Pointer من نوع
cout << *ptr << endl;
*ptr = 44;
// الصحيح
// void أو تعديل القيمة ل Pointer من نوع
cout << *(static_cast < int*> (ptr)) << endl;

(static_cast < int> (ptr)) = 99;
cout << *(static_cast < int*> (ptr)) << endl;

```

- ❖ من أسباب قوة C++ هي : إدارة الذاكرة – التحكم الكامل للذاكرة
- ❖ تستطيع تعريف وحذف المتغير **أثناء** تشغيل البرنامج Run Time باستخدام **Pointer**
- ❖ المتغير العادي يبقى في الذاكرة الى نهاية البرنامج
- ❖ من غير وجود **Pointer** لا تستطيع جعل الذاكرة أتوماتيكية
- ❖ كل **delete NamePointer** لا بد أن تستخدم مقابلها **new Type**
- ❖ تستخد **delete** و **new** مثلا مع الشروط

```
// declare an int Pointer
int* ptrX;

// declare a float Pointer
float* ptrY;

// dynamically allocate memory
// إنشاء متغير مؤقت في الذاكرة
ptrX = new int;
ptrY = new float;

// assigning value to the memory
// إضافة قيمة للمتغير المؤقت في الذاكرة
*ptrX = 45;
*ptrY = 58.35f;

cout << *ptrX << endl;
cout << *ptrY << endl;

// deallocate the memory
// إزالة المتغير المؤقت من الذاكرة
delete ptrX;
delete ptrY;

cout << endl;
```

- العادلة تحجز مساحة محددة في الذاكرة – سواء استخدمتها كلها أو لا –
- ❖ حجز مساحة في الذاكرة مع عدم استخدامها يؤدي إلى بطيء في البرنامج
- ❖ تحجز في الذاكرة ما تحتاجه فقط مع **Pointer Array**
- ❖ كل **delete** **NamePointer** لا بد أن تستخدم مقابلها **new Type**

```

int Num;
cout << "Enter total number of students : ";
cin >> Num;

float* ptr;

// memory allocation of Num number of floats
ptr = new float[Num];

cout << "Enter grades of students." << endl;

for (int x = 0; x < Num; x++)
{
 // Array العنصر في رقم [x + 1]
 cout << "Student [" << x + 1 << "] : ";

 // index = (ptr + x)
 cin >> *(ptr + x);
}

cout << "\n Displaying grades of students." << endl;

for (int i = 0; i < Num; i++)
{
 cout << "Student [" << i + 1 << "] : " << *(ptr + i) <<
endl;
}

// ptr memory is released
delete [] ptr;

```

- ❖ الذاكرة **Memory** يقصد بها في البرمجة :
  - ❖ البرنامج يقسم في **RAM** إلى أربعة أقسام هي
    ١. يخزن فيه كود الأوامر **Source Code / Instruction** ( يأخذ مساحة قليلة )
    ٢. تكون دورة حياتهما طوال حياة كامل البرنامج **Variables Static / Global** ( يأخذ مساحة قليلة )
    ٣. هو: الذاكرة **Stack** التي يخصصها **Operating System** ( يأخذ مساحة على حسب البرنامج )
    ٤. يستطيع Developer الوصول إلى **Heap** بقية الذاكرة باستخدام **Pointers** ( Local : Variables / Function / Pointers )  
يُخزن فيه ( Stack Address )
- ptrX = new int;**  
**Heap** : any dynamic Variables / Objects / Arrays ...etc.

### ❖ **Vector** يعامل معاملة **Array**

```
// { 1, 2, 3, 4, 5 } Array
vector <int> num{ 1,2,3,4,5 };

