بسم الله الرحمن الرحيم

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

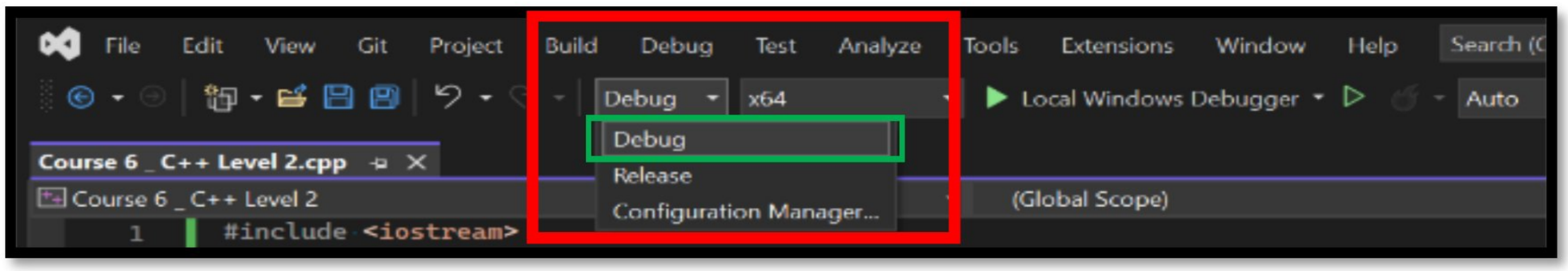
الأخطاء في البرمجة ثلاثة

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)

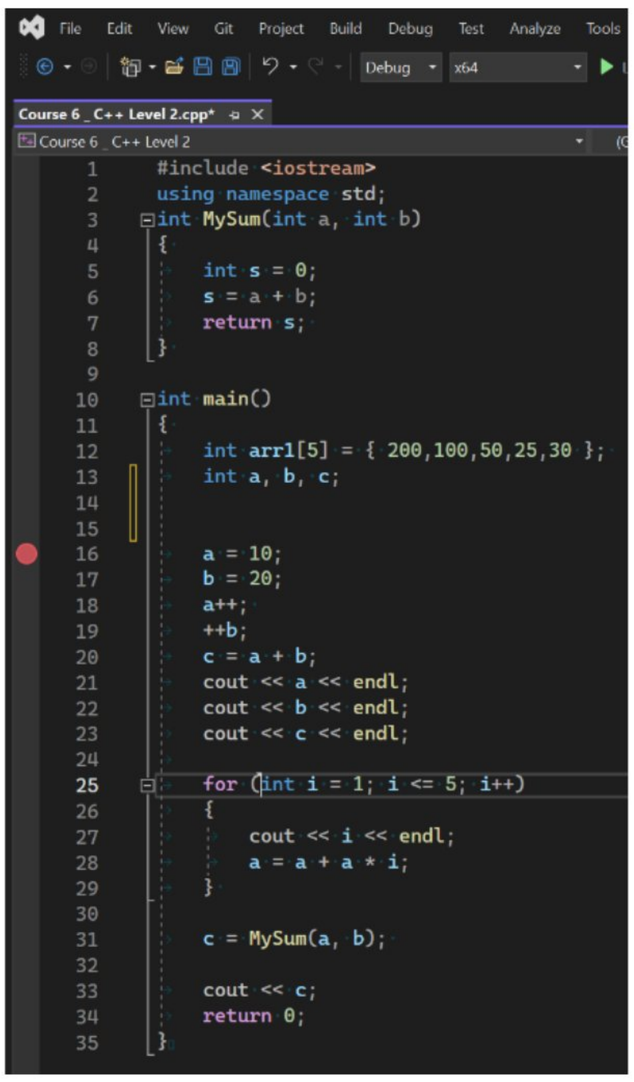
مميزات Debugging mood

- ❖ سرعة حل المشاكل
- ❖ تتبع الكود سطر سطر

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



Debugging mood : بطيء في تشغيل البرنامج + يستخدمه مطورو البرامج

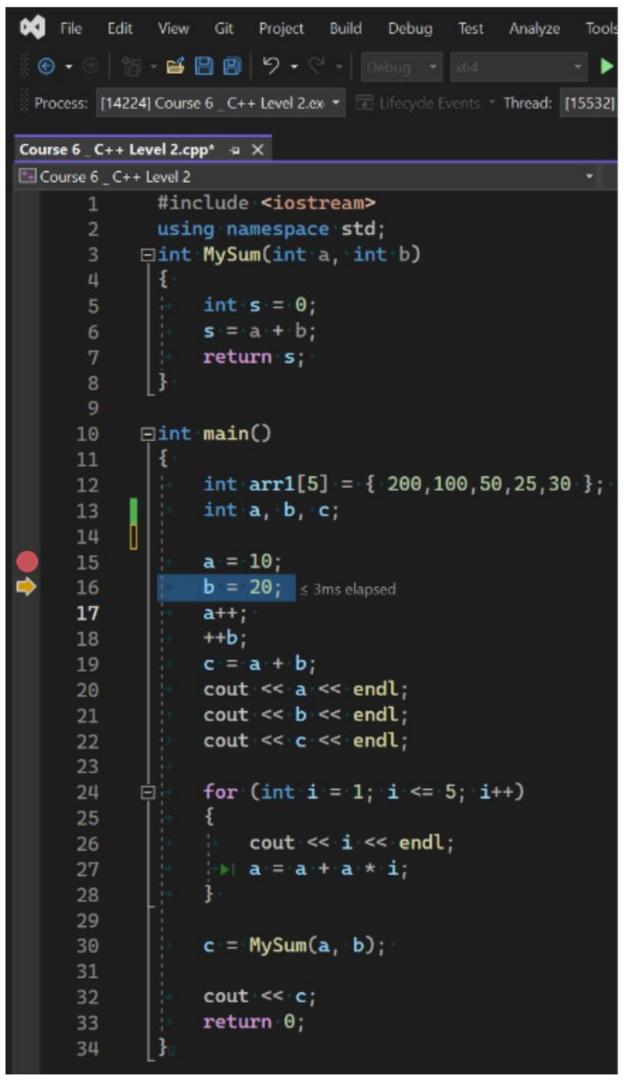


Breakpoint ●

عندما تعمل Bilde يشتغل 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 ●



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

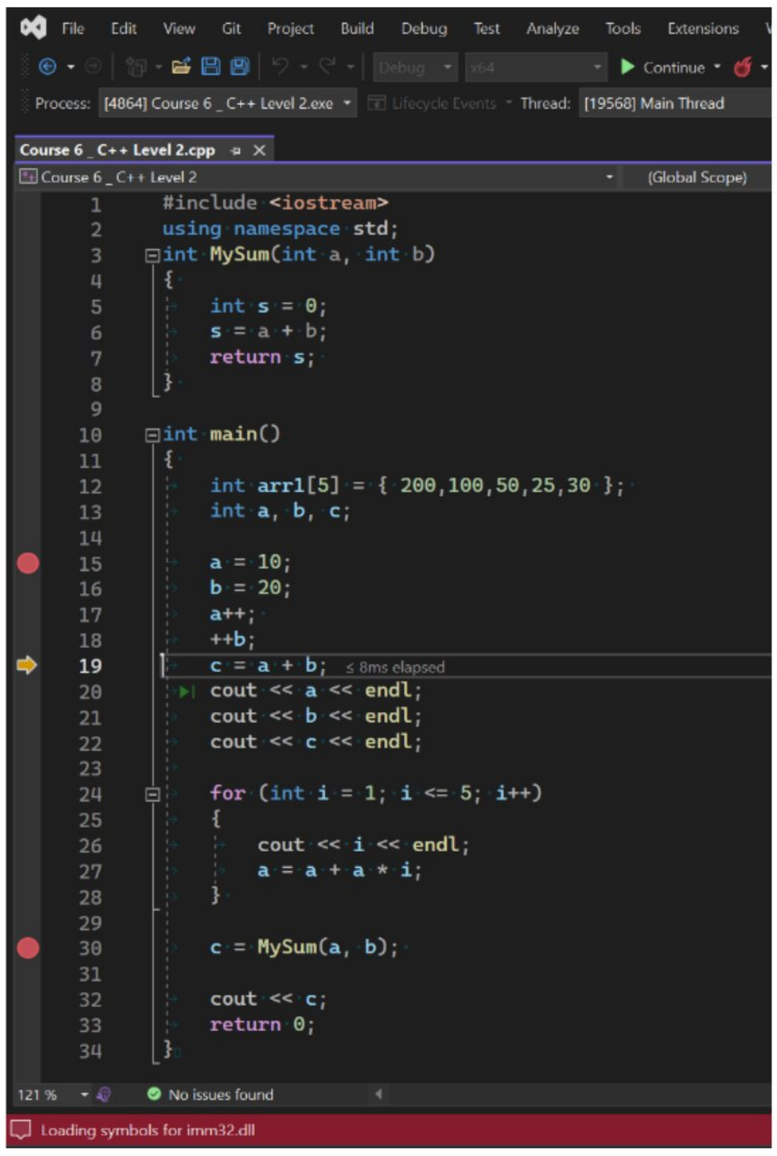
⇨ يدل على أن السهم متوقف عند الكود (لم يتم تنفيذه)

لتنفيذ سطر الكود والنزول لسطر جديد اضغط على اختصار (F11)

عند وضع مؤشر الفأرة على اسم المتغير تظهر القيمة المخزنة فيه

إذا لم تضع قيمة مبدئية للمتغير يضع الكومبيلر 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)

