مبادئ اساسية في البرمجة1

سنة أولى رياضيات

مدرسة المقرر

د. مها وهبي

مفهوم البرمجة

- جهاز الحاسب الآلـي آلــة تنفذ ما يأتيها من أوامــر بدقـة عالية.
- حيث تكـون هذه الأوامـر مكتوبــة فيمـا يسمى (برنـامج).
- وجميع البرامج تكون مكتوبة على هيئة سلسلة من الأوامرالتي ينفذها الحاسب الآلي لتخرج لنا بالشكل الذي نراه.
- هذه الأوامر تكتب بلغة معينة يفهمها جهاز الحاسب.

الفرق بين المبرمج ومستخدم البرنامج

المبرمج : هو من يقوم بكتابة البرامج ، حيث تمر العملية بعدة مراحل هي :

1) مرحلة فهم وتحليل المشكلة.

2) كتابة سلسلة من الأوامر لحل المشكلة.

3) اختبار البرنامج والتأكد من صحة عمله.

4) تحويل البرنامج إلى صيغة تنفيذية ، تمثل الشكل النهائي الذي يحتوي على الواجهة التي يراها المستخدم.

المستخدم : هو من يستخدم البرنامج، حيث تظهر له واجهة البرنامج ولن تظهر له الأوامر التي كتبها المبرمج.

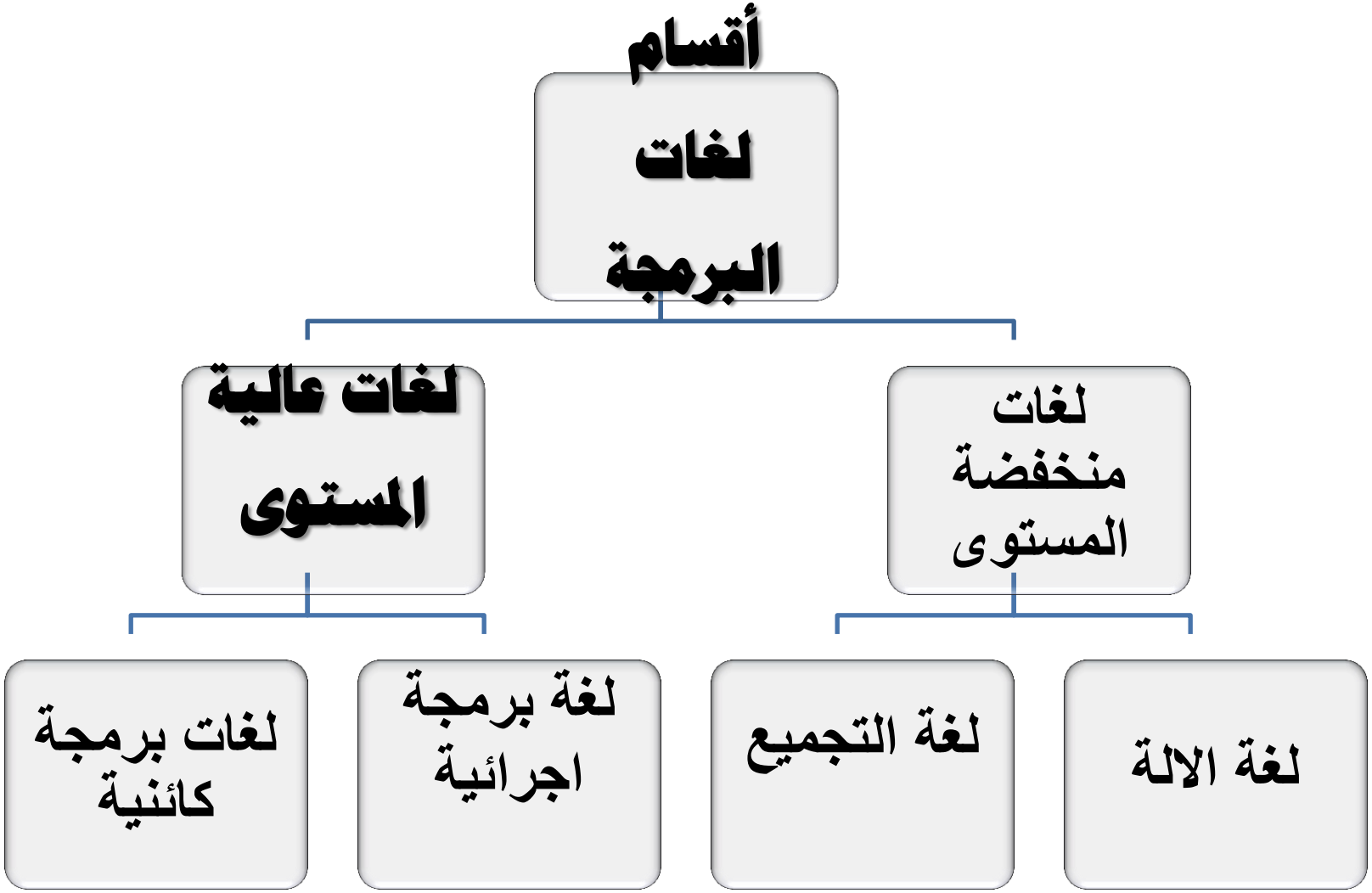
لغات البرمجة :

همزة الوصل بين الانسان وجهاز الحاسب

البرنامج:

هي مجموعة من التعليمات المتسلسلة والمترابطة لتنفيذ مهمة محددة و تكتب بأحدى لغات البرمجة.