// يستطيع الوصول إلى أي عنصر بطريقتين
// NameVector.at(Index) باستخدام الطريقة الأولى
cout << "\n\n using .at(1) \n";
cout << "Element at Index 0 : " << num.at(0) << endl;
cout << "Element at Index 2 : " << num.at(2) << endl;
cout << "Element at Index 4 : " << num.at(4) << endl;
// زراعة Index عن العناصر يعرض البرنامج على ذلك Exception وينذهب بك إلى كود cout << "Element at Index 5 : " << num.at(5) << endl;

// NameVector[Index] باستخدام الطريقة الثانية
cout << "\n\n using [1] \n";
cout << "Element at Index 0 : " << num[0] << endl;
cout << "Element at Index 2 : " << num[2] << endl;
cout << "Element at Index 4 : " << num[4] << endl;
// زراعة Index عن العناصر يحدث خطأ للبرنامج Wrong عند cout << "Element at Index 5 : " << num[5] << endl;
```

❖ لطباعة **Vector** فقط وعدم التعديل عليه استعمل **const** مع إشارة **&** تكون سريعة جدا في **Ranged Loop**

```
vector <int> num{ 1,2,3,4,5 };

cout << "Initial Vector : ";

// & مع const
// تثبيت قيمة المتغير وعدم القدرة على تغييرها
for (const int& i : num)
{
 cout << i << " ";
 // Print 1 2 3 4 5
}

cout << "\n\n Updated Vector : ";

// إشارة &
// تستطيع تغيير قيمة المتغير
for (int& i : num)
{
 i = 20;
 cout << i << " ";
 // Print 20 20 20 20 20
}

// تستطيع تغيير قيمة المتغير
num[1] = 40;
num.at(2) = 80;
num.at(4) = 90;

cout << "\n\n Updated Vector : ";
// & مع const
// تثبيت قيمة المتغير وعدم القدرة على تغييرها
for (const int& i : num)
{
 cout << i << " ";
 // Print 20 40 80 20 90
}
```

**Pointer** : طريقة للمرور على عناصر **Vector** باستخدام **Iterator** ♦♦♦

لابد من تطابق نوع **Iterator** مع نوع **Vector** ♦♦♦

لوصول الى أول عنصر في **Vector** **NameVector.begin()** ♦♦♦

لوصول الى آخر عنصر في **Vector** **NameVector.end()** ♦♦♦

### Syntax

| Vector | < Type > | :: | Iterator | Name Iterator |
|--------|----------|----|----------|---------------|
| vector | < int >  | :: | Iterator | Iter          |

```
vector <int> num{ 1,2,3,4,5 };

// declare iterator
vector <int> ::iterator iter;

// use Iterator with for loop
for (iter = num.begin(); iter != num.end(); iter++)
{
 cout << *iter << " ";
}
```

## الأخطاء في البرمجة ثلاثة

١. أخطاء في قواعد الكتابة (Syntax error) وهو أسهل أنواع الأخطاء
٢. خطأ منطقي (Logical error) وهو أصعب أنواع الأخطاء ولهذا جاء التصحيح Debugging
٣. خطأ في وقت التشغيل (Run time error) مثل إغلاق البرنامج بشكل مفاجئ Crash

❖ يقصد بها : كتابة الكود الذي قد يسبب أي مشكلة في البرنامج بطريقة تضمن أنه إذا حدث الخطأ المتوقع – أو أي خطأ آخر يصعب التحكم به بطريقة أخرى – فإن البرنامج لن يعلق أو يتم إغلاقه بشكل مفاجئ Crash

❖ موجود فيأغلب لغات البرمجة Exception Handling

❖ إغلاق البرنامج بشكل مفاجئ له أحوال كثيرة أمثلة Crashes

❖ ( Vector , حفظ الملفات على مجلد أو حفظ المجلد C or D drive )

❖ يجب الحذر من Exception Handling وعدم استخدامه – إذا توفر البديل - إلا عند الضرورة القصوى لأنه يبطئ البرنامج

❖ السطر الذي تشك أنه يعمل إغلاق البرنامج بشكل مفاجئ Crash ولا يتم معالجته إلا بـ ( try { } catch(..) ) Exception Handling

### Syntax

|     |           |               |           |
|-----|-----------|---------------|-----------|
| try | { Crash } | catch ( ... ) | { Crash } |
|-----|-----------|---------------|-----------|

{ تنفيذ الأوامر إذا حدث Crash }

