



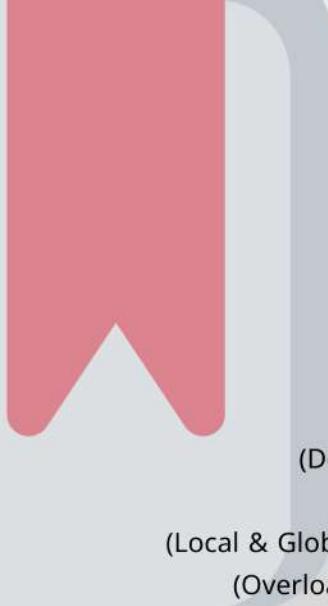
COUT<< "CODE";

```
gth;C++)  
    .push(a[c]);  
    action h() {  
        for  
        iser_logged").a(), a  
        lace(/ +(?= )/g, "",  
        b = [], c = 0;c < a.  
        0 == r(a[c], b) &  
        c = {};  
        = b.length -  
    }  
    var.n, r={l  
    nivigato  
    toS
```



ملخص لغة البرمجة C++

الفهرس



21 الدوال (Functions)

- تعريف الدوال (Declaration function)
- الاستدعاء الذاتي (Recursion)
- (Local & Global variable) المتغير المحلي والعام
- دالة التحميل الزائد (Overloading function)
- المتغير الافتراضي (Default argument)
- دوال المصفوفة (Array function)
- تعبير لامدا (Lambda expression)

1 أساسيات اللغة (Basics of language)

- مقدمة عن اللغة (Introduction)
- واجهة البرنامج (The interface)
- التعليقات (Comments)
- الأحرف الهاربة (Escape Sequance)
- النطاق (Scope)

29 تعليمات الذاكرة (RAM instructions)

- المؤشرات (Pointers)
- علاقة المؤشر بالمصفوفة (Pointers with array)
- الدالة المضمونة (Inline)
- المتغير الثابت والمستعار والساكن (Constant & Aliasing & Static)
- الاستدعاء بالقيمة والاستدعاء بالعنوان (Call by value & by refrence)

34 السلاسل النصية (string)

13 الحلقات التكرارية (Loops iteration)

- تعريف الحلقات (Defintion of Loops)
- أنواع الحلقات (Types of loops)
- أدوات التحكم (Control statement)
- حلقة الملانهاية (Infinity loop)

37 معالجة الاستثناء (Exception Handling)

38 الرقم العشوائي (Random number)

39 مكتبة الرياضيات (Math library)

16 المصفوفات (Arrays)

- المصفوفات ذات البعد الواحد (1D)
- المصفوفات ذات البعدين (2D)
- مصفوفات الحروف (Array of character)

أساسيات اللغة (Basics of language)

■ مقدمة عن اللغة (Introduction)

- تعتبر لغة كائنية التوجه، تدعم البرمجة الشيئية (OOP) وهي اختصار للجملة (Object Oriented Programming)
- تستخدم في كثير من المجالات مثل برمجة تطبيقات سطح المكتب، برمجة الألعاب، برمجة أنظمة التشغيل (OS)، برمجة متصفحات الانترنت، وتطوير لغات برمجية جديدة.

■ واجهة البرنامج (The interface)

- الملف الذي يدعم لغة C++ يكون امتداده .cpp، حيث يمكنك من كتابة وتنفيذ الأوامر البرمجية عليه

واجهة البرنامج عندما
تنشئ ملف امتداده
.cpp جديد

```

main.cpp x
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     cout << "Hello world!" << endl;
8
9 }
10

```

1] #include<iostream>

include : (مكتبة أساسية تدعم الادخال والاخراج في اللغة)

i : input (مدخلات)

o : output (مخرجات)

stream : عرض

2] using namespace std;

تحتوي على

دالة الادخال

(تمكن المستخدم من إدخال المطلوب منه)

دالة الطباعة

(يظهر المطلوب على شاشة المستخدم مطبوعاً)

تكتب هكذا

`cin>>x;`

تكتب هكذا

`cout<<"Hello World;"`

يمكن حذف

`using namespace std;`

من الواجهة وكتابتهم هكذا

`std :: cin>>x;`

`std :: cout<<"Hello World;"`

3] in main ()

هي الدالة الرئيسية في البرنامج دائمًا يبدأ تنفيذ التعليمات البرمجية من عندما عند تشغيل البرنامج، ولا يمكن تنفيذ أي أمر برمجي إلا إذا كان بداخلها أو بواسطتها

يتم اعتبار أي شيء داخل القوسين تابعًا لها

4] return 0;

• تستخدم لإرجاع آخر قيمة تم الوصول إليها

• تُرجع قيمة واحدة فقط مثل 0 ويمكنك تغييرها إلى أي قيمة أخرى مثل 1;

• لا يقرأ الكومبيوتر أي تعليمات برمجية بعدها

• يمكنك كتابة `()` بدلاً من `return` ووضع أي قيمة داخل القوسين مثل `(0)` مثل

الفرق بين `exit` و `return`، أن `exit` لا تُرجع قيمة مثل `return` ، لذلك لا يمكنك طباعة آخر قيمة، وضعتها في `exit` مثلا، على عكس `return`

- هي عبارة عن ملاحظات (notes) لا يستطيع الكومبيوتر قراءة أو تنفيذ ما بداخليها
 - 1] ملاحظة لسطر واحد //
 - 2] /* ملاحظة لأكثر من سطر */

الأحرف الهاربة (Escape Sequance) ■

- cout<<"Hello\nWorld;" → سطر جديد
Hello
World
- cout<<"Hello\tWorld;" → مقدار Tab بين الكلمات = 8 فراغات
Hello World
- cout<<"Hello\0World;" → لن يتم طباعة أي شيء بعد ١٥
Hello
- cout<<"Hello\aWorld;" → اصدار صوت بيبي عند الوصول للسطر وتنفيذه
- cout<<"Hello\bWorld;" → حذف حرف واحد قبل \b
HellWorld
- cout<<"Hello\rWorld;" → حذف ما قبل \r حسب عدد حروف ما بعدها
 - 5 5
حروف حروف

- يُرمز له بهذه القوسين { } كل ما بداخل القوسين يكون تابع لنفس المجال المحدد
- في حال كنت خارج حدود المجال، لا يسمح لك بالوصول لما بداخل القوسين
- يستخدم النطاق في المواضيع الآتية : الدوال (functions)، جملة (if)، جملة الدوران (loops)، المتغير المحلي والعام (local & global variable) و جملة (Exception handling)، معالجة الاستثناء (switch)

```
int main ()
```

```
{
```

```
}
```

كل التعليمات تُكتب داخل هذين القوسين، ولا يمكنك الوصول للتعليمات أو تنفيذها إن كتبتها خارج القوسين

• خصائص النطاق :

- 1] يستخدم لفصل التعليمات البرمجية عن بعضها البعض
- 2] يساعد في ترتيب الكود الطويل وجعله أكثر تنظيماً
- 3] كل التعليمات البرمجية يجب أن تكتب داخل قوسين الدالة الرئيسية، ما عدا الدوال (functions) يمكن كتابتها خارج الدالة الرئيسية

المتغيرات (Variables)



■ تعريف المتغيرات (Declaration)

• مكونات المتغير:

1] نوع البيانات : هو نوع القيمة التي سيخزنها المتغير (شكلها) يتكون من 4 أنواع :

short / int	float / double	bool	char
↓ للأعداد الصحيحة short x = 5; int y = 5;	↓ للأعداد العشرية والصحيحة float x = 5.6; double y = 5.6;	↓ للأعداد المنطقية bool x = false;	↓ للحروف char x = "A";

2] اسم المتغير : هو الاسم الذي تُعطيه للقيمة التي سيخزنها المتغير

مثال : القيمة 5 سيكون اسمها في الذاكرة x

• شروط عن تسمية المتغيرات :