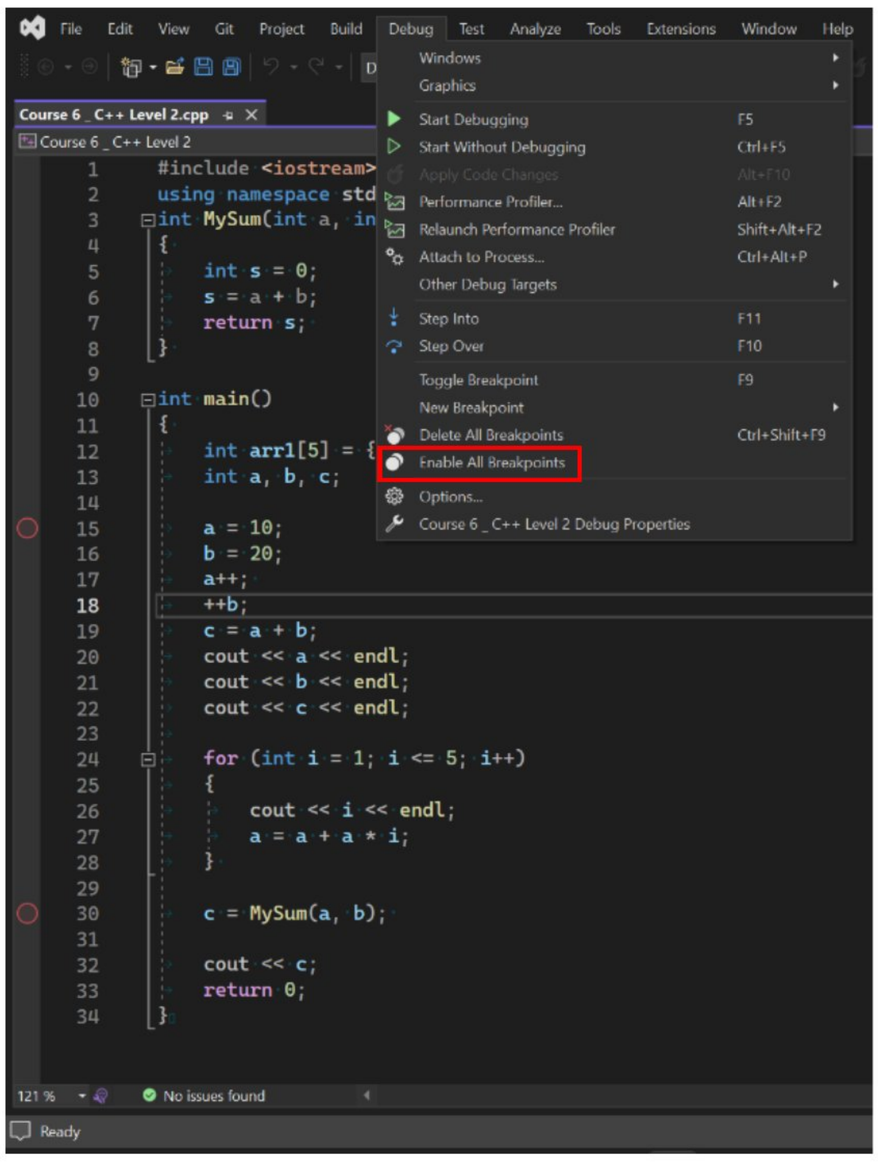


تستطيع إنشاء أكثر من واحدة في البرنامج

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

تستطيع التنقل أيضا بإمساك السهم بالماوس ووضعه في أي سطر تريده

تستطيع إيقاف Debugging mood عن طريق في الشريط العلوي



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

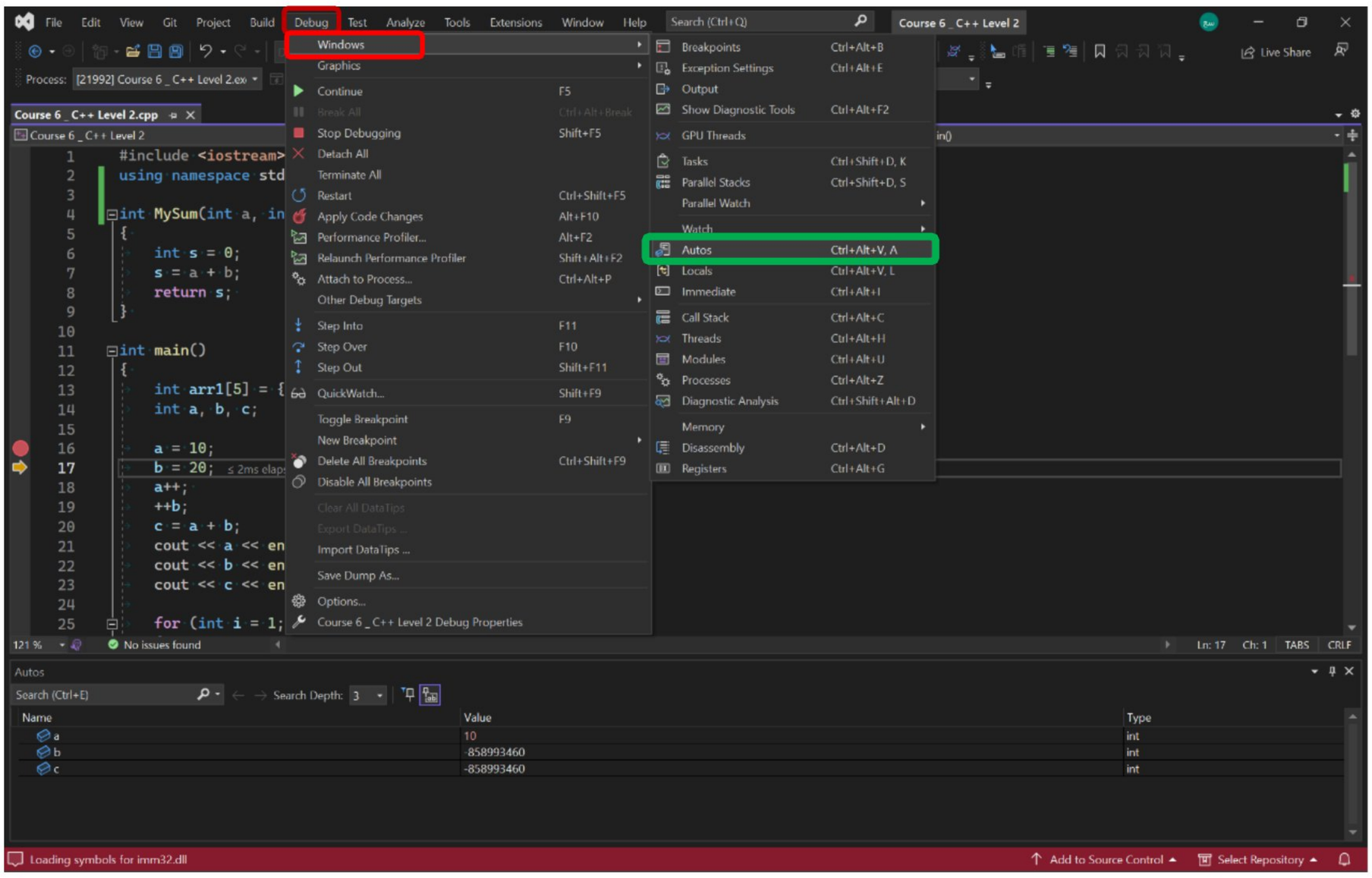
الدرس 4 : نافذة السيارات (Lesson #04 - Autos Window)

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

Autos نوافذ السيارات قد تشتغل تلقائيا

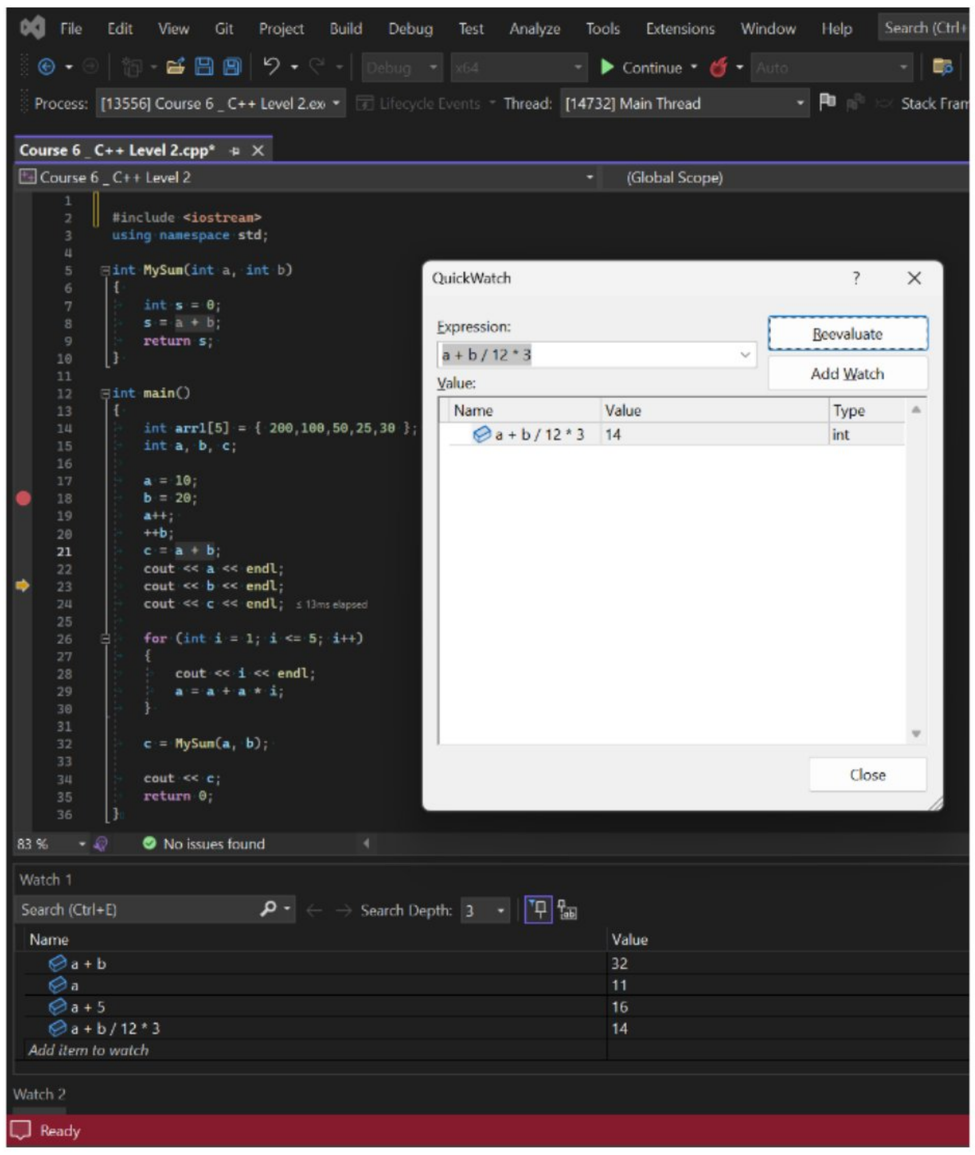
وإذا لم تشتغل تلقائيا اتبع الخطوات التالية