```
vector <int> num{ 1,2,3,4,5 };

try
{
 // Exception
 // crash
 cout << num.at(5);
}
catch (...)
{
 // Handling
 // crash
 cout << "out of bound \n";
}
```

- ❖ بعض أو أغلب **String Object** أو **Function** الجاهزة في **Method**
- ❖ عند استخدام **String Object** لابد من استدعاء مكتبة **<string>**
- ❖ ( Object Oriented Programming ) **OOP** هي مثل على **String Object**
- ❖ **NameString.length()** مثل **Function and Procedure** فيها **Method**

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
 string S1 = "My Name is Mohammed Abu-Hadhoud , I Love Programming.";

 // Prints the length of string
 cout << S1.length() << endl;

 // Returns the letter at position 3
 cout << S1.at(3) << endl;

 // Adds @ProgrammingAdvices to the end of string
 S1.append(" @ProgrammingAdvices");
 cout << S1 << endl;

 // inserts Ali at Position 7
 // أدرج (في الموقع 7 ، النص التالي) !
 S1.insert(7, " Ali ");
 cout << S1 << endl;

 // Prints all the next 8 letters from position 16
 // اعط جزء من النص (الموضع 16 ، بطول 8) أحرف
 cout << S1.substr(16,8) << endl;

 // Adds one character to the end of the string
 S1.push_back('X');
 cout << S1 << endl;

 // Removes one character to the end of the string
 S1.pop_back();
 cout << S1 << endl;

 // Finds Ali in the string
 cout << S1.find("Ali") << endl;

 // Finds Ali in the string
 cout << S1.find("ali") << endl;
 // إذا لم يوجد النص يرجع رقم طويل
 if (S1.find("ali") == S1npos)
 {
 // مخزن فيه الرقم الذي يدل على الخطأ . npos
 // يعني لم يوجد النص المطلوب
 cout << "ali is not found";
 }

 // clears all string letters
 S1.clear();
 cout << S1 << endl; }
```

- ❖ استدعاء مكتبة `<cctype>`
- ❖ أي رقم غير الصفر `0` يعتبر `True`

```
#include <iostream>
#include <cctype>
using namespace std;
int main()
{
 char x;
 char w;

 تجويل الحرف من الصغير الى الكبير //
 x = toupper('a');

 تجويل الحرف من الكبير الى الصغير //
 w = tolower('A');

 طباعة رقم الرمز أو الحرف في جدول
 cout << toupper('a') << endl;
 cout << tolower('A') << endl;

 cout << "converting a to A : " << x << endl;
 cout << "converting A to a : " << w << endl;

 // إذا كان الجواب 0 فهو خطأ
 // أما إن كان الجواب رقم آخر غير 0 فهو صحيح //
 // Digits (A to Z)
 // returns zero if not , and non zero of yes
 cout << "isupper('A') " << isupper('A') << endl;

 // lower case (a to z)
 // returns zero if not , and non zero of yes
 cout << "islower('a') " << islower('a') << endl;

 // Digits (0 to 9)
 // returns zero if not , and non zero of yes
 cout << "isdigit('5') " << isdigit('5') << endl;

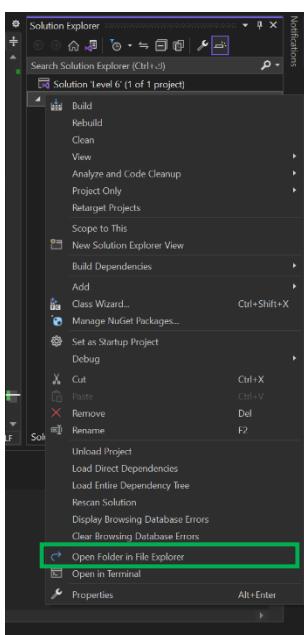
 // punctuation characters are !#$%&'()*+,-./:;<=>?@[\\]^_`{|}~
 // returns zero if not, and non zero of yes
 cout << "ispunct(';) " << ispunct(';) << endl;

 return 0;
}
```