لا يجوز وضع رموز لو أرقام قبل الاسم مثل int 10x
ولكن يسمح التسمية برمزي (-) و (\$) مثل int \$x

أن لا يكون اسم المتغير كلمات محفوظة باللغة (keywords) مثل int return

أن لا يحتوي على فراغ مثل int na me

أن لا تقوم بتعريف متغيرين بنفس الاسم في نفس المجال

[3] قيمة المتغير : هي القيمة التي يخزنها المتغير داخله، تكون إما عدديّة، أو منطقية، أو نصيّة

int x = 5;

نوع المتغير اسم المتغير قيمة المتغير

int x;	متغير فارغ (empty variable)
int x = 5;	اسناد قيمة (initialization)
int x = 0;	قيمة ابتدائية (initial value)

• ادخال وطباعة المتغيرات :

يمكن طباعة قيمة المتغير فقط على شاشة المستخدم عن طريق استخدام اسم المتغير بهذا الشكل

float x = 6.7;
cout << x << endl;

سيطبع 6.7

يمكن الطلب من المستخدم ادخال قيمة المتغير عن طريق تحديد نوع القيمة التي سيدخلها عند تعريف المتغيرات مثل

char x;
cin >> x;

سيدخل حرف

■ تحويل المتغيرات (Converting)

- هو تحويل نوع بيانات قيمة المتغير لنوع بيانات آخر
- يمكننا استخدام 3 طرق للتحويل :

1] الطريقة الأولى : cout<< (اسم المتغير) نوع بيانات جديد <<

```
double x = 5.5;
cout<<int(x);
```

تعريف متغير اسمه x ونوع بياناته double
تحويل المتغير x من النوع double للنوع int

2] الطريقة الثانية : cout<< (اسم المتغير) (نوع بيانات جديد) <<

```
double x = 5.5;
cout<<(int) x;
```

تعريف متغير اسمه x ونوع بياناته double
تحويل المتغير x من النوع double للنوع int

3] الطريقة الثالثة : cout << static_cast<> (نوع بيانات جديد) << (اسم المتغير) ;

```
double x = 5.5;
cout<<static_cast<int>(x);
```

تعريف متغير اسمه x ونوع بياناته double
تحويل المتغير x من النوع double للنوع int



■ العمليات الحسابية (Math operations)

- هي العمليات التي تستخدم في الرياضيات من أجل الحساب
- تتكون من 6 عمليات أساسية في الحساب :

يتم تنفيذ أولويات العمليات من اليسار لليمين حسب ترتيب أولويتها :

- 1] الأقواس
- 2] الضرب والقسمة وباقى القسمة
- 3] الجمع والطرح

الرمز $(++)$ يعني $x + 1$
الرمز $(--)$ يعني $x - 1$

2] النوع (postfix)

$x++$ أو $--x$

1] النوع (prefix)

$++x$ أو $--x$

■ مفهوم المفهوم (Assignment operators)

2] النوع (postfix)

`cout << x++;`

`int x = 0;`

يطبع قيمة x وهي صفر لأنه يراها
أولا ثم يخزن 1 في الذاكرة

1] النوع (prefix)

`cout << ++x;`

يجمع 1 لقيمة x ويطبع ناتج
الجمع وهو واحد

الشروط (Conditions)

■ العمليات المنطقية (Logic operation)

- هي العمليات التي نستخدمها للمقارنة وانشاء الشروط بين المتغيرات

$x > y$

الشكل العام للشرط
كأنك تسأل هل المتغير x أكبر من المتغير y ؟



- هناك 6 أشكال للعمليات المنطقية وهم :

هل أكبر من ? > [1]

هل أكبر من أو يساوي ? >= [2]

هل أصغر من ? < [3]

هل أصغر من أو يساوي ? <= [4]

هل يساوي ? == [5]

هل لا يساوي ? != [6]

- يوجد عمليات خاصة بالشروط، تستخدم مع الشروط بعد انشائها :

1] الشرط الأول & الشرط الثاني

إذا تحقق كلا من الشرطين سينفذ العمليات

2] الشرط الأول || الشرط الثاني

إذا تحقق أحد الشرطين سينفذ العمليات

3] (الشرط) !

يعكس نتيجة الشرط

- تستخدم من أجل التحكم بتنفيذ التعليمات البرمجية حسب شروط معينة، في حال تحقق الشرط الذي وضعته سينفذ التعليمات وإن لم يتحقق لن ينفذها
- تقسم لثلاثة أنواع :

1] جملة (if) البسيطة ← هو النوع الأساسي والرئيسي لجملة if

→ : (الشرط) if

{ تعليمات برمجية خاصة بشرط if في حال تحقق شرط if سينفذ تعليماتها

→ : (الشرط) else if

{ تعليمات برمجية خاصة بشرط if إن لم يتحقق شرط if سيذهب لها فإن تتحقق شرط else if سينفذ تعليماتها

→ else :

{ تعليمات برمجية خاصة في else إن لم يتحقق شرط else if سينفذ تلقائيا تعليمات else

2] جملة (if) القصيرة ← هو نوع مختصر أكثر للنوع الرئيسي

; تعليمات (else) : تعليمات (if) ?(الشرط)) الشكل العام لكتابتها

3] جملة (if) المتداخلة ← هو نوع يمكنك من كتابة النوع الرئيسي داخل بعضه البعض، ويكون من نوعين جمل if داخليه وجمل if خارجية

جملة (if)
داخلية

جملة (if) المتداخلة ←
 { if : (الشرط)
 else if : (الشرط)
 else :
 else :

جملة (if)
خارجية

if : (الشرط)

esle if : (الشرط)

esle :



: (الشرط)

if : (الشرط)

if : (الشرط)



ملاحظة

في حال كانت جملة if بهذا الشكل فإنه سيتحقق من الشروط بالترتيب، وسينفذ أول (واحدة) منها حتى تتحقق الشرط فقط

في حال كانت جملة if بهذا الشكل فإنه سيتحقق من الشروط بالترتيب، وسينفذ (كل) اللواتي حقق الشرط

(switch statement) (switch) ■

- تستخدم لتنفيذ التعليمات البرمجية حسب تحقق الشرط
- ما يميزها عن if أننا نستعملها في حال كانت التعليمات البرمجية كثيرة ضمن شرط واحد، لأنها مختصرة أكثر من if

switch (الشرط)

{ case (value)

تعليمات برمجية خاصة في 1

break;

الشكل العام لكتابتها



case (value2)

تعليمات برمجية خاصة في 2

break;

default :

} تعليمات برمجية خاصة في

هي عبارة عن نتيجة الشرط : value

int X = 5 ;

switch(X == 5)

مثلاً :

{ case (2) :

cout<<"Hi 2";

case (5) :

cout<<"Hi 5";

default :

cout<<"Nothing";

سينفذ هذا



• خصائص جملة (switch) :

يمكنك كتابة عدد ما تريده من case

قيمة value دائماً يجب أن تكون عدد صحيح

يسمح أن تكون نتيجة الشرط عدد كسري، سيقربها تلقائياً لأقرب عدد صحيح

`int x = 5.5;`

`switch (x);`

سيقربها لأقرب عدد صحيح وهو 5
ثم ينفذ الـ case(5) في حال وجدت

في حال وضعت الشرط رقم مفرد بدون متغير يجب أن يكون هذا الرقم صحيح

`switch (5);`

سينفذ الـ case(5) في حال وجدت

يمكنك جعل الشرط وقيمة value حروف بدلاً من أرقام وسينفذها بشكل طبيعي

`switch ('A');` ← الشرط

{
 case('A'):
 cout<<"I am A";
 } ← سينفذها

case('B'):
 cout<<"I am B";

default :

cout<<"Nothing";