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



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

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

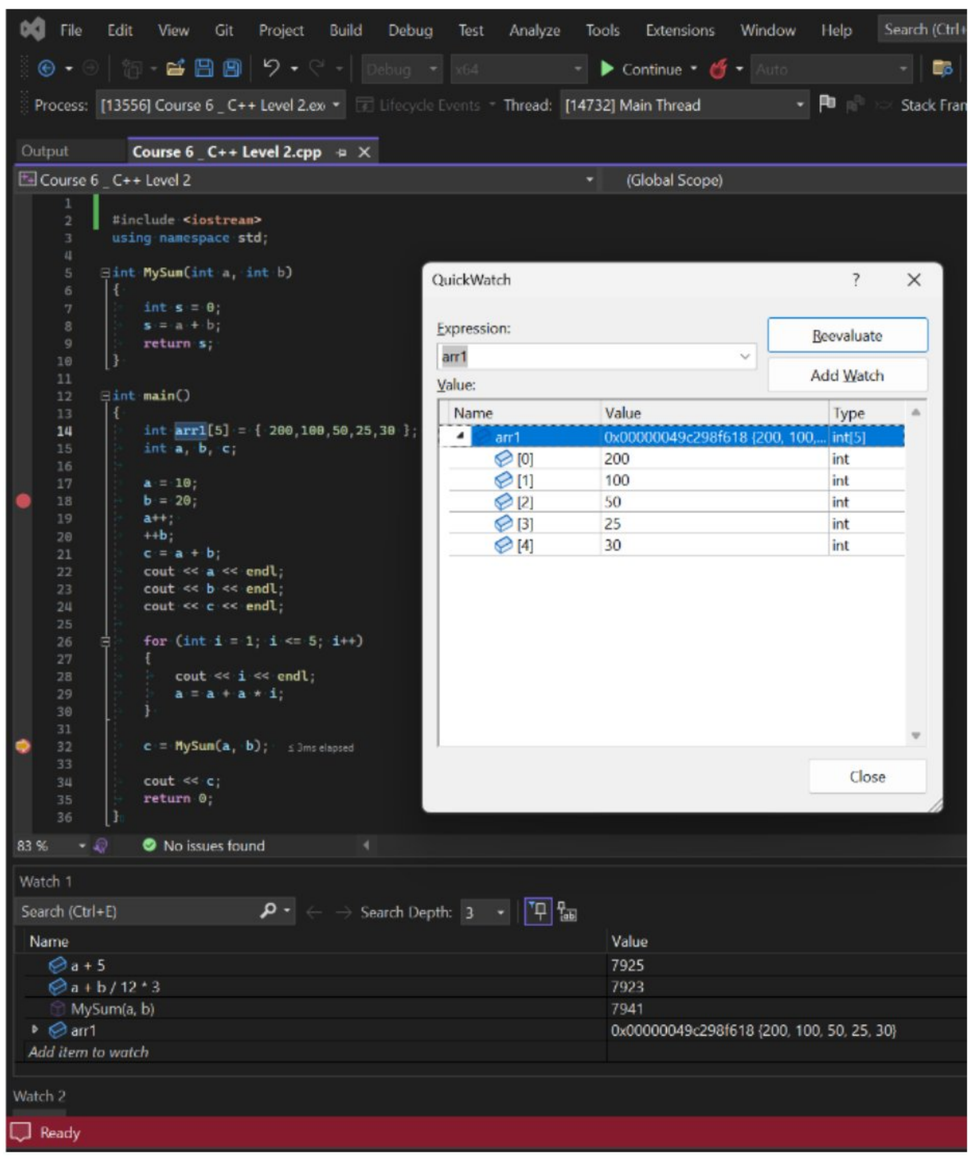


لمراقبة متغير معين

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)

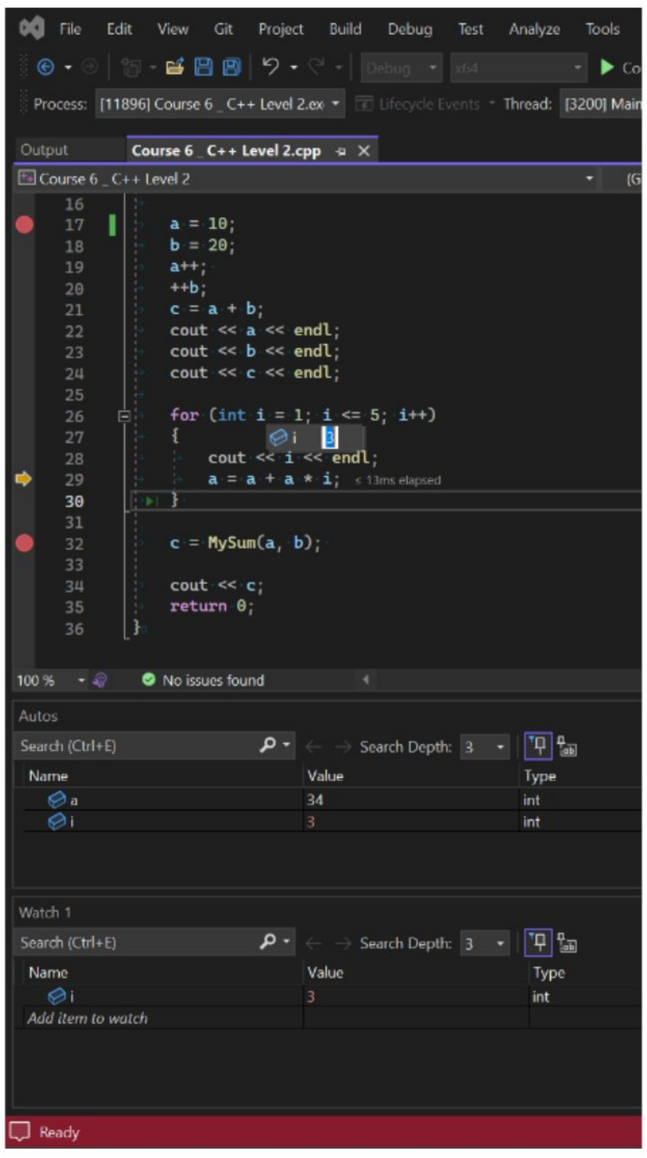


المتغير **arr1** قيمته هي عنوان موقعه في الذاكرة (**Address**) : لو استدعيت arr1 في **Function** سيرسل **Address** فقط وليست نسخة من **arr1** ، أي عند استدعاء **arr1** في **Function** سيذهب **Function** الى عنوان arr1 وليس **arr1** سترسل نسخة الى **Function** (ستدرس لاحقا في **Pointers**

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)

لتغيير قيمة المتغير أثناء التصحيح **Debugging mood** - بدون تغيير قيمته الأصلية في البرنامج -



١. شغل البرنامج على **Debugging mood**

٢. غير قيمة المتغير **باستخدام**

a. **مؤشر الفأرة** : عند وضع مؤشر الفأرة على اسم المتغير تظهر القيمة المخزنة فيه اضغط مرتين على قيمة المتغير ثم ضع القيمة

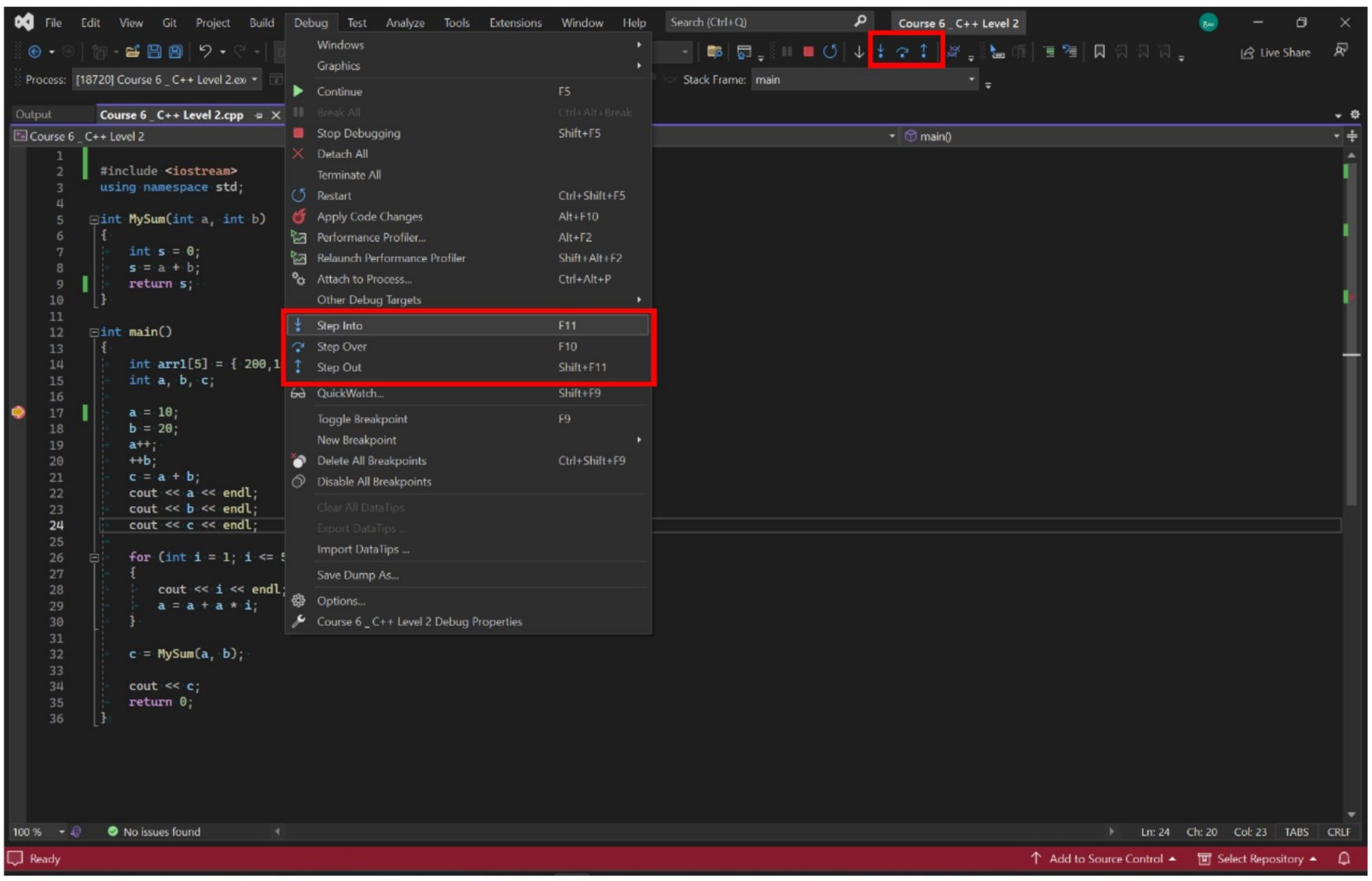
b. **Autos** اضغط مرتين على قيمة المتغير ثم ضع القيمة

c. **QuickWatch** اضغط مرتين على قيمة المتغير ثم ضع القيمة

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

أزرار التنقل في **Debugging mood**

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



الدرس 8 : مكتبة : أنشئ مكتبتك الخاصة (Lesson #08 - Library: Create Your Own Library)