تتفاوت لغات البرمجة من حيث نوعية التطبيقات و بيئة التشغيل وطريقة التفاعل بين المستخدم والتطبيق والوسط الذي يتم خلاله التفاعل


أقسام لغات البرمجة
لغات عالية المستوى
لغات منخفضة المستوى
لغات برمجة كائنية
لغة برمجة اجرائية
لغة التجميع
لغة الالة

اللغات المنخفضة المستوى L.L.L(Low Level Languages)

1. لغة الالة Machine languages

- تكتب الأوامر في لغة الآلة على شكل سلسلة من الأرقام الثنائية (الصفر والواحد) حتى يفهمها جهاز الحاسب الآلي
- وهي اللغة الوحيدة التي يفهمها الحاسب.
- تُحول جميع اللغات الى لغة الآلة حتى تتمكن معدات الحاسب الآلي من التفاهم معها.
- مميزاتها :
  - سرعة التنفيذ لأنها تخاطب وحدة المعالجة مباشرة
- عيوبها :

-غير مرنة (صعوبة كتابة وتصحيح برامجها).

-غير عمومية (برامجها تعتمد على نوع الآلة).

لغات المستوى العالي High_Level languages

- بظهور اللغات ذات المستوى العالي أصبحت عملية التخاطب والتعامل مع الحاسب أسهل نسبياً وذلك لأن لغة التعامل مع الحاسب أصبحت قريبة من لغة البشر.
- **بعض مميزات هذه اللغات:**

- قريبة من لغة الانسان.

- مرنة(سهولة في كتابة وتعديل وتصحيح البرامج).

-عمومية (عدم الارتباط بآلة معينة).

- توفير الوقت والجهد

عيوبها :

- بطء التنفيذ لاحتياجها لوسيط يقوم بتحويل البرنامج المصدر (Source Code) المكتوب باحدى هذه اللغات الى البرنامج الهدف (Object Code) المكتوب بلغة الآلة .

- **ومن الطبيعي لبرنامج مكتوب بلغة عالية المستوى أن يترجم إلى برنامج بلغة الآلة، ويطلق على البرنامج المكتوب باللغة عالية المستوى برنامج المصدر أو كود المصدر والبرنامج المترجم يطلق عليه في هذه الحالة المترجم (Compiler).**

اقسام اللغات عالية المستوى

1. لغات البرمجة الاجرائية(Procedural Programming Language)

- تستخدم المتغيرات وجمل الاسناد و جمل التحكم وجمل التكرار لكتابة البرنامج الاجرائى
- امثلة :

- لغة البيسك (Basic Language):طورت لمساعدة المبتدئين من كتابة برامجهم نظرا لبساطة تعليماتها، من اللغات المفسرة
- لغة فورتران(FORTRAN Language):تستخدم في المجال العلمي والهندسي، من اللغات المترجمة
- لغة كوبول (COBOL Language): متخصصة في الأعمال المالية والتجارية ، من اللغات المترجمة.
- لغة باسكال (PASCAL Language): تميزت بالسهولة والبساطة وقوة البرامج الفرعية، من اللغات الهيكلية المترجمة.
- لغة سي (C-Language):تمتعت بامكانية العمل على حواسيب مختلفة.

اقسام اللغات عالية المستوى

2. لغات البرمجة موجهة الاهداف
(Object Oriented Programming Language)

- تدعم مقومات مبنية على اساس كل كائن فى الحياة ينتمي الى طبقة أو صنف و كل طبقة تنحدرمن طبقة أعلى.
- من هذه المقومات: التغليف، اخفاء البيانات، الوراثة، اعادة الاستعمال .
- تدعم اسلوب البرمجة المرئية (تصميم الواجهات الرسومية)
- من امثلتها : visual basic , visual c++ , java builder

مقدمة عن لغة البرمجة C++

لغة البرمجة C++ تاريخها يرجع الى عام 1979 عندما كان يعمل Bjarne Stroustrup فى رسالته للدكتوراه.