- ❖ لابد من استدعاء مكتبة `#include <fstream>`
- ❖ يوجد أكثر من طريقة للكتابة على الملف منها **Write Mode**
- ❖ تستطيع تخزين البيانات في ملف :

  ١. يمكن التعديل على البيانات
  ٢. إضافة بيانات أخرى
  ٣. حذف البيانات

- ❖ كلما تعلم Run يبدأ الحفظ من جديد – يحذف البيانات السابقة –
- ❖ بعد الانتهاء من البيانات المراده حفظها في الملف : يجب إغلاق الملف **close()**



١. اختر من الشريط العلوي (**View**)
٢. ثم اختر (**Solution Explorer**) ستظهر نافذة
٣. ثم اختر من النافذة (**اسم المشروع** (**Project**)) زر الفأرة الأيمن
٤. ثم اختر (**Open Folder In File Explorer**)
٥. سينذهب الى الملف في الهايد ديسك ثم افتح الملف

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
 fstream MyFile;
 // MyFile.open("اسم الملف المخزن في الهايد ديسك. ونوعه", ios::out);
 MyFile.open("MyFile.txt", ios::out); // Write Mode

 // للتأكد من أن الملف يعمل
 if (MyFile.is_open())
 {
 // للطباعة على الملف وليس على الشاشة
 // لكن بدل cout اسم الملف
 MyFile << "Hi, this is the first line \n";
 MyFile << "Hi, this is the second line \n";
 MyFile << "Hi, this is the third line \n";

 // بعد الانتهاء من الملف لابد من إغلاقه
 MyFile.close();
 }
}
```

❖ **Append Mode** يبدأ بـ Run بـ **Run** كلما تعلم يضيف المعلمات - لا يحذف البيانات السابقة

❖ إضافة معلومات إضافية مع الاحتفاظ بالبيانات السابقة **ios::app**

❖ إنشاء ملف **ios::out** أو إضافة معلومات **ios::app** مع إعطاء الأولوية لإضافة

المعلمات

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
 fstream MyFile;

 // MyFile.open("MyFile.txt", ios::out | ios::app);
 // إنشاء ملف ios::out ، اسم الملف المخزن في الهايد ديسك. ونوعه ios::app
 // إضافة معلومات إضافية مع الاحتفاظ بالبيانات السابقة
 // إنشاء ملف ios::out | ios::app مع إعطاء أولوية لإضافة المعلومات
 MyFile.open("MyFile.txt", ios::out | ios::app); // append Mode

 // للتأكد من أن الملف يعمل
 if (MyFile.is_open())
 {
 // للطباعة على الملف وليس على الشاشة
 // لكن بدل cout اسم الملف
 MyFile << "Hi, this is a new line \n";
 MyFile << "Hi, this is another line \n";

 // بعد الانتهاء من الملف لابد من إغلاقه
 MyFile.close();
 }
}
```

لقراءة معلومات فقط الملف وطباعتها على الشاشة – لا يمكن التعديل عليها – ♦  
أسرع أمر لقراءة الملفات **ios::in**

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

void PrintFileContent(string FileName)
{
 fstream MyFile;

 // للقراءة فقط ios::in
 MyFile.open(FileName, ios::in); // Read Mode

 // للتأكد من أن الملف يعمل
 if (MyFile.is_open())
 {
 string Line;

 // احصل على السطر الأول من الملف ، واحفظه في هذا
 while (getline(MyFile, Line))
 {
 cout << Line << endl;
 }