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>`

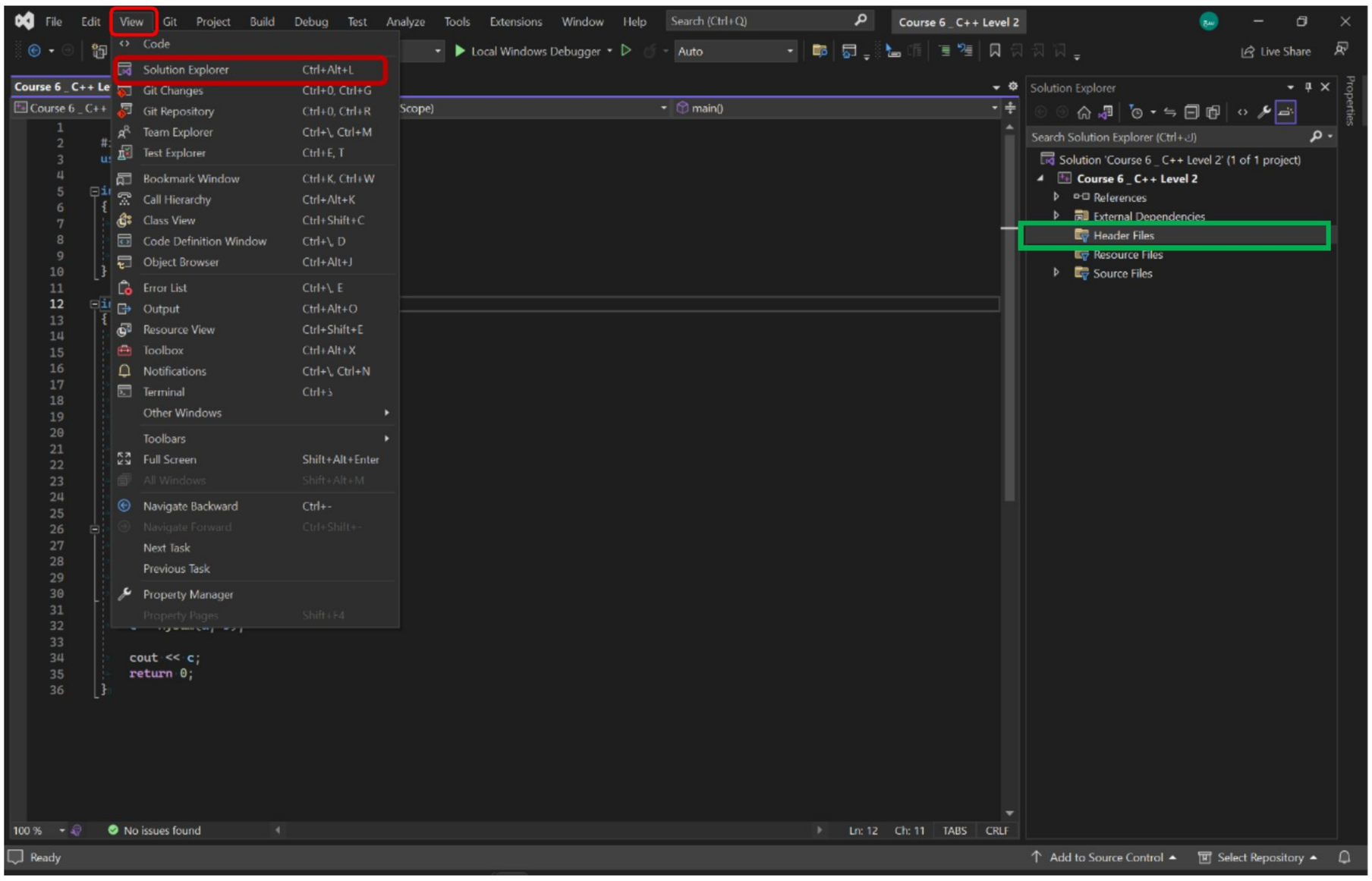
<u>اجعل لكل مكتبة اسم يعبر عن محتواها</u>

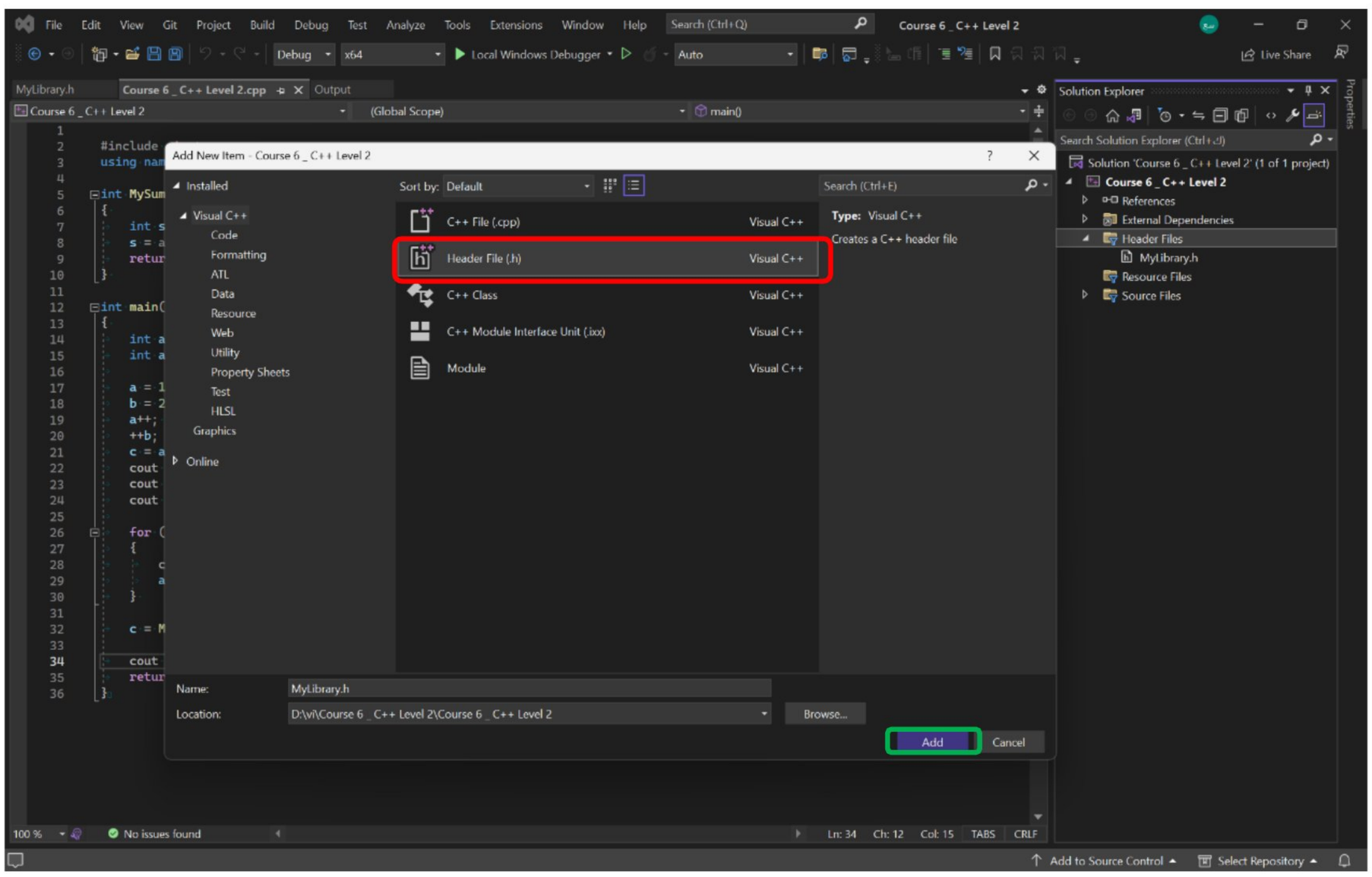
مميزات المكتبات

- 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


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Course 6 _ C++ Level 2
Code
Solution Explorer Ctrl+Alt+L
Git Changes Ctrl+0, Ctrl+G
Git Repository Ctrl+0, Ctrl+R
Team Explorer Ctrl+\, Ctrl+M
Test Explorer Ctrl+E, T
Bookmark Window Ctrl+K, Ctrl+W
Call Hierarchy Ctrl+Alt+K
Class View Ctrl+Shift+C
Code Definition Window Ctrl+\, D
Object Browser Ctrl+Alt+J
Error List Ctrl+\, E
Output Ctrl+Alt+O
Resource View Ctrl+Shift+E
Toolbox Ctrl+Alt+X
Notifications Ctrl+\, Ctrl+N
Terminal
Other Windows
Toolbars
Full Screen Shift+Alt+Enter
All Windows Shift+Alt+M
Navigate Backward Ctrl+-
Navigate Forward Ctrl+Shift+-
Next Task
Previous Task
Property Manager
Property Pages Shift+F4
cout << c;
return 0;
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Course 6 _ C++ Level 2' (1 of 1 project)
Course 6 _ C++ Level 2
References
External Dependencies
Header Files
Resource Files
Source Files
No issues found
Ln: 12 Ch: 11 TABS CRLF
Ready
Add to Source Control
Select Repository


Course 6 _ C++ Level 2
MyLibrary.h
Course 6 _ C++ Level 2.cpp
Output
Add New Item - Course 6 _ C++ Level 2
Installed
Visual C++
Code
Formatting
ATL
Data
Resource
Web
Utility
Property Sheets
Test
HLSL
Graphics
Online
Sort by: Default
C++ File (.cpp) Visual C++
Header File (.h) Visual C++
C++ Class Visual C++
C++ Module Interface Unit (.ixx) Visual C++
Module Visual C++
Type: Visual C++
Creates a C++ header file
Name: MyLibrary.h
Location: D:\vi\Course 6 _ C++ Level 2\Course 6 _ C++ Level 2
Browse...
Add
Cancel
Solution Explorer
Solution 'Course 6 _ C++ Level 2' (1 of 1 project)
Course 6 _ C++ Level 2
References
External Dependencies
Header Files
MyLibrary.h
Resource Files
Source Files
No issues found
Ln: 34 Ch: 12 Col: 15 TABS CRLF
Add to Source Control
Select Repository

في مكتبتك تستطيع إنشاء Function & Procedures لكن تحت `namespace` ثم تضع أي **اسم** خاص بها ; ثم **{ تضع فيها Function & Procedures }**

تستطيع إستدعاء مكتبات - سواء موجودة في C++ أو أنت أنشأتها - داخل مكتبتك مثل

```
#include <iostream>
```

كيف تستطيع استدعاء مكتبتك في المشروع (الملف الرئيسي)