- هي لغة برمجة عالية المستوى متعددة الاستخدام.
- وتعتبر لغة برمجة كائنية Object Oriented Programming.
- اعتبرها الكثيرون اللغة الأفضل لتصميم التطبيقات ذات الواجهة الكبيرة ، وذلك لسرعتها في التنفيذ والتي لا تختلف كثيرًا عن لغة C.
- توفر تعامل أكثر تعقيدًا مع البيانات.
- لغة C++ من لغات البرمجة العالية المستوى وفي نفس الوقت قريبة من لغة التجميع ذات المستوى المنخفض.
- كما أنها تعد لغة برمجة إجرائية ولغة غرضية التوجه.

مميزات لغة C++

- بالإضافة إلى المزايا الموجودة في لغة C تدعم لغة C++العديدة من المزايا الجديدة، نذكر منها الآتي :
- تدعم لغة C++ البرمجة الغرضية التوجه (OOP) Object Oriented Programming

وهي تمكن المبرمج من كتابة برامج تدعم النهج الجديد في البرمجة وهو البرمجة الموجهة نحو الأشياء (OOP) والتي فيها يتم تحليل وتصميم النظام بعد تحديد مكوناته، و لكل مكون يتم تحديد خصائصه، والعمليات المعرفة عليه(الصفوف والكائنات Classes and Objects، التحميل الزائد للعمليات، القوالب Templates، التعددية الشكلية polymorphism ، الوراثة)

- أستخدام الرمز // لتضمين الملاحظات بطول سطر واحد والتي يتم تجاهلها من قِبل المترجم عند القيام بعملية الترجمة.

مميزات لغة C++

- الإعلان عن المتغيرات free variable declaration.

أصبح الممكن في c++ الإعلان عن المتغيرات في أي موضع من البرنامج مما يتيح ربط المتغير بالوظيفة التي من اجلها تم الإعلان عنه، مما يزيد من سهولة متابعة وفهم البرنامج .

- الإعلان عن الثوابت constant:-

في c++ يتم استخدام الكلمة المحجوزة const للإعلان عن الثوابت كلآتي :-

- Data type const constant_ name = value
- Const data type constant_ name = value

ومن مزايا هذه الطريقة تساعد المترجم على فحص الأنواع type checkingوحجز ذاكرة تتناسب ونوع الثابت.

C++ components

- مكتبات السي ++:

1. iostream.h وهي مكتبة الادخال والاخراج.
2. stdio.h وهي ايضا مكتبة ادخال واخراج.
3. conio.h وهي مكتبة دوال اوامر الشاشة.
4. math.h وهي مكتبة الدوال الرياضية.
5. String.h مكتبة دوال معالجة النصوص.

البرنامج الاول بلغة C++

- لاحظ البرنامج كالتالي:

```
// my first program in C++

    #include <iostream.h>

int  main()
 {
 cout << "Hello World!";

    return 0;
}
```

تعليق لا ينفذ

تضمين أحد المكتبات القياسية

بداية الدالة الرئيسية للبرنامج ونوعها

الذي سوف يطبع على الشاشة

قوس نهاية الدالة

Header Files استيراد المكتبات #include <library_name.h>

Public Declaration منطقة التصاريح العامة

Subprograms الدوال الفرعية

<Data Type> main () الدالة الرئيسية

{ بداية الدالة الرئيسية

Private Declaration منطقة التصاريح الخاصة

Program Body Statements;

Statements; جمل برمجية

Statements;

return <value>; (option)القيمة الراجعة

}نهاية الدالة الرئيسية

Constant Escape Sequences

The following escape sequences can be used to print out special characters.

Escape Sequence	Description
\'	Single quote
\"	Double quote
\\	Backslash
\0	Null character
\a	Audible bell
\b	Backspace
\f	Formfeed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\xnnn	Hexadecimal number (nnn)

ASCII Chart

The following chart contains ASCII decimal, octal, hexadecimal and character codes for values from 0 to 127.

65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
48	60	30	0
49	61	31	1
50	62	32	2
51	63	33	3
52	64	34	4
53	65	35	5
54	66	36	6
55	67	37	7
56	70	38	8
57	71	39	9

أهم أنواع البيانات الأولية في C++ و أكثرها استخداماً هي:

`int` - `float` - `double` - `bool` - `char`

النوع `int`

يستخدم هذا النوع لتخزين عدد صحيح, أي عدد لا يحتوي على فاصلة عشرية.

مثال

```
int x = 10;
```

النوع `float`

يستخدم هذا النوع لتخزين عدد يمكن أن يحتوي على فاصلة عشرية.

يمكن لهذا العدد أن يحتوي على 7 أرقام بعد الفاصلة.

مثال

```
float x = 12.5;
```

النوع `double`

يستخدم هذا النوع لتخزين عدد يمكن أن يحتوي على فاصلة عشرية.

يمكن لهذا العدد أن يحتوي على 15 رقم بعد الفاصلة لهذا يعتبر أكثر دقة من النوع `float` في العمليات الحسابية الدقيقة.