في حال لم يتحقق الشرط لأي case دائمًا سينفذ تعليمات default فمبدأ عمل default هو نفس مبدأ عمل else

يمكنك جعل default فارغة بدون تعليمات برمجية وذلك بوضع فاصلة منقوطة جانبها مثل ; default:

الحلقات التكرارية (Loops iteration)

■ تعريف الحلقات (Defintion of Loops)

- تستخدم لتكرار تعليمات برمجية لعدد معين من المرات، فبدلاً من كتابة سطر برمجي 5 مرات مثلاً، يمكنك تكرار السطر البرمجي الواحد 5 مرات باستخدام الحلقات التكرارية
- لإنشاء أي حلقة تكرارية تحتاج لكتابه 3 أجزاء معًا، وهذه الأجزاء هي :

: [1] عداد (Counter)

تعريف متغير يحتوي على عدد مرات التكرار (وهنا التكرار سيكون 5 مرات)

نحدد فيه عدد مرات التكرار ←

int x = 5;

: [2] شرط (Condition)

نحدد فيه متى سيتوقف التكرار؟ ←

x == 10;

x تصبح 10

: [3] زيادة أو نقصان (Increment or Decrement)

نحدد فيه القفزات بين كل تكرار؟ ←

x++

أنواع الحلقات ■ (Types of loops)

- تكون جمل الدوران من ثلاثة أنواع، كل نوع يحمل الثلاث أجزاء التي ذكرناها
- الأنواع هي :

(3)

Do while loop

do

{

تعليمات برمجية
زيادة أو نقصان

}

while(الشرط);

مثـل
↓

```
int x = 5;
do { cout<<"I am while";
x++; }
while(x == 10)
```

(2)

For loop

(;الزيادة ;الشرط ;العداد)

{

تعليمات برمجية

}

مثـل
↓

```
for (int x = 5; x==10; x++)
{ cout<<"I am for"; }
```

(1)

While loop

العداد

while(الشرط)

{

تعليمات برمجية
زيادة أو نقصان

}

مثـل
↓

```
int x = 5;
while(x == 10)
{ cout<<"I am while";
x++; }
```

تعد **for loop** أكثر الأنواع شيوعا في الحلقات لسهولة كتابتها. كما أنه توجد حلقات متداخلة، نفس جملة (**if**) المتداخلة، تكون الأنواع داخل بعضها البعض بحلقات داخلية وخارجية.



أدوات التحكم (Control statement) ■

break ستتوقف عن تكرار أي تعليمات برمجية عندما $x = 2$ وتجاهل الشرط في الحلقة

أما continue ستتخطى التعليمات عندما $x = 2$ وتعود لتكمل التكرار حتى ينتهي الشرط في الحلقة

- هي أدوات تستخدم في التحكم بتكرار الحلقات

- تقسم لنوعين وهما :

: break [1]

يكسر التكرار عندما يتحقق شرط break، وبعد الكسر تتوقف الحلقة عن التكرار تماماً

أداة break في الحلقات هي نفسها في جملة switch (أيضاً

if (الشرط) break;

الشكل العام لكتابتها في الحلقات

مثلاً
if ($x == 2$) break;

: continue [2]

يتخطى التكرار عندما يتحقق شرط continue ويكمel التكرار لما بعده

if (الشرط) continue;

الشكل العام لكتابتها في الحلقات

مثلاً
if ($x == 2$) break;

حلقة الملانهاية (Infinity loop) ■

- هو تكرار التعليمات البرمجية في الحلقات لعدد لا نهائي من المرات

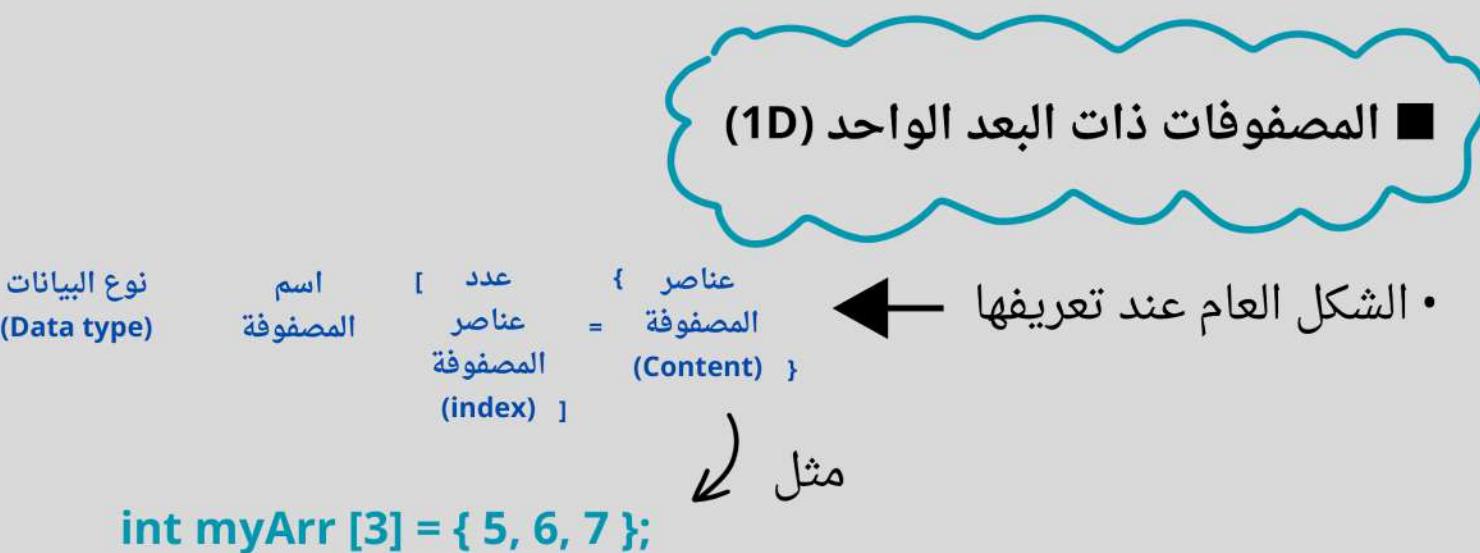
لاستخدامه في while و do mwhile نحذف الشرط ونستبدلها بكلمة true

ولاستخدامه في for نحذف الزيادة والنقصان

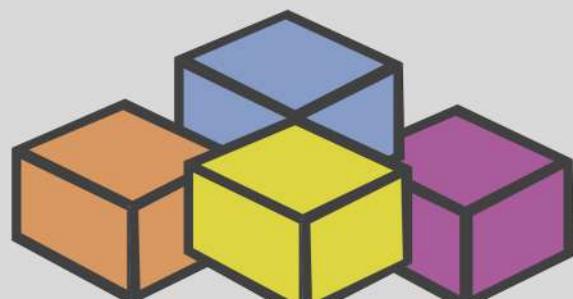
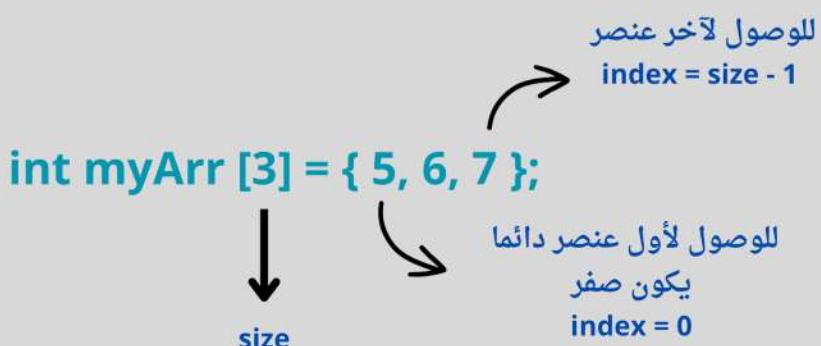
for (int x = 0; x < 5;)