 // بعد الانتهاء من الملف لا بد من إغلاقه
 MyFile.close();
 }
}

int main()
{
 PrintFileContent("MyFile.txt");
}
```

احصل على السطر الأول من الملف ، واحفظه في هذا  
 بعد الانتهاء من الملف لا بد من إغلاقه

❖ للتعديل على الملف من داخل البرنامج

### ١. نسخ المعلومات في **Vector**

٢. أثناء نسخ المعلومات يتم إضافة معلومات جديدة أو التعديل على **Vector**

٣. حفظ المعلومات التي تم تعديليها أو إضافتها على الملف **File**

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>

void LoadDataFromFileToVector(string FileName , vector <string> &vFileContent)
{
 fstream MyFile;

 // ios::in للقراءة فقط
 MyFile.open(FileName, ios::in); // Read Mode

 // للتأكد من أن الملف يعمل
 if (MyFile.is_open())
 {
 string Line;

 // احصل على السطر الأول من الملف ، واحفظه في هذا
 while (getline(MyFile, Line))
 {
 // كل سطر في الملف يتم إضافتها إلى
 vFileContent.push_back(Line);
 }

 // بعد الانتهاء من الملف لا بد من إغلاقه
 MyFile.close();
 }
}

int main()
{
 vector <string> vFileContent;

 LoadDataFromFileToVector("MyFile.txt", vFileContent);

 for (string& Line : vFileContent)
 {
 cout << Line << endl;
 }

 return 0;
}
```

❖ للتعديل على الملف من داخل البرنامج

١. نسخ المعلومات في **Vector**

٢. إنشاء نسخ المعلومات يتم إضافة معلومات جديدة أو التعديل على **Vector**

٣. حفظ المعلومات التي تم تعديلها أو إضافتها على الملف **File**

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>

void SaveVectorToFile(string FileName , vector <string> vFileContent)
{
 // عدم وضع إشارة & للفيكتور في الباراميتر : لأنه للقراءة فقط وعدم التعديل عليه
 fstream MyFile;

 // ios::out لفتح ملف جديد أو حفظ المعلومات الجديدة في الملف مع كل أمر
 MyFile.open(FileName, ios::out); // Write Mode

 // للتأكد من أن الملف يعمل
 if (MyFile.is_open())
 {
 // - Address لعدم إنشاء نسخة أخرى - الذهاب الى
 for (string& Line : vFileContent)
 {
 // ! = "" لالتأكد من أن المدخل ليس فارغ
 if (Line != "")
 {
 MyFile << Line << endl;
 }
 }

 // بعد الانتهاء من الملف لابد من إغلاقه
 MyFile.close();
 }

 int main()
 {
 vector <string> vFileContent{"Saeed" , "Omar" , "Rowida" ,
 "Saleh" , "Mohammed" , "Khaled" };

 SaveVectorToFile("MyFile.txt" , vFileContent);