مثال

```
double x = 12.5;
```

النوع `bool`

يستخدم هذا النوع لتخزين إما القيمة `true` و إما القيمة `false` .

مثال

```
bool x = true;
```

النوع `char`

يستخدم هذا النوع لتخزين حرف أجنبي أو لتخزين عدد صحيح قيمته تمثل حرف نسبةً لرقم الآسكي كود (**ASCII Code**) الخاص به.

فمثلاً `char x = 65;` نفسها تماماً إن كتبت `char x = 'A';` .

سبب تحول الرقم `65` إلى الحرف `'A'` في الذاكرة هو أنه في نظام **ASCII** الرقم `65` يمثل الحرف `'A'` .

مثال

```
char x = 'A';
```

تحديد خصائص أنواع البيانات في C++

في C++ يوجد مجموعة كلمات يقال لها **Data Type Modifiers** يمكنك استخدامها لتحديد خصائص القيم التي يمكن تخزينها, مثل هي يمكن أن تكون قيم أصغر من صفر أم لا, بالإضافة إلى تكبير حجم المساحة التي يتم تخصيصها لكل نوع في الذاكرة مما يجعلك قادر على تخزين قيم أكبر.

الأنواع `char` و `int` و `double` يمكن تحديد خصائص القيم التي يمكن تخزينها فيها إذا أضفنا معها إحدى الكلمات التالية:

- `signed` نضيفها لأحد الأنواع السابقة إذا أردنا تحديد أن قيمة المتغير لا يهم إن كانت أكبر أصغر أو تساوي صفر.
- `unsigned` نضيفها لأحد الأنواع السابقة إذا أردنا تحديد أن قيمة المتغير لا يمكن أن تكون أصغر من صفر.
- `short` نضيفها لأحد الأنواع السابقة إذا أردنا تحديد أن حجم الذاكرة التي سيتم تخصيصها للمتغير هو **2Byte**.
- `long` نضيفها لأحد الأنواع السابقة إذا أردنا تحديد أن حجم الذاكرة التي سيتم تخصيصها للمتغير هو **8Byte**.

النوع	الحجم في الذاكرة	القيمة التي يمكن تخزينها
`char`	1Byte	بين `-128` و `127`
`unsigned char`	1Byte	بين `0` و `255`
`short int`	2Bytes	بين `-32768` و `32767`
`unsigned short int`	2Bytes	بين `0` و `65535`
`int`	4Bytes	بين `-2147483648` و `2147483647`
`unsigned int`	4Bytes	بين `0` و `4294967295`
`long int`	4Bytes	بين `-2147483648` و `2147483647`
`unsigned long int`	4Bytes	بين `0` و `4294967295`
`long long int`	8Bytes	بين `-9223372036854775808` و `9223372036854775807`
`unsigned long long int`	8Bytes	بين `0` و `18446744073709551615`
`float`	4Bytes	بين `1.17549e-038` و `3.40282e+038`
`double`	8Bytes	بين `2.22507e-308` و `1.79769e+308`
`long double`	12Bytes	بين `3.3621e-4932` و `1.18973e+4932`

العوامل التي تستخدم في العمليات الحسابية (Arithmetic Operators)

إسم العامل	رمزه	مثال	شرح الكود
Assignment	`=`	`a = b`	أعطي `a` قيمة `b`
Addition	`+`	`a + b`	أضف قيمة `b` على قيمة `a`
Subtraction	`-`	`a - b`	إطرح قيمة `b` من قيمة `a`
Unary plus	`+`	`+a`	أضرب قيمة `a` بالعامل `+`
Unary minus	`-`	`-a`	أضرب قيمة `a` بالعامل `-`
Multiplication	`*`	`a * b`	أضرب قيمة `a` بقيمة `b`
Division	`/`	`a / b`	أقسم قيمة `a` على قيمة `b`
Modulo	`%`	`a % b`	للحصول على آخر رقم يبقى عندما نقسم قيمة `a` على قيمة `b`
Increment	`++`	`a++`	لإضافة 1 على قيمة `a` و تستخدم في الحلقات
Decrement	`--`	`a--`	لإنقاص 1 من قيمة `a` و تستخدم في الحلقات

العوامل التي تستخدم في المقارنات (Comparison Operators)

إسم العامل	رمزه	مثال	شرح الكود
Equal to	==	(a == b)	هل قيمة a تساوي قيمة b ؟ إذا كان الجواب نعم فإنها ترجع true
Not equal to	!=	(a != b)	هل قيمة a لا تساوي قيمة b ؟ إذا كان الجواب نعم فإنها ترجع true
Greater than	>	(a > b)	هل قيمة a أكبر من قيمة b ؟ إذا كان الجواب نعم فإنها ترجع true
Less than	<	(a < b)	هل قيمة a أصغر من قيمة b ؟ إذا كان الجواب نعم فإنها ترجع true
Greater than or Equal to	>=	(a >= b)	هل قيمة a أكبر أو تساوي قيمة b ؟ إذا كان الجواب نعم فإنها ترجع true
Less than or Equal to	<=	(a <= b)	هل قيمة a أصغر أو تساوي قيمة b ؟ إذا كان الجواب نعم فإنها ترجع true