المصفوفات (Arrays)

- # هي ترتيب مجموعة من المتغيرات داخل مصفوفة واحدة منظمة
- # تسهل علينا المصفوفات التعامل مع المتغيرات بسلاسة أكثر، فبدلاً من أن نتعامل مع المتغيرات بأسماءها، نتعامل مع قيم المتغيرات حسب مكان موقعها في المصفوفة وموقع المصفوفة يسمى (index)
- # تقسم المصفوفات لثلاثة أنواع حسب حجمها ونوع البيانات فيها



- الوصول موقع (index) العنصر الأول والعنصر الأخير ثابت دائماً بهذا الشكل :



- الوصول للعناصر جميعها يكون باستخدام اسم المصفوفة [موقع العنصر]

`int myArr [3] = { 5, 6, 7 };`



نقوم بالعد بالترتيب بدءاً من العنصر الأول
صفر للوصول لباقي العناصر

الوصول للعنصر الثاني `myArr [1];`
وهو الرقم 6

- عند استخدام الحلقات مع المصفوفات، حتى نصل لجميع العناصر نجعل شرط الحلقات أقل من size أو يساويه

■ المصفوفات ذات البعدين (2D)

• الشكل العام عند تعريفها ←

نوع البيانات (Data type)	اسم المصفوفة	[عدد الصدوف]	[عدد الأعمدة]	=	{ عناصر المصفوفة (Content) }
-----------------------------	--------------	----------------	-----------------	---	---------------------------------

`int myArr [2] [3] = { { 5, 6, 7 },
{ 8, 9, 10 } };`

مثـل

عدد عناصر المصفوفة
= (content)
عدد الأعمدة × عدد الصدوف

- الوصول للعناصر جميعها يكون باستخدام اسم المصفوفة [موقع العنصر]

`int myArr [2] [3] = { { 5, 6, 7 }, → myArr [0][1];
{ 8, 9, 10 } };`

تعني الصف الأول والعمود الثاني
سيكون الجواب العنصر = 6

خصائص مشتركة بين مصفوفة (1D) ومصفوفة (2D) :

1] في حال حجزنا بال size عناصر ووضعنا عنصر واحد فقط داخل content فإنه سيعتبر الباقي أصفار

```
int myArr [3] = { 5 };
int myArr [2] [3] = { 5 };
```

سيكون دائماً أول عنصر للمصفوفة = 5
أما بقية العناصر سيحجزها بأنها = 0

2] للوصول لجميع العناصر في المصفوفة (1D) مرة واحدة نستخدم نوع من الحلقات (loops)، بينما للوصول لجميع العناصر في المصفوفة (2D) نستخدم الحلقات المتداخلة (Nestead loops)

3] في المصفوفة (1D) يمكن أن يكون size فارغاً وسيعرف البرنامج لوحده عدد العناصر، بينما في المصفوفة (2D) يمكن أن يكون size عدد الصفوف فقط فارغاً وسيعرف لوحده عدد الصفوف

مثـل

```
int myArr [ ] = { 5, 6, 7 };
```



size فارغ
في مصفوفة 1D

```
int myArr [ ] [3] = { { 5, 6, 7 } ,
```



size الصدوف فارغ
في مصفوفة 2D

4] لا يُسمح بإضافة عناصر للمصفوفة عددها أكثر من size لكلا المصفوفتين

```
int myArr [ 2 ] = { 5, 6, 7 };
```

مثـل

```
int myArr [ 1 ] [3] = { { 5, 6, 7 } ,
```

```
 {8, 9, 10} };
```



■ مصفوفات الحروف (Array of character)

• هي مثل أنواع المصفوفات السابقة ولكن تكون عناصر المصفوفة نصوص ورموز

`char` العناصر { عدد العناصر [] اسم المصفوفة = size] } النصية

ممثل ↴

`char My_arr [5] = { "Lama" };`

• في مصفوفة الحروف نحسب عدد العناصر (size) = عدد الحروف + NULL

اسم Lama يتكون من 4 أحرف
وال size = الأحرف (4) + الفراغ (1)
size = 5
سيصبح

↳ `char My_arr [5] = { "Lama" };`

• العناصر النصية (content) تكون بشكلين :

أو `\0` NULL

تعني فراغ ومع كل نص في لغة C++

يتم اعتبار فراغ للنص كأنه حرف

أحرف مفردة ↴
 { 'L', 'a', 'm', 'a', '\0' }
 لا نضيف شيء
 نضيف \0 أو NULL في النهاية

• خصائص مصفوفة الحروف :

في حال وضعت عدد الأحرف = size فإنما سيعتبر آخر حرف هو الفراغ (NULL) ولن تصل للأحرف كلها

في حال كانت العناصر < size لا تضفي NULL أو \0 في الأحرف المفردة

يمكن ترك الـ size فارغاً للنص الكامل وسيعرف لوحده عدد العناصر، أما لو تركت الـ size فارغاً للأحرف المفردة فسيطبع رموز عشوائية مع الأحرف

إذا أردت من المستخدم (user) إدخال نص في مصفوفة الحروف استخدم :

`cin.get(اسم المصفوفة, size)`

إذا أردت من المستخدم (user) إدخال نصوص متعددة لأكثر من مصفوفة

حروف استخدم

`cin.get(اسم المصفوفة الأولى, size)`

`cin.get(اسم المصفوفة الثانية, size)`

في حال أردت الوصول للعناصر أو طباعتها أو إدخالها يتم استخدام الحلقات (loops) مثل أنواع المصفوفات الأخرى



الدوال (Functions)

تستخدم لترتيب وتنظيم الأكواد، من أجل تسهيل التعامل مع الأسطر البرمجية عند تعديلها، أو اصلاح الأخطاء (debugging) وغيرها ... الخ

تختلف عن أي قاعدة في C++ من ناحية أن الدوال تكتب خارج الدالة الرئيسية (int main) ولكن تنفيذ الدالة يكون مثل أي قاعدة داخل (int main)

في أي دالة يجب أن تحتوي على جميع أجزاء الدوال ونوع واحد من أنواع الدوال

■ تعريف الدوال (Declaration function)

- لتعريف الدوال تحتاج التعرف إلى قسمين أولاً، وهم :

[2]

أجزاء الدوال



Call function

هي الاستدعاء للدالة وب بواسطتها يتم فيها تنفيذ التعليمات البرمجية للدالة

(دائما تكون int داخل (main

Prototype

Body Function فيها جميع التعليمات البرمجية للدالة

(دائما تكون int خارج (main

[1]