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

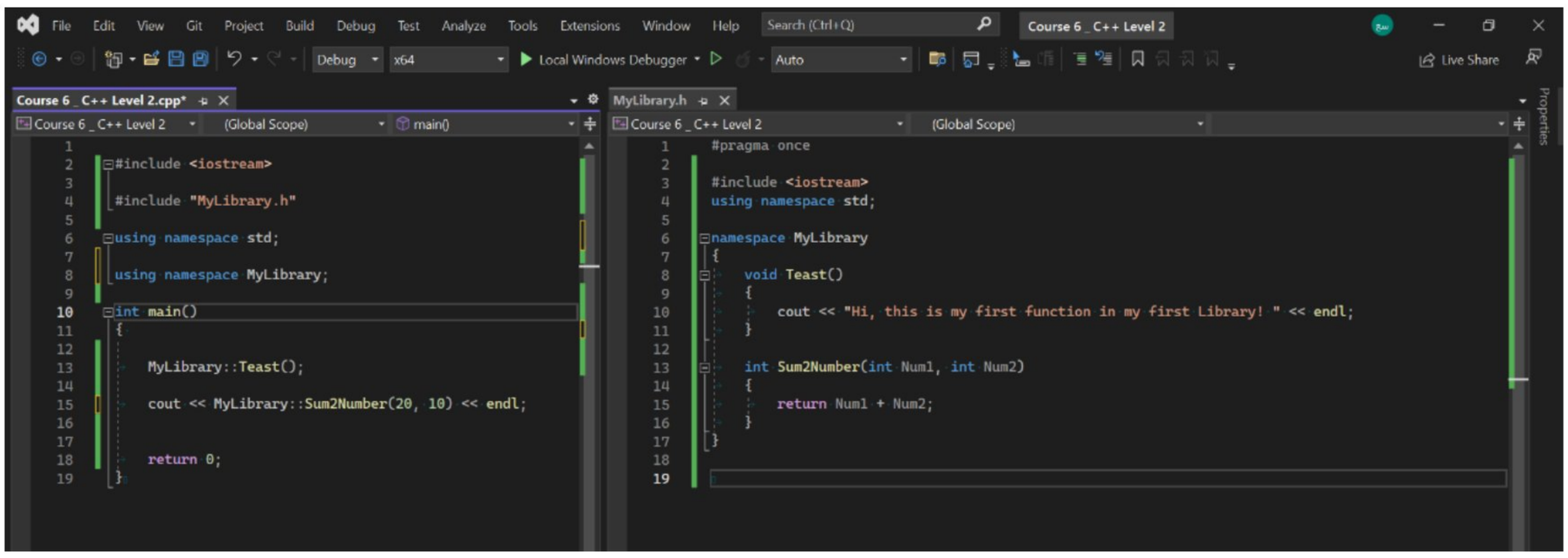
مثل

```
MyLibrary::Teast();

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

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

لكن يفضل عدم استخدام هذا الاختصار

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



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

- Ternary Operator اختصار كتابة الكود
- ليس كل شرط يعمل Ternary Operator
- Ternary Operator يرجع نتيجة عكس If else
- ? (Mark >= 50) الشرط + علامة الاستفهام ؟
- إذا تحقق الشرط : & إذا لم يتحقق الشرط ; تستطيع عمل الاتي : مثال
  - Result = "PASS";
  - cout << "PASS";
  - إضافة **Function & Procedures**
  - ? (Mark >= 50) إضافة شرط جديد

Syntax الطريقة القصيرة			
إذا تحقق الشرط	الشرط		مثال

{ تنفيذ الشرط }	(إضافة شرط) ؟	؟	`int Mark = 90;`
			`string Result;`
			`Result = (Mark >= 50 ) ? "PASS" : "FAIL";`
			`cout << Result << endl;`

{ تنفيذ الشرط }	إذا تحقق الشرط :	True :	
{ تنفيذ الشرط }	إذا لم يتحقق الشرط ;	False ;	

	Syntax الطريقة الطويلة		
إذا تحقق الشرط	الشرط		مثال

{ تنفيذ الشرط }	(إضافة شرط)	if	`int Mark = 90;`
			`string Result;`
			`if (Mark >= 50)`
			`{ Result = "PASS"; }`
			`else`
			`{ Result = "FAIL"; }`
			`cout << Result << endl;`

{ تنفيذ الشرط }	(إضافة شرط جديد ، إذا لم يتحقق الشرط الذي قبله)	else if	
{ تنفيذ الشرط }	إذا لم يتحقق أي شرط	else	

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

- **Ranged Loop** ليست بديل من for Loop العادية
- نوع النطاق (int , bool , string ...) لابد أن يكون مثل نوع **Collection**
- Ranged Loop تستخدم مثلا مع (Array , Victor , Object)
- يبدأ النطاق من أول عنصر {1} ثم العنصر الذي يليه {2} ... مثل

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

- **Ranged Loop** تستخدم مع أشياء ديناميكية مثلا (Array , Victor , Object) Collection of things

	Syntax				
مثال		تسمية المتغير **وتحديد نوعه**		**مجموعة من الأمور** Collection	
`int Array1[] = { 1,2,3,4};` `for (int n : Array1)` `{` `Cout << n << endl;` `}`	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)

- Validate التحقق أن المدخل رقم فقط
- cin هي **OOP** وليست Function (ستدرس لاحقا OOP)
- **cin.fail()** هل المدخل خطأ ؟

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

int ReadNumber()
{
	int Number = 0;

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

	while (cin.fail())
		// cin.fail() يعني 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;

		cin >> Number;
	}

	return Number;
}

int main()
{

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

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

- ❖ أحادي المعامل **Bitwise** تستخدم إشارة واحدة فقط **&**
- ❖ في المنطق Logic AND تستخدم إشارتين من **&&**
- ❖ **Bitwise &** يتم تحويل الأرقام الى Binary
- ❖ ثم يتم المقارنة بين كل Bit مع مقابلها (Logic AND &&)
- ❖ الناتج يتم تحويله الى 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 : أحادي المعامل | (Lesson #13 - Bitwise | Operator)

- ❖ أحادي المعامل **Bitwise** تستخدم إشارة واحدة فقط **|**
- ❖ في المنطق Logic OR تستخدم إشارتين من **||**
- ❖ **Bitwise |** يتم تحويل الأرقام الى Binary
- ❖ ثم يتم المقارنة بين كل Bit مع مقابلها (Logic OR ||)
- ❖ الناتج يتم تحويله الى 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 : (Lesson #14 - Declaration Vs Definition)

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

```
// 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 تعريف
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)

❖ **Default Parameters** فشل البارامتر يعني بارامتر اختياري **Option Parameter**

```
// 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;

}
```

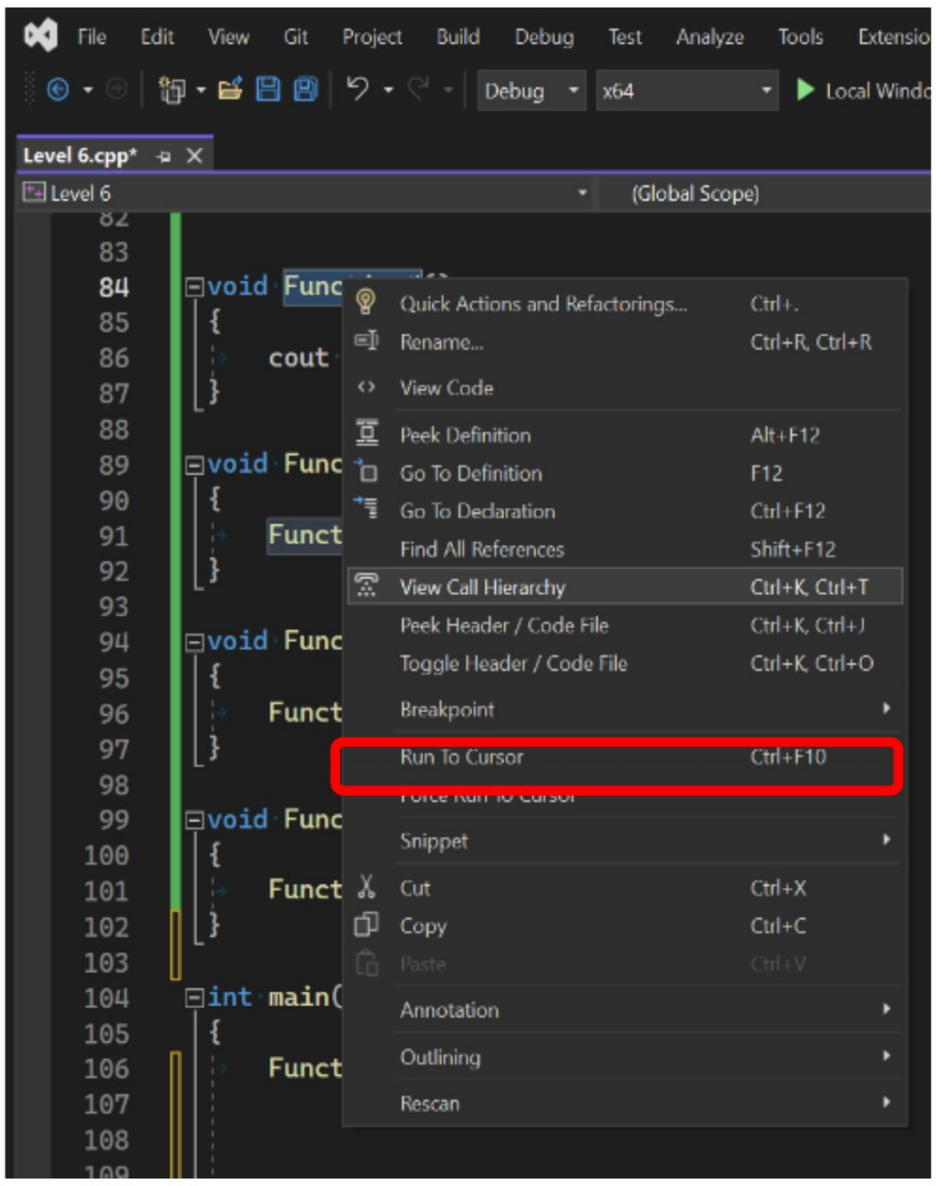
الدرس 16 : وظيفة التحميل الزائد (Lesson #16 - Function Overloading)