العوامل التي تستخدم في وضع شروط منطقية (Logical Operators)

إسم العامل	رمزه	مثال	شرح الكود
AND	`&&`	`(a && b)`	هل قيمة `a` و `b` تساويان `true` ؟ هنا يجب أن يتم تحقيق الشرطين ليرجع `true`
OR	`||`	`(a || b)`	هل قيمة `a` أو `b` أو كلاهما تساويان `true` ؟ هنا يكفي أن يتم تحقيق شرط واحد من الشرطين ليرجع `true`
NOT	`!`	`!a`	هل قيمة `a` لا تساوي `true` ؟ إذا كان الجواب نعم فإنها ترجع `true`

العوامل التي تستخدم لإعطاء قيم للمتغيرات (Assignment Operators)

إسم العامل	رمزه	مثال	شرح الكود
Basic Assignment	`=`	`a = b`	ضع قيمة `b` في `a` .
Add AND Assignment	`+=`	`a += b`	أضف قيمة `a` على قيمة `b` و خزن الناتج في `a`
Susbtract AND Assignment	`-=`	`a -= b`	أنقص قيمة `a` من قيمة `b` و خزن الناتج في `a`
Multiply AND Assignment	`*=`	`a *= b`	أضرب قيمة `a` بقيمة `b` و خزن الناتج في `a`
Divide AND Assignment	`/=`	`a /= b`	أقسم قيمة `a` على قيمة `b` و خزن الناتج في `a`
Modulo AND Assignment	`%=`	`a %= b`	أقسم قيمة `a` على قيمة `b` و خزن آخر رقم يبقى من عملية القسمة في `a`

مفهوم التعامل مع الأعداد في C++

هناك الكثير من الدوال الجاهزة في C++ التي يمكن استخدامها لإجراء عمليات حسابية. بعض هذه الدوال يمكن استخدامها بشكل مباشر لأنها تعتبر معروفة بالنسبة لمترجم اللغة بشكل إفتراضي و بعض الدوال بحاجة لأن تضّمنها بنفسك أولاً حتى تتمكن من استخدامها.

بالإجمال, حتى تستطيع استخدام الدوال المخصصة لإجراء العمليات الحسابية, يجب تضمين الملف `cmath` الذي يحتويها.

لذلك ستجد أننا سنضيف السطر التالي في أي مثال نستخدم فيه إحدى الدوال التي سنتعلمها في هذا الدرس.

```
#include <cmath>
```

هذا السطر يعني أننا نريد إضافة محتوى الملف `cmath` في البرنامج مما يجعلنا قادرين على استخدام الدوال الموجودة فيه.

الدالة `abs()`

تعريفها

ترجع **القيمة المطلقة** (Absoulte Value) للعدد الذي نمرره لها مكان الباراميتر `x` .

باراميترات

مكان الباراميتر `x` نمرر لها رقم نوعه `double` أو `float` أو `long double` .

قيمة الإرجاع

ترجع القيمة المطلقة للعدد الذي نمرره لها مكان الباراميتر `x` و ترجعها من نفس نوعه.

أي إذا مررت لها قيمة نوعها `double` فأنت بذلك تستدعي الدالة الأولى التي ترجع قيمة مطلقة نوعها `double` .

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

int main()
{
    double a = -130;
    float b = -0.15;
    long double c = -5;

    cout << "abs(a) = " << abs(a) << endl;
    cout << "abs(b) = " << abs(b) << endl;
    cout << "abs(c) = " << abs(c) << endl;

    return 0;
}
```

```
abs(a) = 130
abs(b) = 0.15
abs(c) = 5
```

قيمة الإرجاع

ترجع القيمة المطلقة للعدد الذي نمرره لها مكان البارامتر `x` و ترجعها من نفس نوعه.

أي إذا مررت لها قيمة نوعها `double` فأنت بذلك تستدعي الدالة الأولى التي ترجع قيمة مطلقة نوعها `double` .

fabs (x)

مثال

Main.cpp

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    double a = -130;
    float b = -0.15;
    long double c = -5;

    cout << "fabs(a) = " << fabs(a) << endl;
    cout << "fabs(b) = " << fabs(b) << endl;
    cout << "fabs(c) = " << fabs(c) << endl;

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
fabs(a) = 130
fabs(b) = 0.15
fabs(c) = 5
```

قيمة الإرجاع

ترجع ناتج عملية مضاعفة قيمة الباراميتر `x` بقيمة الباراميتر `y` و ترجعها من نفس نوع القيم التي تم تمريرها لها.