أنواع الدوال



دالة بلا ارجاع

- نستخدم فيه أي النوع (void) فقط

- نستخدمه لتعليمات برمجية لا تتوقع منها نتيجة مختلفة مثل طباعة النصوص

- لا نستخدم معها return

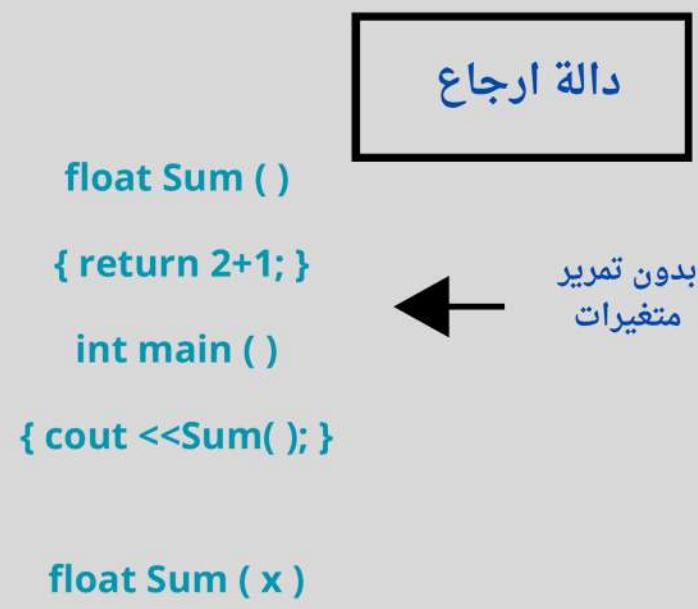
دالة ارجاع

- نستخدم فيه أي نوع بيانات

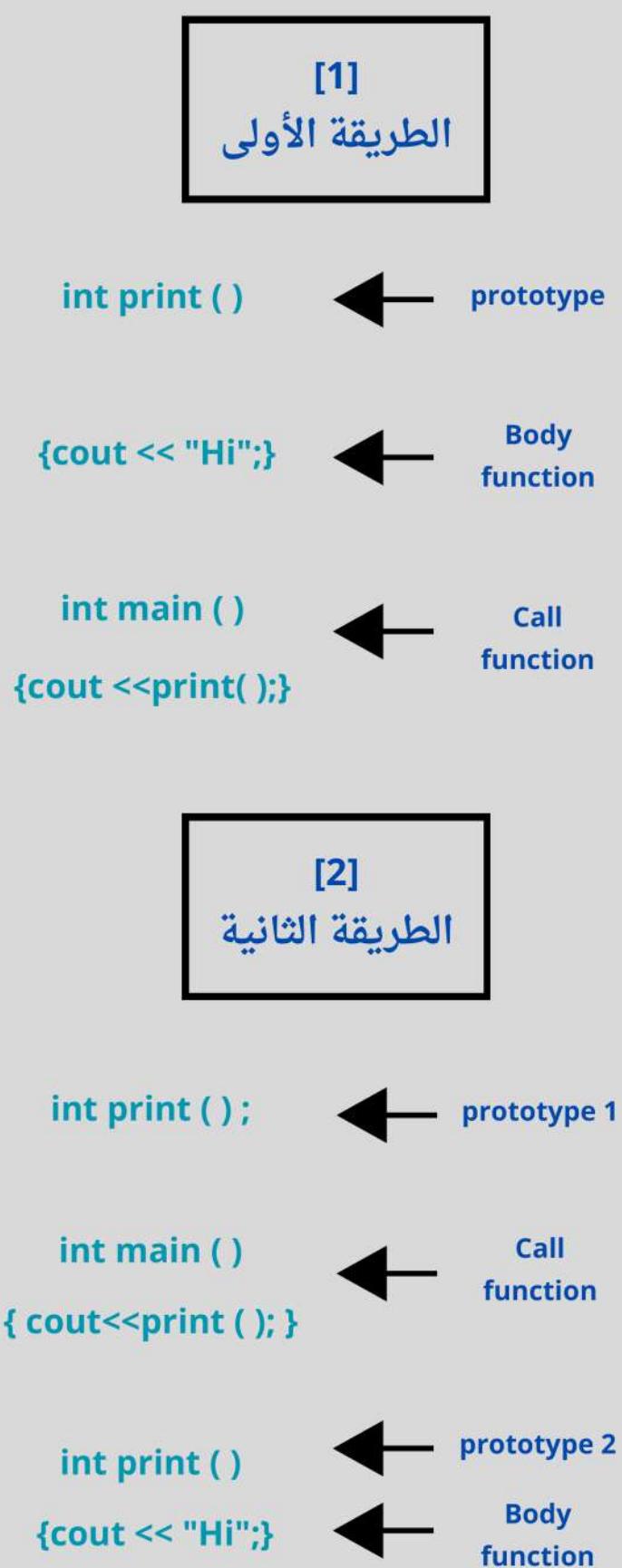
- نستخدمه لتعليمات برمجية تتوقع منها نتيجة مختلفة مثل جمع الأعداد

- نستخدم معها return

- أمثلة على طرق تعريف الدوال حسب أنواع الدوال



- أمثلة على طرق تعريف الدوال حسب أجزاء الدوال



الاستدعاء الذاتي (Recursion) ■

- هو عبارة عن دالة (function) تستدعي نفسها بنفسها لعدد معين من المرات

📌
 في حال كنت تريده أن تنفذ تعليمات
 الدالة 5 مرات مثلا ..
 وفي الدوال العاديّة يجب استدعاء
 الدالة 5 مرات داخل `int main`
 أما في الاستدعاء الذاتي نستدعيها
 مرة واحدة داخل `int main` وهي
 ستكمّل استدعاء نفسها 4 مرات

- أجزاء يتكون منها الاستدعاء الذاتي :

[1] تحديد شرط (Base Case)

[2] التعليمات البرمجية للدالة (Logic)

[3] زيادة أو نقصان (Sub problem)

[4] استدعاء الدالة (Call function)

- مثال على تعريف الاستدعاء الذاتي :

```
int numbers (int x)
{
  if ( x<1 )
    return 0;
```

(Base Case) → الشرط

Sub problem الـ

دائماً يحتوي على
 اسم الدالة + زيادة
 أو نقصان وبواسطته
 تستدعي الدالة
 نفسها

↑
 (Logic) → `cout << x << endl;`

(Sub problem) → `numbers (x-1) }`

`int main ()`

(Call function) → `{ cout << numbers(4) << endl;`

■ المتغير المحلي والعام (Local & Global variable)

• المتغير المحلي (Local variable) :

- # هو عندما يكون لكل دالة متغير خاص بها فهذا المتغير يسمى (متغير محلي)
- # يتم تعريف المتغير داخل كل دالة بشكل منفصل

• المتغير العام (Global variable) :

- # هو عندما يكون لكل الدوال متغير مشترك فهذا المتغير يسمى (متغير عام)
- # يتم تعريف المتغير أعلى الدوال وخارج الدالة الرئيسية (int main)

`int x = 5;` → متغير عام

`int fun1 ()`

`{ int x = 3;` → متغير محلي
`cout<< x << endl; }`

`int fun 2 ()`

`{ int x = 4;` → متغير محلي
`cout<< x << endl; }`

`int main ()`

`{ cout << fun1() << endl;`
`cout << fun2() << endl; }`

دائماً في حال وجود متغير محلي وعام يحملان نفس الاسم في الكود فإنه دائماً يصل للمتغير المحلي أولاً ..

مثل في الدالة fun1 سيطبع x على أنها القيمة 3 وليس 5

إذا كنت تريده أن يصل للمتغير العام ويطبع 5 بدلاً من 3 فإنه عليك استخدام علامة :: scope resolution بجانب اسم المتغير مثل أن تكتب في دالة fun1

`cout<<:: x << endl;`

■ دالة التحميل الزائد (Overloading function)

• هي مشكلة تمنع الكود من العمل، بسبب تسمية الدوال بنفس الاسم وتمرير نفس المتغيرات للدوال أو عدم تمريرها

• قد نضطر أحياناً لتوحيد أسماء الدوال، لذلك هناك طرق لتجنب مشكلة الـ (overloading function) وهي :

2] تمرير متغيرات تختلف بنوع البيانات

```
void print ( int x )
{ cout<< x << endl; }
```

```
void print ( double Y )
{ cout<< x << endl; }
```

```
int main ( )
{ cout << print( 10 ) << endl;
cout<< print( 10.5 ) << endl; }
```

1] تمرير متغيرات تختلف بعدها

```
void print ( int x, int y )
{ cout<< x << y; }
```

```
void print ( double Y )
{ cout<< x << endl; }
```

```
int main ( )
{ cout << print( 10 , 20 ) << endl;
cout<< print( 10 ) << endl; }
```

■ المتغير الافتراضي (Default argument)

• هي إعطاء المتغير قيمة احتياطية، في حال نسي المستخدم (User) تمرير قيمة للمتغير، فإنه سيضع تلقائياً القيمة الاحتياطية التي حزنتها سابقاً