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

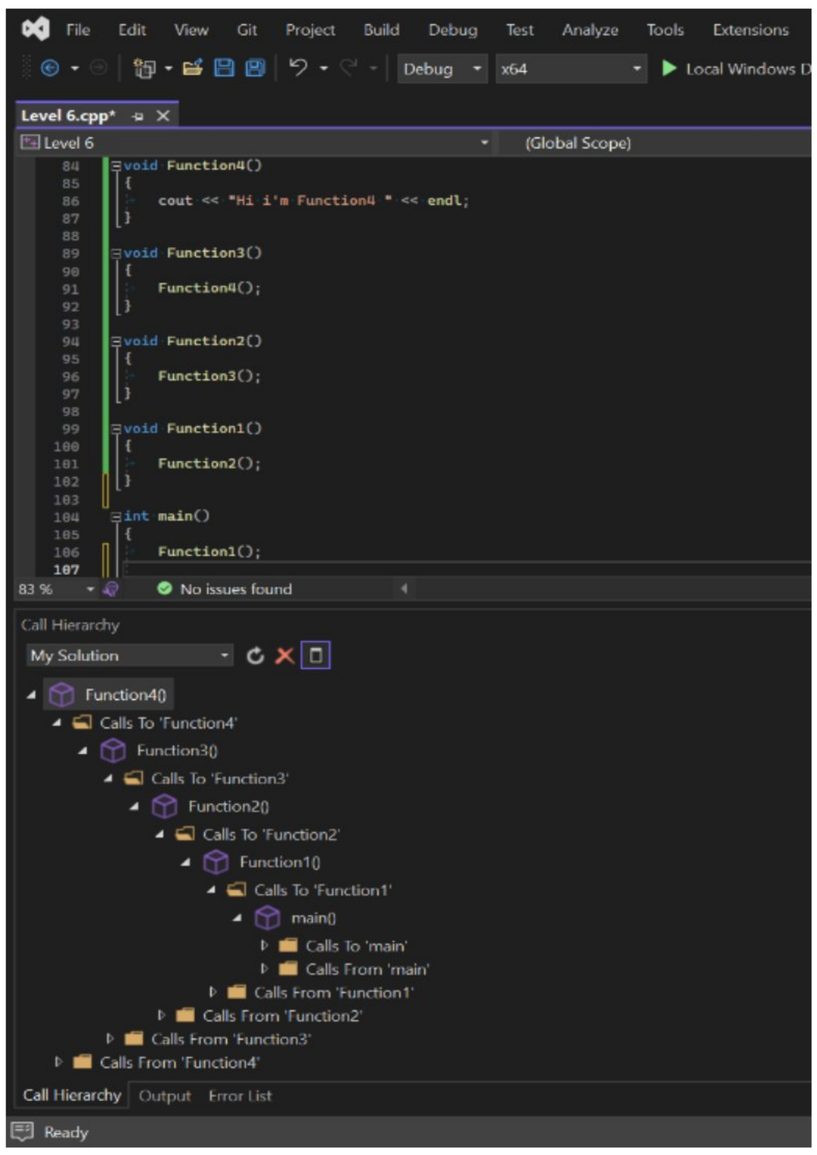
```
// اختلاف نوع البارامتر
double  MySum(double a, double b)
{
      return (a + b);
}

// 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;
}
```

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

- **Call Stack OR Call Hierarchy** هو نفس الشيء (اختلاف أسماء فقط والفعل واحد)
- **Call Stack / Call Hierarchy** سلسلة مناداة Function ل Function آخر الى أن يصل لآخر Function يتم استدعاؤه (تدرج استدعاء Function في البرنامج)
- **Active Frame** هو الإطار النشط في الذاكرة
- **Active Frame** يكون واحد فقط نشط
- أول Function يتم استدعاؤه في البرنامج هو **int main()** وآخر Function إزالة
- أول عنصر يدخل في Stack هو آخر عنصر يخرج من Stack (والعكس بالعكس)
- إضافة عنصر الى 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 بجانب اسم Function

٦. ثم اضغط على بجانب **(Call To)** + " اسم Function "

الدرس 18 : (Lesson #18 - VS Function Tips)

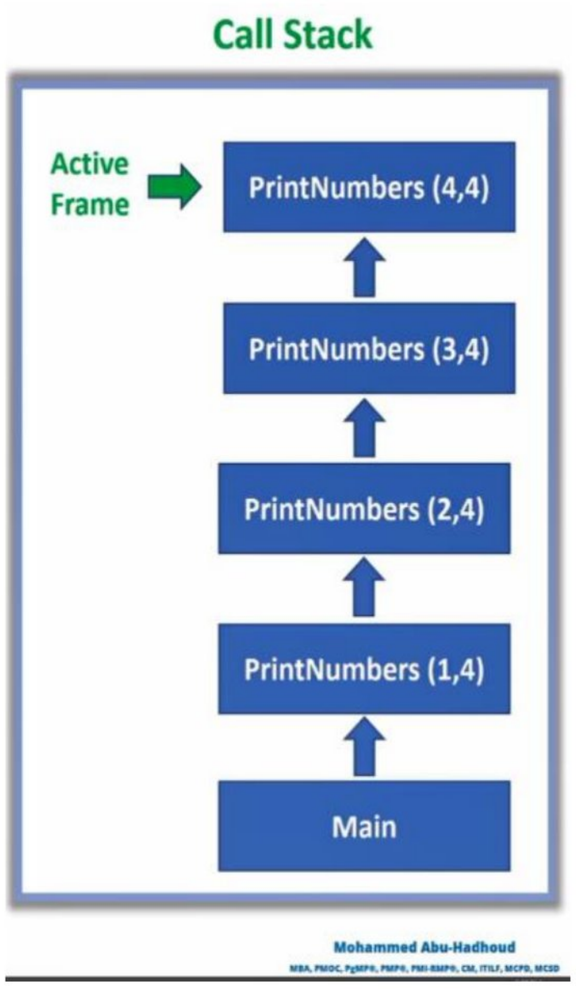
بعض (**اختصارات**) **Visual Studio**

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 **Find All References** (**Shift + F12**)
٥. نظرة خاطفة ل Function مع عدم الذهاب إليه **Peek Definition** (**Alt + F12**)
٦. لتغيير اسم Function مع جميع استدعاءاته دفعة واحدة **Rename...** (**Ctrl + R**)

الدرس 19 : استدعاء ذاتي (Lesson #19 - Recursion)

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

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

                //استدعاء Function لنفسه
                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 الأول)
- ❖ **Static تثبيت قيمة المتغير بعد الخروج من** Function**(حياته بانتهاء كامل البرنامج)**
- ❖ **Static متغير يتم الاحتفاظ بقيمته في البرنامج بأكمله**

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

        int Number = 1;

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

        Number++;
}

void MyFunctionStatic()
{
        //Function  بانتهاء  Static لا تنتهي حياة المتغير مع
        // 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 : (Lesson #22 - Register Variable)

- ❖ **Register Variable تم إلغاؤه من C++ 11 فما فوق**
- ❖ **Variable** المتغيرات يتم تخزينها بشكل تلقائي في RAM
- ❖ Register كانت تستخدم مع السرعات العالية جدا جدا جدا

أسرع وحدات التخزين من حيث تلقي الأوامر من CPU بالترتيب

١. Registers البيانات المتاحة على الفور (أقرب وحدة ل CPU)
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 (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  ,  هي تعويض بأصفار عن الأرقام الناقصة ، على حسب الأعداد (*)
        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);
}
```

الدرس 24 : تنسيق طباعة الأرقام (Lesson #24 - Float Format (Printf))

موجود على أغلب اللغات Format

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

```
int main()
{
        float PI = 3.14159265;

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

        // %.3f Round  لإظهار أرقام الخانات (3 عدد ) بعد الفاصلة مع تقريب الرقم الأخير
        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
}
```

الدرس 25 : تنسيق طباعة النصوص (Printf) Lesson #25 - String And Char Format

❖ **Printf** لا يتعامل مع **String** ، يتم تحويل String الى **Char Array**

```
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 2 1 هي للفراغات على حسب الأرقام (*)
        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 short = %d OR %0*d ");	**int** **short**
printf (" float and double = %.*f OR %.5f ");	**float** **double**
printf (" char array = %s ");	**char array** **(string)**
printf (" char array = %*c ");	**char**

الدرس 26 : تنسيق الفراغات (Lesson #26 - Setw Manipulator)

```
#include <iostream>
#include <iomanip>  // this library stored the std::setw
// هذه المكتبة لحجز 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 يتخطى 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|
---------|--------------------------------|---------|
```

❖ **Two Dimensional Arrays** الأول [للصفوف] الثاني [للأعمدة]

```
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;
        }

}
```

الدرس 28 : فيكتور(Lesson # 28: Introduction , Declaration and Initialization)

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

	Syntax		
Vectors	**< Type >**	**Name**	**Initial Value**
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 هي Vector من الطرق لطباعة
      // vector <int> لا بد أن يكون مثل int Number
// الى آخره int Number الى vector ينسخ العنصر الأول من (int Number : vNumbers)
      // عملية النسخ تأخذ وقت وحجم = برنامج بطيء
      for (int & Number : vNumbers)
      {
//  إشارة & ، عملها الذهاب الى موقع العنصر في الذاكرة (int & Number : vNumbers)
      //إشارة & : عدم نسخ العناصر
            cout << Number << " ";
      }

      cout << endl;
}
```

الدرس 29 : إضافة عناصر للفيكتور (Lesson # 29 - Add elements)

- في **Vector** يوجد **Method** اسمه **push_back** لإضافة العناصر
- عند إضافة عنصر ل **push_back(10)** يتم إضافتها في **Stack**
- **Vector** يستخدم نوع من أنواع Data Structure وهي **Stack**
- 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**
- **Array** في Parameters هي **&** By Reference
- **Vector** في Parameters هي By Value
- عند إنشاء Parameters من نوع Vector **يفضل دائما وضع إشارة &** By Reference (للتعديل على Vector الرئيسي)
- **يفضل دائما وضع إشارة &** في Ranged Loop (للذهاب الى Address وعدم نسخه

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

int main()
{
      vector <int> vNumbers ;

      // لإضافة عناصر الى vector وأخذ مساحة له بشكل أتوماتيكي
      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 الى vector ينسخ العنصر الأول من (int Number : vNumbers)
      // عملية النسخ تأخذ وقت وحجم = برنامج بطيء
      for (int & Number : vNumbers)
      {
//  إشارة & ، عملها الذهاب الى موقع العنصر في الذاكرة (int & Number : vNumbers)
            //إشارة & : الى عدم نسخ العناصر
            cout << Number << endl;
      }
      cout << "\n\n";

}
```

Homework Solution

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

// عدم وضع إشارة & للباراميتر يعني
// 1- عمل نسخة من Vector وتعبئتها ثم بعد الخروج من Function يتم تدمير هذه النسخة
// 2- لا يتم تعديل أو تعبئة على Vector الرئيسي الموجود في main()