أي إذا مررت لها قيم نوعها `double` فأنت بذلك تستدعي الدالة الأولى التي ترجع ناتج نوعه `double` .

```
pow(x, y)
```

مثال

Main.cpp

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    float a = 5;
    float b = 2;

    cout << a << " ^ " << b << " = " << pow(a, b);

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
5 ^ 2 = 25
```

قيمة الإرجاع

ترجع قيمة **الجذر التربيعي** للعدد الذي تمرره لها مكان البارامتر `x` و ترجعها من نفس نوعه.

أي إذا مررت لها قيمة نوعها `double` فأنت بذلك تستدعي الدالة الأولى التي ترجع قيمة نوعها `double` .

```
sqrt(x)
```

مثال

Main.cpp

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    float x = 25;

    cout << "The square root of " << x << " is: " << sqrt(x);

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
The square root of 25 is: 5
```

قيمة الإرجاع

ترجع قيمة `e` مضاعفة بقيمة الباراميتر `x` و ترجعها من نفس نوعه.

أي إذا مررت لها قيمة نوعها `double` فأنت بذلك تستدعي الدالة الأولى التي ترجع قيمة نوعها `double` .

exp(x)

مثال

Main.cpp

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    double x = 0.5;

    cout << "e ^ " << x << " = " << exp(x);

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
e ^ 0.5 = 1.64872
```

قيمة الإرجاع

ترجع قيمة اللوغاريتم الطبيعي للبارامتر `x` و ترجعها من نفس نوعه.

أي إذا مررت لها قيمة نوعها `double` فأنت بذلك تستدعي الدالة الأولى التي ترجع قيمة نوعها `double` .

`log(x)`

مثال

Main.cpp

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    double x = 5.5;

    cout << "log(" << x << ") = " << log(x);

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
log(5.5) = 1.70475
```

باراميترات

يمكن تمرير عددين من أي نوع لها عند إستدعائها مكان الباراميترين `x` و `y` و سيقوم مترجم لغة C++ بإستدعاء الدالة المناسبة لنوع القيم التي قمت بتمريرها.

قيمة الإرجاع

ترجع الرقم الذي يتبقى من قسمة قيمة الباراميتر `x` على قيمة الباراميتر `y` و ترجعه على حسب نوع القيم التي تم تمريرها.

```
fmod(x, y)
```

مثال

Main.cpp

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    int a = 10;
    int b = 7;

    cout << a << " % " << b << " = " << fmod(a, b);

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
10 % 7 = 3
```

C++

جمل الشرط

جمل الشرط بشكل عام

الشكل العام لوضع الشروط هو التالي.

```
if ( condition )
{
    // إذا كان الشرط صحيحاً نفذ هذا الكود
}

else if ( condition )
{
    // إذا كان الشرط صحيحاً نفذ هذا الكود
}

else
{
    // نفذ هذا الكود في حال لم يتم التعرف على الكود في أي شرط
}
```

المثال الأول

- إذا كانت قيمة المتغير `S` أكبر من 5 سيتم طباعة الجملة: `S is bigger than 5` .

Main.cpp

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

int main()
{
    int S = 0;

    if( S > 5 )
    {
        cout << "S is bigger than 5";
    }

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
```

- هنا سأل نفسه التالي: هل قيمة المتغير `S` أكبر من 5؟

فكان جواب الشرط كلا (`false`) , لذلك لم ينفذ أمر الطباعة الموجود في جملة الشرط.

المثال الثاني

- إذا كانت قيمة المتغير `S` أكبر من 5 سيتم طباعة الجملة: `S is bigger than 5` .

Main.cpp

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

int main()
{
    int S = 30;

    if( S > 5 )
    {
        cout << "S is bigger than 5";
    }

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
S is bigger than 5
```

- هنا سأل نفسه التالي: هل قيمة المتغير `S` أكبر من 5؟

فكان جواب الشرط نعم (`true`)، لذلك نفذ أمر الطباعة الموجود في جملة الشرط.

جملة الشرط `else`

`else` في اللغة العربية تعني " أي شيء آخر ". و هي تستخدم فقط في حال كنا نريد تنفيذ كود معين في حال كانت نتيجة جميع الشروط التي قبلها تساوي `false` .

يجب وضعها دائماً في الأخير لأنها تستخدم في حال لم يتم تنفيذ أي جملة شرطية قبلها.

إذاً, إذا نفذ البرنامج الجملة `if` أو `else if` فإنه سيتجاهل الجملة `else` .

و إذا لم يتفذ أي جملة من الجمل `if` و `else if` فإنه سينفذ الجملة `else` .

المثال الأول

- إذا كانت قيمة المتغير `S` تساوي 5 سيتم طباعة الجملة: `S is equal 5` .
- إذا كانت قيمة المتغير `S` لا تساوي 5 سيتم طباعة الجملة: `S is not equal 5` .

Main.cpp

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

int main()
{
    int S = 5;

    if( S == 5 )
    {
        cout << "S is equal 5";
    }

    else
    {
        cout << "S is not equal 5";
    }

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
S is equal 5
```

- هنا سأل نفسه التالي: هل قيمة المتغير `S` تساوي 5؟

فكان جواب الشرط نعم (`true`) , لذلك نفذ أمر الطباعة الموجود في الجملة `if` .