 return 0;
 }
}
```

❖ للتعديل على الملف من داخل البرنامج

١. نسخ المعلومات في **Vector**

٢. إثناء نسخ المعلومات يتم إضافة معلومات جديدة أو التعديل على **Vector**

٣. حفظ المعلومات التي تم تعديلها أو إضافتها على الملف **File**

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>

void LoadDataFromFileToVector(string FileName , vector <string> &vFileContent)
{
 fstream MyFile;

 // ios::in للقراءة فقط
 MyFile.open(FileName, ios::in); // Read Mode

 // للتأكد من أن الملف يعمل
 if (MyFile.is_open())
 {
 string Line;

 // احصل على السطر الأول من الملف ، واحفظه في هذا
 while (getline(MyFile, Line))
 {
 // Vector يتم إضافتها إلى
 vFileContent.push_back(Line);
 }

 // بعد الانتهاء من الملف لابد من إغلاقه
 MyFile.close();
 }
}

void SaveVectorToFile(string FileName , vector <string> vFileContent)
{
 // عدم وضع إشارة & للفيكتور في الباراميتر : لأنه للقراءة فقط وعدم التعديل عليه
 fstream MyFile;

 // Run ios::out لفتح ملف جديد أو حفظ المعلومات الجديدة في الملف مع كل أمر
 MyFile.open(FileName, ios::out); // Write Mode

 // للتأكد من أن الملف ي العمل
 if (MyFile.is_open())
 {
 // - Address & لعدم إنشاء نسخة أخرى - الذهاب الى
 for (string& Line : vFileContent)
 {
 // != لالتأكد من أن المدخل ليس فارغ
 if (Line != "")
 {
 MyFile << Line << endl;
 }
 }
 }
}
```

```

 }
 }

 بعد الانتهاء من الملف لابد من إغلاقه //

 MyFile.close();
}

void DeleteRecordFromFile(string FileName, string Record)
{
 vector <string> vFileContent;

 LoadDataFromFileToVector(FileName, vFileContent);

 لابد من استخدام إشارة & للتعديل على

 for (string& Line : vFileContent)
 {
 حذف البيانات //

 if (Line == Record)
 {
 Line = "";
 }
 }

 SaveVectorToFile(FileName, vFileContent);
}

void PrintFileContent(string FileName)
{
 fstream MyFile;

 // ios::in للقراءة فقط
 MyFile.open(FileName, ios::in); // Read Mode

 // التأكد من أن الملق يعمل
 if (MyFile.is_open())
 {
 string Line;

 // string Line , واحفظه في هذا
 while (getline(MyFile, Line))
 {
 cout << Line << endl;
 }

 بعد الانتهاء من الملف لابد من إغلاقه //

 MyFile.close();
 }
}

int main()
{
 cout << "File Content Before Delete. \n";
 PrintFileContent("MyFile.txt");

 DeleteRecordFromFile("MyFile.txt", "Omar");

 cout << "\nFile Content After Delete. \n";
 PrintFileContent("MyFile.txt");
 return 0;
}

```

❖ للتعديل على الملف من داخل البرنامج

١. نسخ المعلومات في **Vector**

٢. أثناء نسخ المعلومات يتم إضافة معلومات جديدة أو التعديل على **Vector**

٣. حفظ المعلومات التي تم تعديلها أو إضافتها على الملف **File**

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>

void LoadDataFromFileToVector(string FileName , vector <string> &vFileContent)
{
 fstream MyFile;

 // ios::in للقراءة فقط
 MyFile.open(FileName, ios::in); // Read Mode

 // للتأكد من أن الملف يعمل
 if (MyFile.is_open())
 {
 string Line;

 // احصل على السطر الأول من الملف ، واحفظه في هذا
 while (getline(MyFile, Line))
 {
 // Vector يتم إضافتها إلى
 vFileContent.push_back(Line);
 }

 // بعد الانتهاء من الملف لا بد من إغلاقه
 MyFile.close();
 }
}

void SaveVectorToFile(string FileName , vector <string> vFileContent)
{
 // لفتح ملف جديد أو حفظ المعلومات الجديدة في الملف مع كل أمر
 // Run ios::out
 MyFile.open(FileName, ios::out); // Write Mode

 // للتأكد من أن الملف ي العمل
 if (MyFile.is_open())
 {
 // - Address & لعدم إنشاء نسخة أخرى - الذهاب إلى
 for (string& Line : vFileContent)
 {
 // != لتأكيد من أن المدخل ليس فارغ
 if (Line != "")
 {
 MyFile << Line << endl;
 }
 }
 }
}
```

```

 }
 }

 بعد الانتهاء من الملف لابد من إغلاقه //
 MyFile.close();
}

void UpdateRecordInFile(string FileName, string Record , string UpdateTo)
{
 vector <string> vFileContent;

 LoadDataFromFileToVector(FileName, vFileContent);