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

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);
}
```

الدرس 30 : (Lesson # 30 - Vector of Structure)

- 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 تعبئة stEmployee ثم إضافتها الى

	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
	// عدم وضع إشارة & : يتم إنشاء نسخة ثانية من vEmployee
	// 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 أولا)
- **Vector** 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 تعبئة عناصرها -
cout << "Stack Size : " << vNumbers.size() << endl;

//... إزالة آخر عنصر دخل Stack (50)
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 pop_back() و Stack لا يوجد به عناصر : البرنامج لا يشتغل error
// للتأكد من أن Stack فاضي استخدم

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

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

// لإزالة جميع العناصر من Stack دفعة 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;
```

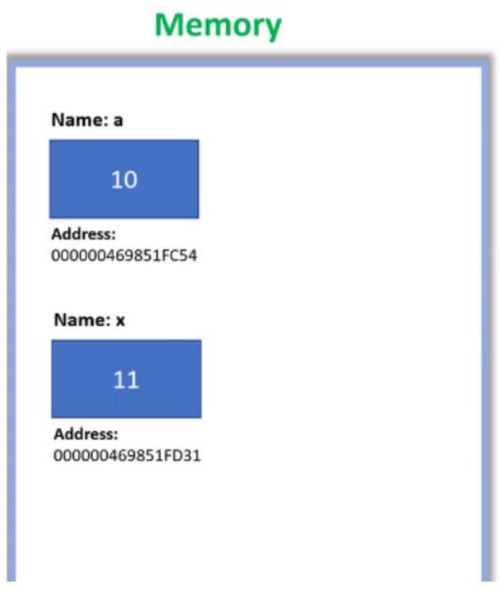
الدرس 32 : (Lesson #32 - Vector Functions)

```
#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 لو تم حذف العناصر من Stack لا تستطيع طباعة أول وآخر عنصر
        // exception الخطأ يسمى استثناء أو اعتراض
        // empty() أو size() لذا يفضل وضع شرط لأخذ بيانات من العناصر إما

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

        if (!vNumbers5.empty())
        {
                // Vector يطبع 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;
}
```

الدرس 33 : (Lesson #33 - Call By Ref / By Val Important Review)

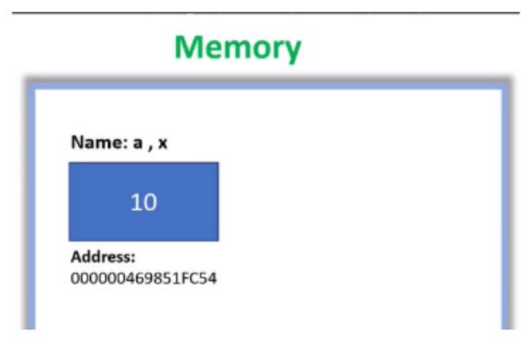
- ❖ لكل متغير يتم حجز مكان له (**Slot**) في الذاكرة يحتوي على
  - ○ اسم المتغير
  - ○ وقيمته
  - ○ وعنوانه **Address** في الذاكرة
- ❖ **Address** مكتوب بلغة Hexadecimal ويسمى **Address** أو - **Reference** -
- ❖ **Address** هو مرجع لعنوان مكان المتغير في الذاكرة
- ❖ **إشارة & تسمى Reference**

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

// By Value يسمى Parameter ( int x )
void Function1(int x)
{
        // main() يتم إنشاء نسخة أخرى من المتغير الرئيسي في  By Value
        // هذه النسخة لها اسم وقيمة وعنوان Address آخر مختلف
        // main() التغيرات داخل Function لا تؤثر على القيمة الأصلية داخل

        x++; // 10 + 1 = 11
}

// By Reference  يسمى Parameter ( int & x )
void Function2(int & x)
{
//  يتم نسخ عنوان Address من المتغير الرئيسي في main() للتعديل عليه By Reference
        // هذه لها اسم مختلف ( اسمان ) وعنوان Address واحد
        // main() التغيرات داخل Function تؤثر على القيمة الأصلية داخل

        x++; // 10 + 1 = 11
}

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

        int a = 10;

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

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

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

}
```



الدرس 34 : (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 ولهما 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 )

}
```

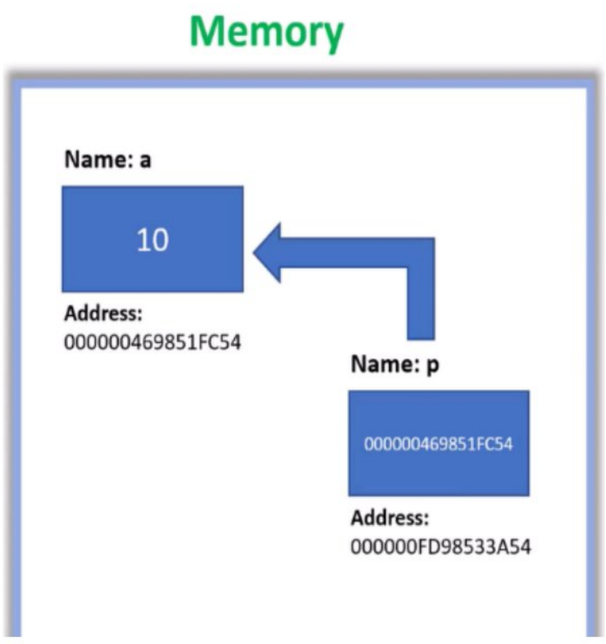


الدرس 35 : المؤشر 1 (? Lesson #35 - What is Pointer)

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

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

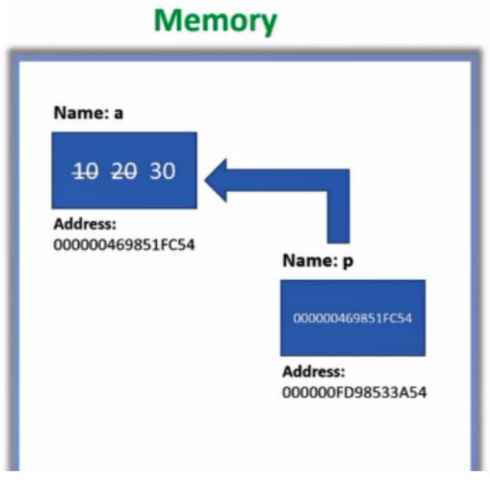
```
#include <iostream>
using namespace std;

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

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

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

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

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

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

        cout << endl;

}
```



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

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

```
#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 Pointer
        *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;

}
```

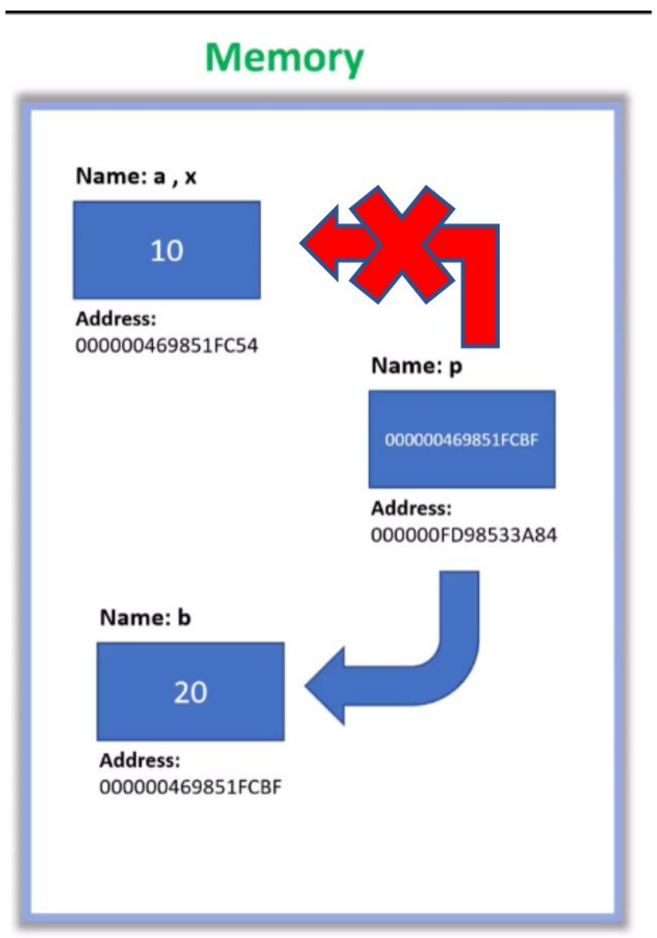


أخطاء شائعة يجب تجنبها في Pointer

```
int x = 10 , * p;

// خطأ
// Value  لا يخزن قيم P = Pointer
p = x;
p = 50;

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

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


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

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 (مرن)**

```
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 * Pointer الى متغير آخر
p = &b;

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

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



```
#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 فيحتاج * Pointer البارامیتر من نوع
swap_Poin(&s, &r);

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

)
```

الدرس 40 : المؤشر 6 (Lesson #40 - Pointers and Arrays)

- ❖ **Array** : مجموعة من المتغيرات **Variable**
- ❖ كل **Variable** له عنوان **Address**
- ❖ للوصول للقيم **Value** في **Array** = **(Name Pointer + Index)** *

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

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

        cout << "Address array : \n";

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

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

        // &arr[2] يخزن عنوان Address ثالث عنصر في ptr + 2
        cout << ptr + 2 << endl;
        // &arr[1] يخزن عنوان Address رابع عنصر في 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;

}
```

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

```
#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 التغيرات ثم بعد الضغط على إحداها يتم تحويله الى ptr.
	cout << "\n Using Pointer : \n" ;
	cout << ptr->FirstName << endl;
	cout << ptr->Salary << endl;
}
```

الدرس 42 : المؤشر 8 (Lesson #42 - Pointer to Void)

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

Syntax		
*(static_cast	< Type * >	(NamePointer))
*(static_cast	< float * >	(ptr))

```
void* ptr;

float f1 = 10.5;
ptr = &f1;

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

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

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

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

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

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

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

الدرس 43 : (Lesson #43 - Memory Management: new and delete)

- 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 Type **new** لا بد أن تستخدم مقابلها NamePointer **delete**
- **new** و **delete** تستخدم 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;
```

الدرس 44 : (Lesson #44 - Dynamic Arrays: new and delete)

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

```
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 رقم العنصر في = [ 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;
```

الدرس 45 : (Lesson #45 - Stack vs Heap)

- ❖ الذاكرة **Memory** يقصد بها في البرمجة : **RAM**
- ❖ البرنامج يقسم في **RAM** الى **أربعة أقسام** هي

١. يخزن فيه كود الأوامر **Source Code / Instruction** (يأخذ مساحة قليلة)

٢. **Static / Global** هما **Variables** تكون دورة حياتهما طوال حياة كامل البرنامج (يأخذ مساحة قليلة)

٣. **Stack** هو: الذاكرة **Memory** التي يخصصها **Operating System** نظام الشغيل للبرنامج - بعد معرفة حجمه - (يأخذ مساحة على حسب البرنامج) يخزن فيه (Local : Variables / Function / **Pointers**)

٤. يستطيع Developer الوصول الى Heap بقية الذاكرة باستخدام **Pointers** (**Pointers** يخزن Address في Stack) أما الأشياء الديناميكية تخزن في **Heap** مثل