المثال الثاني

- إذا كانت قيمة المتغير `S` تساوي 5 سيتم طباعة الجملة: `S is equal 5` .
- إذا كانت قيمة المتغير `S` لا تساوي 5 سيتم طباعة الجملة: `S is not equal 5` .

Main.cpp

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

int main()
{
    int S = 20;

    if( S == 5 )
    {
        cout << "S is equal 5";
    }

    else
    {
        cout << "S is not equal 5";
    }

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
S is not equal 5
```

- هنا سأل نفسه التالي: هل قيمة المتغير `S` تساوي 5؟

فكان جواب الشرط كلا (`false`) , لذلك نفذ أمر الطباعة الموجود في الجملة `else` .

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

int main()
{
    int number = 3;

    if( number == 1 )
    {
        cout << "one";
    }

    else if( number == 2 )
    {
        cout << "two";
    }

    else if( number == 3 )
    {
        cout << "three";
    }

    else if( number >= 4 )
    {
        cout << "four or greater";
    }

    else
    {
        cout << "negative number";
    }

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
three
```

- هنا سأل نفسه التالي: هل قيمة المتغير `number` تساوي 1؟

فكان جواب الشرط كلا (`false`)، فانتقل إلى الشرط الذي يليه.

- ثم سأل نفسه التالي: هل قيمة المتغير `number` تساوي 2؟

فكان جواب الشرط كلا (`false`)، فانتقل إلى الشرط الذي يليه.

- ثم سأل نفسه التالي: هل قيمة المتغير `number` تساوي 3؟

فكان جواب الشرط هذه المرة نعم (`true`)، فقام بتنفيذ أمر الطباعة الموجود في جملة الشرط الثالثة، ثم تجاوز جميع جمل الشرط التي أتت بعده.

تعريف الجملة `switch`

`switch` نستخدمها إذا كنا نريد إختبار قيمة متغير معين مع لائحة من الإحتمالات نقوم نحن بوضعها, و إذا تساوت هذه القيمة مع أي إحتمال وضعناه ستتنفذ الأوامر التي وضعناها في هذا الإحتمال فقط.

كل إحتمال نضعه يسمى **case**.

أنواع المتغيرات التي يمكن إختبار قيمتها باستخدام هذه الجملة هي:

`int` - `byte` - `short` - `char` - `enum`.

طريقة تعريفها

يمكننا تعريفها بعدة أشكال, الشكل الأساسي هو التالي:

```
switch(expression) {

    case value:
        // Statements
        break;

    case value:
        // Statements
        break;

    default:
        // Statements
        break;

}
```

- `switch` تعني إختبر قيمة المتغير الموضوع بين قوسين.

- `expression` هنا يقصد بها المتغير الذي نريد إختبار قيمته.

نوع المتغير الذي يسمح لنا بإختباره: `int` - `byte` - `short` - `char` - `String` - `enum` .

- `case` تعني حالة, `value` تعني قيمة, و `Statements` تعني أوامر.

و يقصد من هذا كله, أنه في حال كانت قيمة الـ `expression` تساوي هذه القيمة سيقوم بتنفيذ الأوامر الموضوعة بعد النقطتين `:` .

الآن بعد تنفيذ جميع الأوامر الموضوعة بعد النقطتين, يجب وضع `break` لكي يخرج من الجملة `switch` مباشرةً بدل أن ينتقل لـ `case` التالية الموجودة في الجملة `switch` .

نستطيع وضع العدد الذي نريده من الـ `case` بداخل الجملة `switch` .

إنتبه: الـ `expression` و الـ `value` يجب أن يكونا من نفس النوع.

- `default` تعني إفتراضياً و هي نفس فكرة الجملة `else` , و يمكننا أن لا نضعها أيضاً.

هذه الجملة تنفذ فقط في حال لم تنفذ أي `case` موجودة في الجملة `switch` و لذلك نضعها بالآخر.

> لا حاجة لوضع `break` للحالة الأخيرة لأن البرنامج سيخرج من الجملة `switch` في جميع الأحوال.

مفهوم الحلقات

نستخدم الحلقات (Loops) بهدف تكرار نفس الكود عدة مرات.