- طريقة وضعها تكون عند تعريف المتغير في الدالة



```
int sum ( int x, int y = 10 )
{ return x + y }
```

`int main ()`

إما تمرير قيمة `x` فقط والناتج سيكون 15

أو تمرير قيمة `x` و `y` والناتج سيكون 9

- خصائص المتغير الافتراضي :

1] لا يجوز إسناد المتغيرات بدون المتغير الأخير، مثل :

`int sum (int x = 4 , int y = 5, int z)` ✗

`int sum (int x = 4 , int y = 5, int z = 6)` ✓

2] يمكن إسناد متغير افتراضي لأكثر من متغير، مثل :

`int sum (int x , int y = 5, int z = 6)` ✓

3] لا يجوز إسناد المتغيرات بشكل متقطع (أن لا تكون تلو بعضها البعض)، مثل :

`int sum (int x = 4 , int y , int z = 6)` ✗

`int sum (int x = 4 , int y = 5, int z = 6)` ✓



■ دوال المصفوفة (Array function)

- هي استخدام المصفوفات داخل الدوال (Function)
- نعرف المصفوفة وحجمها (size) في الدالة prototype مثل المتغيرات
- تعامل المصفوفة معاملة المتغيرات عند استخدامها في الدوال، حيث نمررها عند استدعاء الدالة

**void print (int arr [], int size) ←
تعريف المصفوفة وحجمها
{ cout<< arr[1] << endl; } Prototype داخلي الدالة**

```
int main ()
{ int arr[3] = { 10, 20, 30 } ← إعطاء عناصر للمصفوفة
cout<< print( arr , 3 ) << endl; } ← تمرير عناصر المصفوفة للدالة
```

■ تعبير لامدا (Lambda expression)

- يجب استدعاء مكتبة #include <functional> حتى يعمل تعبير لامدا
- تدعمها نسخ لغة C++ الحديثة فقط وهي النسخ الـ 11 وما فوقها
- يستعمل تعبير لامدا لكتابة الدوال (function) بصورة أبسط وأسهل

auto fun = [] () {}

**الصيغة العامة لكتابة
تعبير لامدا**

↑
↑
↑
↑

اسم الدالة
نضع المصفوفة
إن وجدت
لتمرير المتغيرات
إن وجدت
التعليمات
البرمجية
للدالة

• يقسم تعبير لامدا إلى 4 أنواع :

1] دالة بدون إرجاع وبدون تمرير ←

```
auto fun = [] () { cout<<" Hi " ;}
```

```
int main ()
```

{ fun (); } → (call function) استدعاء للدالة

2] دالة إرجاع وبدون تمرير ←

```
auto fun = [] () { return " Hi " }
```

```
int main ()
```

{ cout<<fun (); } → (call function) استدعاء للدالة

3] دالة بدون إرجاع مع تمرير ←

```
auto fun = [] ( int x ) { cout << x ; }
```

```
int main ()
```

{ fun (5); } → (call function) استدعاء للدالة

4] دالة إرجاع مع تمرير ←

```
auto fun = [] ( int x ) { return x ; }
```

```
int main ()
```

{ cout<<fun (5); } → (call function) استدعاء للدالة



تعليمات الذاكرة

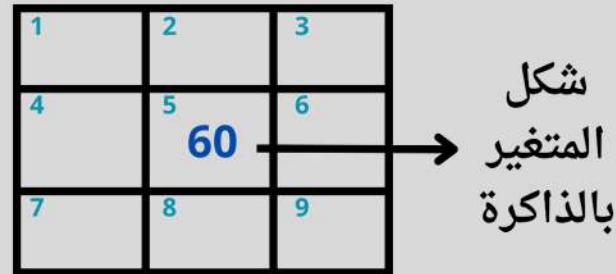
(RAM instructions)

المؤشرات (Pointers) ■

- هي أداة مهملاة في الذاكرة، يتم إنشاؤها عن طريق ربطها بالمتغير لتشير على مكان وجود المتغير (موقعه في الذاكرة RAM)، بالإضافة إلى أن المؤشر يأخذ خصائص المتغير كذلك ويصبح تابعاً له
- عندما نقوم بتعريف متغير فإن شكله بالذاكرة يشبه هذا :

تعريف المتغير
→ int x = 60;

اسم المتغير = x
عنوان المتغير = 5
قيمة المتغير = 60
نوع قيمة المتغير = int



- لماذا نستخدم المؤشرات ؟
- 1) أسرع في الوصول لخصائص المتغير، لأن المؤشر لا يأخذ حيز في الذاكرة مثل المتغير
- 2) تستخدم مع هيكل البيانات خصوصاً المصفوفات (Arrays) وهي ما يساعدنا بمعرفة العناصر index

تعريف المؤشر
وربطه بالمتغير

→ int x = 60;
int *ptr = &x

• كيفية تعريف المؤشرات :

طباعة قيمة
المتغير وهي 60

→ cout << x;
cout << *ptr ;

نفس	اسم *	=	اسم &
نوع	المؤشر		المتغير
بيانات			
المتغير			

طباعة عنوان
المتغير وهو 5

→ cout << &x;
cout << ptr;

الصيغة العامة
للتعريف

■ علاقة المؤشر بالمصفوفة (pointers with array)

- المصفوفات لا تحتاج إلى عملية ربط المؤشر مثل المتغيرات، فالمؤشرات بالمصفوفات تلقائية. وعند تعريف المصفوفة يكون المؤشر مربوط تلقائياً مع العنصر (index)
- طرق الوصول لعناصر المصفوفة حسب (index) :

`int x[3] = {10, 20, 30}`

تعريف المصفوفة

الطريقة الأولى : `cout << x[1];`

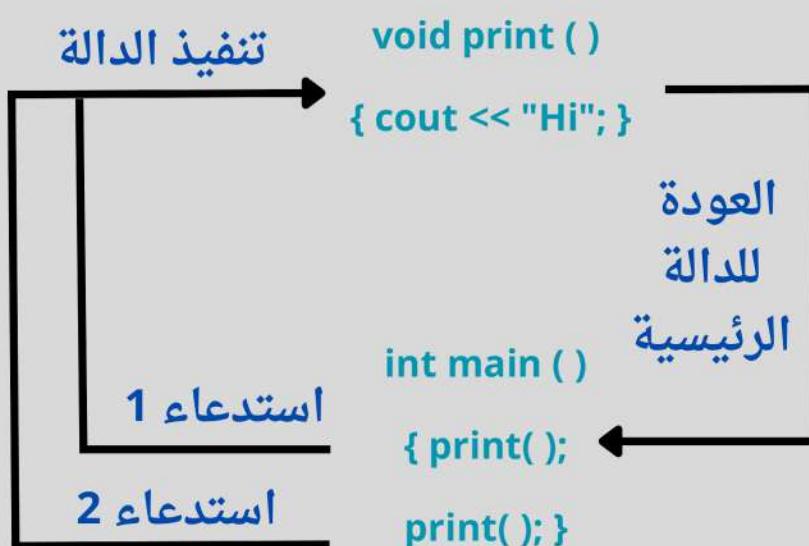
الطريقة الثانية : `cout << x+1;`

الطريقة الثالثة : `cout << *(x+1);`

مثال طباعة العنصر الثاني
وهو 20 بواسطة الطرق
الثلاثة

■ الدالة المضمنة (Inline)

- عندما تقوم بتنفيذ دالة (Function) بعد إنشائها، يتم الأمر هكذا :



تلقائياً مع كل عملية استدعاء يصعد للدالة ينفذها ثم يعود للدالة الرئيسية وهكذا، هذا يؤدي لجعل البرنامج أبطأ وأخذ وقت أكثر عند تنفيذه ، لذلك الحل في استخدام الدالة المضمنة (Inline)