```
ptrX = new int;
```

Heap : **any dynamic** Variables / Objects / Arrays ...etc. يخزن فيه

الدرس 46 : Vectors (Lesson #46 - Access Elements)

- ❖ **Vector** يعامل معاملة **Array**

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

// تستطيع الوصول الى أي عنصر بطريقتين

// NameVector.at(Index) الطريقة الأولى باستخدام
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;
// زيادة Index عن العناصر يعترض البرنامج على ذلك Exception ويذهب بك الى كود Vector
cout << "Element at Index 5 : " << num.at(5) << endl;

// NameVector[Index] الطريقة الثانية باستخدام
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;
// عند زيادة Index عن العناصر يحدث خطأ للبرنامج Wrong أو Garbage
cout << "Element at Index 5 : " << num[5] << endl;
```

❖ لطباعة **Vector** فقط وعدم عدم 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 : ";

// فقط Reference & 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
}
```

الدرس 48 : Vectors (Lesson #48 - Vector Iterators)

- **Iterator** : طريقة للمرور على عناصر **Vector** باستخدام **Pointer**
- لابد من تطابق نوع **Iterator** مع نوع **Vector**
- **NameVector.begin()** للوصول الى أول عنصر في **Vector**
- **NameVector.end()** للوصول الى آخر عنصر في **Vector**

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 << " ";
}
```

الدرس 49 : معالجة الاعتراض أو الاستثناء(Lesson #49 - Try Catch)

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**

- **Exception Handling** : 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 يعلّق أو يتم إغلاقه بشكل مفاجئ **Crash**
- **Exception Handling** موجود في أغلب لغات البرمجة
- **Crashes** إغلاق البرنامج بشكل مفاجئ له أحوال كثيرة أمثلة (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 **يبطّئ البرنامج**
- السطر الذي تشك أنه يعمل إغلاق البرنامج بشكل مفاجئ **Crash** ولا يتم معالجته إلا ب **Exception Handling** (try { } catch(..) { })

Syntax			
try	{ Crash }	catch (...)	{ تنفيذ الأوامر إذا حدث Crash }

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

try
{
        // Exception
        // crash السطر الذي تشك أنه يعمل إغلاق البرنامج بشكل مفاجئ

        cout << num.at(5);
}
catch (...)
{
        // Handling
        // crash تنفيذ الشرط إذا حدث

        cout << "out of bound \n";
}
```

الدرس 50 : (Lesson #50 - String Object: (Common Methods))

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

```
#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 ( الموقع 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 يجد النص يرجع رقم طويل
      if (S1.find("ali") == S1.npos)
      {
            // .npos مخزن فيه الرقم الذي يدل على الخطأ
            // S1.npos تعني لم يجد النص المطلوب
            cout << "ali is not found";
      }

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

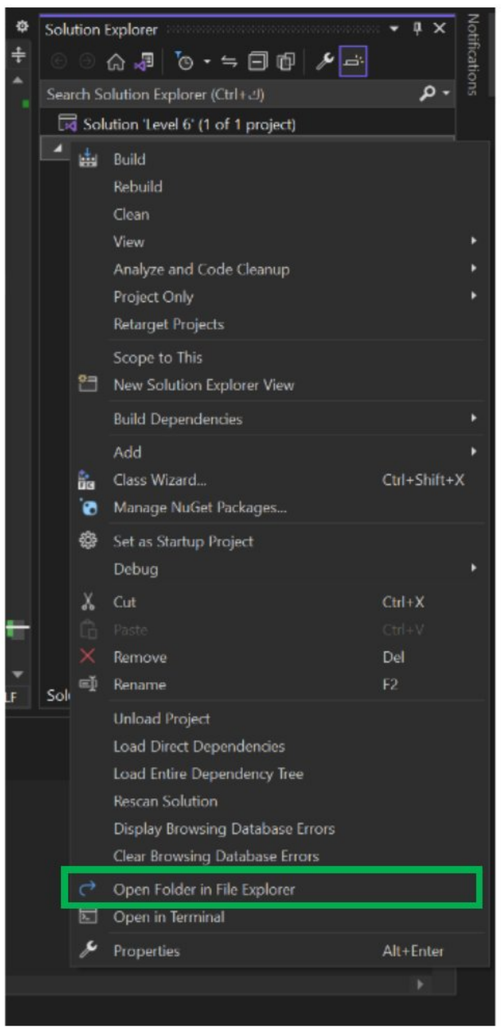
الدرس 51 : (Lesson #51 - Some CCTYPE Functions)

- ❖ استدعاء مكتبة `#include <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');

        // ASCII  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;
}
```

الدرس 52 : File 1 (Lesson #52 - Write Mode: Write Data To File)

- ❖ لابد من استدعاء مكتبة `#include <fstream>`
- ❖ يوجد أكثر من طريقة للكتابة على الملف منها **Write Mode**
- ❖ تستطيع تخزين البيانات في ملف :
  1. يمكن التعديل على البيانات
  2. إضافة بيانات أخرى
  3. حذف البيانات
- ❖ **Write Mode** كلما تعمل Run يبدأ الحفظ من جديد – يحذف البيانات السابقة –
- ❖ بعد الانتهاء من البيانات المرادة حفظها في الملف : **يجب إغلاق الملف close()**



1. اختر من الشريط العلوي (**View**)
2. ثم اختر (**Solution Explorer**) ستظهر نافذة
3. ثم اختر من النافذة (**اسم المشروع Project**) زر الفأرة الأيمن
4. ثم اختر (**Open Folder In File Explorer**)
5. سيذهب الى الملف في الهارد ديسك ثم افتح الملف

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
        fstream MyFile;
// MyFile.open("اسم الملف المخزن في الهارد ديسك. ونوعه", ios::out إنشاء ملف);
        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();
        }
}
```

الدرس 53 : File 2 (Lesson #53 - Append Mode: Append Data to File)

- **Append Mode** كلما تعمل Run يبدأ يضيف المعلومات – لا يحذف البيانات السابقة
- **ios::app** إضافة معلومات إضافية مع الاحتفاظ بالبيانات السابقة
- إنشاء ملف **ios::out** | أو إضافة معلومات **ios::app** مع إعطاء الأولوية لإضافة المعلومات

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

// MyFile.open("اسم الملف المخزن في الهارد ديسك. ونوعه", ios::out إنشاء ملف);
        // ios::app إضافة معلومات إضافية مع الاحتفاظ بالبيانات السابقة
// إنشاء ملف ios::out | أو إضافة معلومات 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();
        }
}
```

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

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

void PrintFileContenet(string FileName)
{
	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 string Line
		while (getline(MyFile, Line))
		{
			cout << Line << endl;
		}

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

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

الدرس 55 : File 4 (Lesson #55 - Load Data From File to Vector)

❖ 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> &
vFileContenet)
{
	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 string Line
		while (getline(MyFile, Line))
		{
			// كل سطر في الملف يتم إضافتها الى Vector
			vFileContenet.push_back(Line);
		}

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

int main()
{

	vector <string> vFileContenet;

	LoadDataFromFileToVector("MyFile.txt", vFileContenet);

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

	return 0;
}
```

الدرس 56 : File 5 (Lesson #56 - Save Vector to File)

❖ 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 SaveVectorToFile(string FileName , vector <string>  vFileContenet)
{
        // عدم وضع إشارة & للفيكتور في البارامتر : لأنه للقراءة فقط وعدم التعديل عليه

        fstream MyFile;

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

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

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

int main()
{

        vector <string> vFileContenet{"Saeed" , "Omar" , "Rowida" ,
"Saleh" , "Mohammed" , "Khaled" };

        SaveVectorToFile("MyFile.txt", vFileContenet);


        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> &
vFileContenet)
{
	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 string Line
		while (getline(MyFile, Line))
		{
			// كل سطر في الملف يتم إضافتها الى Vector
			vFileContenet.push_back(Line);
		}

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

void SaveVectorToFile(string FileName , vector <string>  vFileContenet)
{
	// 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 Run
	MyFile.open(FileName, ios::out); // Write Mode

	// للتأكد من أن الملف يعمل
	if (MyFile.is_open())
	{
		// وضع إشارة & لعدم إنشاء نسخة أخرى - الذهاب الى Address -
		for (string& Line : vFileContenet)
		{
			// للتأكد 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 Vector
        for (string& Line : vFileContent)
        {
            // حذف البيانات
            if (Line == Record)
            {
                Line = "";
            }
        }

        SaveVectorToFile(FileName, vFileContent);

}

void PrintFileContenet(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 string Line
            while (getline(MyFile, Line))
            {
                cout << Line << endl;
            }

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

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

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

        cout << "\nFile Content After Delete. \n";
        PrintFileContenet("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> &
vFileContenet)
{
	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 string Line
		while (getline(MyFile, Line))
		{
			// كل سطر في الملف يتم إضافتها الى Vector
			vFileContenet.push_back(Line);
		}

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

void SaveVectorToFile(string FileName , vector <string>  vFileContenet)
{
	// 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 Run
	MyFile.open(FileName, ios::out); // Write Mode

	// للتأكد من أن الملف يعمل
	if (MyFile.is_open())
	{
		// وضع إشارة & لعدم إنشاء نسخة أخرى - الذهاب الى Address -
		for (string& Line : vFileContenet)
		{
			// للتأكد 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 Vector
        for (string& Line : vFileContent)
        {
            // تبديل البيانات
            if (Line == Record)
            {
                Line = UpdateTo;
            }
        }
        SaveVectorToFile(FileName, vFileContent);

}

void PrintFileContenet(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 string Line
            while (getline(MyFile, Line))
            {
                cout << Line << endl;
            }

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

int main()
{

        cout << "File Content Before Update. \n";
        PrintFileContenet("MyFile.tet");

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

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

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

- ❖ لابد من استدعاء مكتبة `#include <ctime>`
- ❖ هذه المكتبة مأخوذة من لغة C `#include <ctime>`
- ❖ 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()
{
      // التوقيت المحلي
      time_t t = time(0);  // get time now

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

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

      }
```

الدرس 60 : (Lesson #60 Datetime Structure)

- ❖ لابد من استدعاء مكتبة `#include <ctime>`
- ❖ هذه المكتبة مأخوذة من لغة C `#include <ctime>`
- ❖ 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

        // tm* now هو struct يحتوي على العناصر التي بالأسفل
        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;

        }
```