 // لابد من استخدام إشارة & للتعديل على
 for (string& Line : vFileContent)
 {
 تبديل البيانات //
 if (Line == Record)
 {
 Line = UpdateTo;
 }
 }
 SaveVectorToFile(FileName, vFileContent);
}

void PrintFileContent(string FileName)
{
 fstream MyFile;

 // ios::in للقراءة فقط
 MyFile.open(FileName, ios::in); // Read Mode

 // للتأكد من أن الملق يعمل //
 if (MyFile.is_open())
 {
 string Line;

 // احصل على السطر الأول من الملف ، واحفظه في هذا
 while (getline(MyFile, Line))
 {
 cout << Line << endl;
 }

 بعد الانتهاء من الملف لابد من إغلاقه //
 MyFile.close();
 }
}

int main()
{
 cout << "File Content Before Update. \n";
 PrintFileContent("MyFile.tet");

 UpdateRecordInFile("MyFile.tet", "Saeed" , "Riyadh");

 cout << "\nFile Content After Update. \n";
 PrintFileContent("MyFile.tet");
 return 0;
}

```

## # الدروس 59 : التوقيت المحلي ( Lesson #59 Datetime: Local/UTC Time )

- ❖ لابد من استدعاء مكتبة `#include <ctime>`
- ❖ هذه المكتبة مأخوذة من لغة C
- ❖ لغة C++ تعطي خطأ للكود إذا كان غير آمن `warning`
- ❖ يتم تجاوز هذا الخطأ = `#pragma warning(disable : 4996)`

```
تجاوز الخطأ لرقم هذه المشكلة // (4996)

#include <iostream>
#include <ctime>

int main()
{
 // التوقيت المحلي
 time_t t = time(0); // get time now

 // تحويل التاريخ والوقت الى string
 char* dt = ctime(&t); // convert in string from
 cout << "Local date and time is : " << dt << "\n\n";

 // تحويل جرينتش
 tm* gmtm = gmtime(&t); // converting now to tm struct for UTC date /
 time
 // تحويل البيانات من string الى struct
 dt = asctime(gmtm);
 cout << "UTC date and time is : " << dt << "\n\n";
 return 0;
}
```

- ❖ لابد من استدعاء مكتبة `#include <ctime>`
- ❖ هذه المكتبة مأخوذة من لغة C
- ❖ لغة C++ تعطي خطأ للكود إذا كان غير آمن `warning`
- ❖ يتم تجاوز هذا الخطأ = `#pragma warning(disable : 4996)`

```
#pragma warning(disable : 4996) // تجاوز الخطأ لرقم هذه المشكلة

#include <iostream>
#include <ctime>

int main()
{
 /*
 int tm_sec; // seconds of minutes from 0 to 61
 int tm_min; // minutes of hour from 0 to 59
 int tm_hour; // hours of day from 0 to 24
 int tm_mday; // day of month from 1 to 31
 int tm_mon; // month of year from 0 to 11
 int tm_year; // year since 1900
 int tm_wday; // days since Sunday
 int tm_yday; // days since January 1st
 int tm_isdst; // hours of daylight savings time
 */

 time_t t = time(0); // get time now

 // يحتوي على العناصر التي بالأسفل هو struct tm* now
 tm* now = localtime(&t);

 cout << "Year: " << now->tm_year + 1900 << endl;
 cout << "Month: " << now->tm_mon + 1 << endl;
 cout << "Day: " << now->tm_mday << endl;
 cout << "Hour: " << now->tm_hour << endl;
 cout << "Min: " << now->tm_min << endl;
 cout << "Second: " << now->tm_sec << endl;
 cout << "Week Day (Days since sunday): " << now->tm_wday << endl;
 cout << "Year Day (Days since Jan 1st): " << now->tm_yday << endl;
 cout << "hours of daylight savings: " << now->tm_isdst << endl;

 return 0;
}
```