- طريقة تعريف الدالة المضمنة تكون بهذا الشكل :

Inline void print ()

```
{ cout << "Hi"; }
```



اضافة كلمة **Inline**

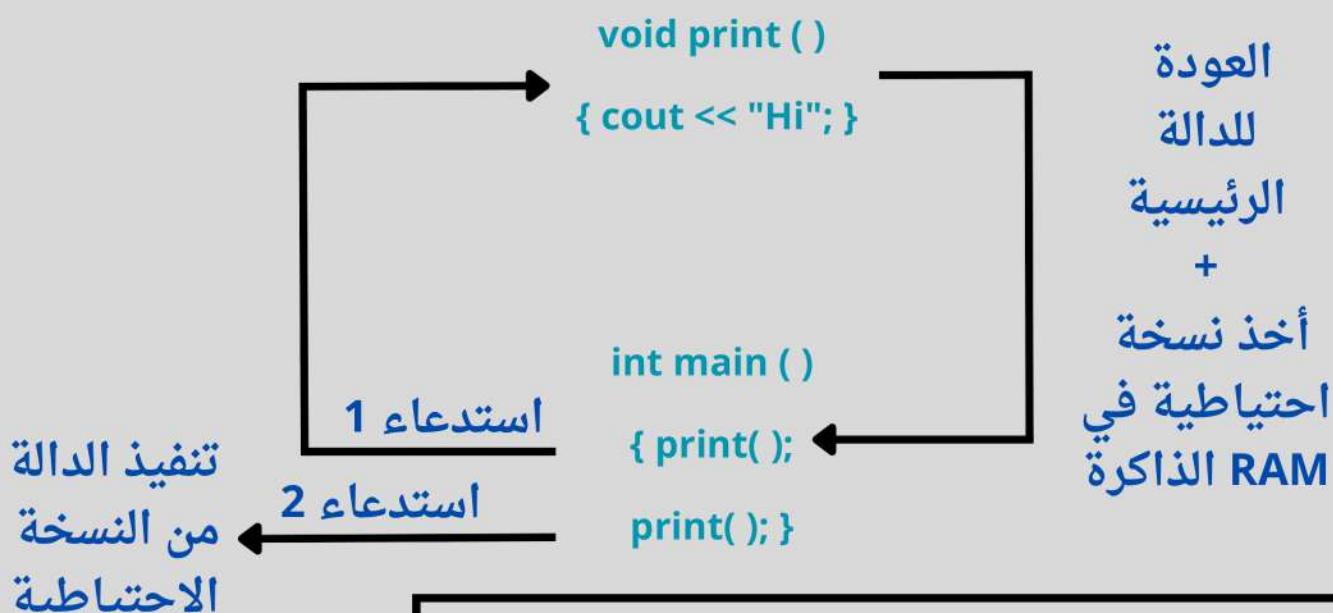
بجانب اسم الدالة

```
int main ()
```

```
{ print(); }
```

```
print(); }
```

- عندما تحول الدالة التي أنشأتها إلى دالة مضمونة، يتم الأمر بهذا الشكل :



عند استدعاء الدالة لأول مرة يصعد لينفذها ثم يأخذ نسخة احتياطية من تعليمات الدالة، ومع كل عملية استدعاء بعد المرة الأولى لا يصعد، بل ينفذها من الدالة الرئيسية عن طريق النسخة الاحتياطية التي أخذها للتعليمات البرمجية للدالة، هكذا سيكون البرنامج أسرع ووقته أقل عند التنفيذ

■ المتغير الثابت والمستعار والساكن (Constant & Aliasing & Static)

المتغير الساكن (Static)

هو المتغير الذي يحفظ آخر قيمةنفذتها تعليمات الدالة في الذاكرة لاستخدامها عند استدعاء الدالة من جديد، وذلك باستخدام `static`

```
void print ()  
{ static int x = 1;  
x++;  
cout<<x}  
int main ()  
{ print();  
print(); }
```

في الاستدعاء الأول سيطبع 2
ثم في الاستدعاء الثاني
سيطبع 3 أما (بدون استخدام
`static`) سيطبع في كل
الاستدعاءين 2

المتغير الثابت (Constant)

هو المتغير الذي لا يمكن تعديل قيمته، حيث تبقى قيمته ثابتة في الذاكرة ولا تتغير، ويمكن جعله متغيراً ثابتاً عن طريق وضع `const` بجانب تعريف المتغير

```
const int x = 50;
```

المتغير المستعار (Aliasing)

هو ربط المتغير الأول بعنوان المتغير الثاني، لتتأثر قيمهما بعضهما البعض، عن طريق علامة &

```
int x = 50;  
int &y = x;
```

المتغير `x` له عنوان منفصل عن المتغير `y` في الذاكرة، لكن قمنا بتعريف متغير مستعار وهو المتغير `y`، بذلك عندما تغير قيمة `y` تتغير قيمة المتغير `x` بنفس القيمة أو العكس

■ الاستدعاء بالقيمة والاستدعاء بالعنوان (Call by value & by reference)

- الاستدعاء بالقيمة (call by value) : هي تمرير متغيرات منفصلة عن بعضها
- الاستدعاء بالعنوان (call by reference) : هي تمرير متغيرات مرتبطة مع بعضها

• طريقة الاستدعاء بالقيمة

```
int sum ( int y )
{ y++;
cout << y << endl;

int main ()
{ int z = 1;
cout << sum( z );
```

يرسل المتغير z نسخة من
قيمتة للمتغير y فيصبح $y = 1$

في الاستدعاء بالقيمة لا علاقة للمتغير y بالمتغير z
حيث بعد تنفيذ الدالة سيكون $y = 1 / z = 2$

• طريقة الاستدعاء بالعنوان

```
int sum ( int &y ) ←
{ y++;
cout << y << endl;

int main ()
{ int z = 1;
cout << sum( z );
```

تتأثر كل من قيمة z و y
بعضهما البعض، لأن المتغير z
أصبح مربوطاً بعنوان المتغير y،
حيث أي تعديل على المتغير
y سيتأثر به المتغير z والعكس

في الاستدعاء بالعنوان يرتبط المتغير y بالمتغير z
حيث بعد تنفيذ الدالة سيكون $y = 2 / z = 2$



السلالس النصية (String)

- السلالس النصية عبارة عن مكتبة تحتوي على مجموعة من الأوامر البرمجية المبرمجة بواسطة مصفوفات الحروف (Array of character)، من أجل إنشاء النصوص بطريقة مختصرة

- طريقة استخدام السلالس النصية تكون بإدراج المكتبة أولاً وهي

`include <string>`

- يعامل `string` معاملة أنواع البيانات، حيث في اللغات البرمجية الحديثة أصبح نوع بيانات رئيسي

`string name = "Ahmed";` ← (Data type)

يتم كتابة النص داخل علامتي التنصيص بهذا الشكل

المكتبة (Library) : هي تعليمات برمجية قام ببرمجتها مبرمج ما داخل ملف منفصل، حيث يمكننا استدعاء ذلك الملف باستخدام أمر `<اسم المكتبة>` للإستفادة من وظائفها، وقد وجدت المكتبات لاختصار الوقت والجهد على المبرمجين

- هناك دوال جاهزة مضمونة موجودة داخل هذه المكتبة وأشهرها هي :

[1] دالة (`swap`) : التبديل بين قيم المتغيرات

```
string girl = "sarah";
string boy = " ahmed";
girl.swap(boy);
```

سيصبح المتغير `girl = "ahmed"` والمتغير `boy = "sarah"`

[2] دالة (assign) : إسناد قيمة متغير داخل متغير آخر

```
string girl = "sarah";
string boy = " ahmed";
girl.assign(boy);
```



سيصبح المتغير "girl" = "ahmed"
والمتغير "boy" كما هو

[3] دالة (length & size) : حساب طول النص (تقوم بعد الأحرف والفراغات)

```
string girl = "sarah";
string boy = " ahmed";
girl.length();
```

سيعطي 5
وهي عدد
أحرف
sarah

```
string girl = "sarah"
string boy = " ahmed"
girl.size();
```



[4] دالة (at) : البحث عن الحرف حسب موقعه (index)

```
string girl = "sarah";
cout<<girl.at(2);
```

سيطبع حرف r

[5] دالة (find) : البحث عن موقع الحرف (index) الخاص به

```
string girl = "sarah";
```

```
cout<<girl.find("s");
```

يبدأ البحث من اليسار لليمين وسيطبع 0

```
cout<<girl.rfind("s");
```

يبدأ البحث من اليمين لليسار وسيطبع 4

[6] دالة (replace) : استبدال حروف بالنص بحروف أخرى

```
string girl = "sarah ahmed";
cout<<girl.replace(6, 5, "adam");
```



سيطبع sarah adam

الرقم 6 : يمثل موقع الحرف الذي سنبدأ
من عنده الاستبدال

الرقم 5 : يمثل عدد الحروف التي نريد
ازالتها

الكلمة "adam" : تمثل الحروف التي نريد
استبدلها (وضعها بالنص)

7] دالة (insert) : إضافة حروف زائدة للنص

```
string girl = "sarah ahmed";
cout<<girl.insert(6, "adam");
```

سيطبع
sarahadamatmehmed

الرقم 6 : يمثل موقع الحرف الذي سنضيف الحروف الزائدة قبله الكلمة "adam" : تمثل الحروف الزائدة التي سنضيفها للنص

8] دالة (substr & erase) : حذف حروف من النص

```
string girl = "sarah ahmed";      الرقم 6      string girl = "sarah ahmed";
```

```
cout<<substr(6);      →      cout<<erase(6);
```

يمثل
موقع
الحرف
(index)

ahmed : (substr)
sarah a : (erase)
أما سيعطي في

الفرق بينهما عند وضع موقع الحرف (index)

أن substr سيحذف كل ما قبل موقع الحرف

أما erase سيحذف كل ما بعد موقع الحرف



معالجة الاستثناء (Exception Handling)

• هو الخطأ الذي يقع نتيجة غفلة المستخدم (user) عن إدخال المطلوب منه

• يتكون من 3 أجزاء :

1] الجزء الأول (try) : نكتب داخله الأكواد والتعليمات البرمجية والتي تطلب من المستخدم إدخال قيمة ما

2] الجزء الثاني (throw) : في هذا الجزء سيحدد نوع الخطأ الذي قام به المستخدم إن لم يدخل القيمة المطلوبة منه، وتقسم إلى خطأ رقم (throw 0) أو خطأ حرف (throw 'a')، وتكون throw داخل try

3] الجزء الثالث (catch) : نضع في الأكواد والتعليمات البرمجية التي تعالج الخطأ حسب نوعه من throw، ويتم استقبال الخطأ حسب نوعه هكذا فإذا أن يكون خطأ رقم catch(int x) أو خطأ حرف catch(char *x)

مثال لكود يطلب من المستخدم إدخال رقم موجب ومعالجة الأمر في حال إدخال رقم سالب



```
try { int x;
    cin>>x;
    if (x < 0)
        throw 0; }
```

catch(int x)

{ cout<<"Negative value"; }



• الدالة التي تحتوي على الأجزاء الثلاثة تسمى (Exception catcher)

• الدالة التي تحتوي على throw و try فقط بدون catch تسمى (Exception propagation)

• يمكنك وضع أكثر من catch و throw في الكود الواحد

• يمكنك وضع أي رقم أو حرف داخل throw، المهم أن يتم التمييز حسب نوعهم بأنهم حروف أم أرقام

الرقم العشوائي (Random number)

- يستخدم في حال أردت مخرجات عشوائية في البرنامج عند كل مرة يتم فيها تنفيذ البرنامج (Run)
- نستخدم فيه مكتبة #include<cstdlib>
- داخل هذه المكتبة يوجد عدة دوال، أشهرها :

الدالة (rands)

نضعها قبل الدالة rand من أجل تغيير ترتيب العناصر في كل مرة نقر على Run لتنفيذ البرنامج

1] `srand(1);`

تمثل الأرقام في الأقواس مرات تغيير الترتيب، فمثلاً إذا وضعنا 1 يعني سيغير الترتيب عند النقر على Run للمرة الأولى، أما إذا وضعنا 2 سيغير الترتيب عند النقر على Run للمرة الثانية، وهكذا ...

2] `srand(time(0));`

- يجب أولاً إدراج مكتبة `<ctime>` لاستخدام الدالة `time`
- تساعده في تغيير ترتيب العناصر كل مرة تضغط عليها على Run بدون تحديد رقم بين القوسين
- يمكنك وضع 0 أو NULL داخل قوسي الدالة `time`

الدالة (rand)

1] `cout << rand();`

طباعة أي رقم عشوائي (تكون من صفر إلى رقم max random وهو محدد باللغة)

2] `cout << rand()%20;`

طباعة رقم عشوائي من صفر لفترة نحددها (هنا من 0 إلى 19)

3] `cout << rand()%10+1;`

طباعة رقم عشوائي من واحد لفترة نحددها (هنا من 1 إلى 10)

4] `cout << rand()%(30-20+1)+20;`

طباعة رقم عشوائي من فترة نحددهما (هنا من 20 إلى 30)

مكتبة الرياضيات (Math library)

- هي مكتبة تضم دوال جاهزة، تستخدم لاجرام العمليات الحسابية في البرنامج (الكود)، فهي مختصة بالرياضيات بشكل عام، وأشهرها بالأمثلة :

1] `cout << max(2 , 3);` → إيجاد الرقم الأكبر بين عددين أو أكثر

2] `cout << min(2 , 3);` → إيجاد الرقم الأصغر بين عددين أو أكثر

3] `cout << abs(-2);` → إيجاد القيمة المطلقة لعدد

4] `cout << sqrt(9);` → إيجاد جذر العدد

5] `cout << pow(2 , 3);` → إيجاد القيمة الأسيّة (2 هو الأساس، و3 هو الأس)

6] `cout << exp2(4);` → إيجاد القيمة الأسيّة للأس 2 (هنا الأساس دائمًا 2، نتحكم فقط بالأس)

7] `cout << log10(3);` → حساب اللوغاريتم الطبيعي

8] `cout << exp(1);` → حساب ثابت أويلر (e)

9] `cout << remainder(7 , 8);`
`cout << fmod(7 , 8);` → إيجاد باقي قسمة عددين

10] `cout << copysign(-3 , 7);` → ينسخ إشارة عدد لعدد آخر (سيأخذ إشارة الرقم 7 ويعطيها للرقم 3 بذلك سيتحول من سالب إلى موجب)

11] `cout << ceil(6.1);` → التقريب لأكبر عدد صحيح

12] `cout << floor(6.1);` → التقريب لأصغر عدد صحيح

13] `cout << round(6.5);` → التقريب لأكبر عدد صحيح
إذا كان يمين الفاصلة = 5

14] `cout << rint(6.5);` → التقريب لأصغر عدد صحيح
إذا كان يمين الفاصلة = 5

15] `double pi = 3.14;`

`cout << sin(30*pi/180);`

`cout << cos(30*pi/180);`

`cout << tan(30*pi/180);`



إيجاد الدوال المثلثية
(جا، جتا، ظا)

16] `double pi = 3.14;`



إيجاد معكوس الدوال المثلثية
(-جا، -جتا، -ظا)

`cout << asin(30) *180/pi ;`

`cout << acos(30) *180/pi ;`

`cout << atan(30) *180/pi ;`



هنا الرقم 30
يمثل الزاوية
ويمكن تغييره