إذاً أي كود نريده أن يتنفذ عدة مرات, نقوم بكتابته داخل حلقة فتقوم هي بإعادة تنفيذ الكود قدر ما شئنا ضمن شروط معينة نقوم نحن بتحديدها.

أنواع الحلقات في C++

إسم الحلقة	
For Loop	تستخدم الحلقة `for` في حال كان عدد المرات التي سيعاد فيها تنفيذ الكود معروفاً.
While Loop	يفضل إستخدام الحلقة `while` في حال كان عدد المرات التي سيعاد فيها تنفيذ الكود غير معروف.
Do While Loop	يفضل إستخدام الحلقة `do while` في حال كان عدد المرات التي سيعاد فيها تنفيذ الكود غير معروف و بنفس الوقت يجب أن يتنفذ مرة واحدة على الأقل.

تعريف الحلقة `for`

نستخدم الحلقة `for` إذا كنا نريد تنفيذ الكود عدة مرات محددة, فمثلاً إذا كنا نريد تنفيذ كود معين 10 مرات, نضعه بداخل حلقة تعيد نفسها 10 دورات.

طريقة إستخدامها

```
for( initialisation; condition; increment أو decrement )
{
    // statements
}
```

- `statements` : هي الخطوة الثالثة, و تعني تنفيذ جميع الأوامر الموجودة في الحلقة و هي تتنفذ في كل دورة.

بعد أن تتنفذ جميع الأوامر سيصعد إلى الخطوة الأخيرة التي تحدث في نهاية كل دورة و هي إما زيادة قيمة العداد أو إنقاصها.

- `initialisation` : هي أول خطوة تتنفذ في الحلقة و هي تتنفذ مرة واحدة فقط على عكس جميع العناصر الموجودة في الحلقة.

في هذه الخطوة نقوم بتعريف متغير (يسمى عداد) و نضع بعده `;` .

- `condition` : هي ثاني خطوة تتنفذ في الحلقة و هي تتنفذ في كل دورة.

في هذه الخطوة نقوم بوضع شرط يحدد متى تتوقف الحلقة, في كل دورة يتم التأكد أولاً إذا تحقق هذا الشرط أم لا, و نضع بعده `;` .

هنا طالما أن نتيجة الشرط تساوي `true` سيعيد تكرار الكود.

- `increment` أو `decrement` : هي الخطوة الرابعة و الأخيرة, و هي تتنفذ في كل دورة.

هنا نحدد كيف تزداد أو تنقص قيمة العداد, و لا نضع بعده `;` .

مثال

Main.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    // هنا قمنا بإنشاء حلقة for تتألف من 10 دورات. في كل دورة تطبع قيمة العداد المستخدم فيها
    for( int i=1; i<=10; i++ )
    {
        cout << i << endl;
    }

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
1
2
3
4
5
6
7
8
9
10
```

هنا كأننا نقول: "طالما أن الشرط لا يزال يتحقق إستمر في تكرار الكود".

طريقة إستخدامها

```
initialisation;

while( condition )
{
    // statements

    increment أو decrement;
}
```

- `initialisation` : هي أول خطوة تنفذ في الحلقة و هي تنفذ مرة واحدة فقط على عكس جميع العناصر الموجودة في الحلقة.

في هذه الخطوة نقوم بتعريف متغير (يسمى عداد).

- `condition` : هي ثاني خطوة تنفذ في الحلقة و هي تنفذ في كل دورة.

في هذه الخطوة نقوم بوضع شرط يحدد متى تتوقف الحلقة, في كل دورة يتم التأكد أولاً إذا تحقق هذا الشرط أم لا.

هنا طالما أن نتيجة الشرط تساوي `true` سيعيد تكرار الكود.

- `statements` : هي الخطوة الثالثة, و تعني تنفيذ جميع الأوامر الموجودة في الحلقة و هي تنفذ في كل دورة.

- `decrement` أو `increment` : هي الخطوة الرابعة و الأخيرة, و هي تنفذ في كل دورة.

هنا نحدد كيف تزداد أو تنقص قيمة العداد.

مثال حول الحلقة `while`

في المثال التالي قمنا بتعريف حلقة تطبع جميع الأرقام من 1 إلى 10.

مثال

Main.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    // هنا قمنا بتعريف المتغير الذي استخدمناه كعداد في الحلقة
    int i=1;

    // هنا أنشأنا حلقة while تظل تنفذ الأوامر الموضوعة فيها طالما أن قيمة العدد لا تزال أصغر أو تساوي 10
    while( i<=10 )
    {
        // في كل دورة سيتم طباعة قيمة العداد ثم إضافة 1 عليها
        cout << i << endl;
        i++;
    }

    return 0;
}
```

```
1
2
3
4
5
6
7
8
9
10
```

تعريف الحلقة `do while`

نستخدم الحلقة `while` إذا كنا نريد تنفيذ الكود عدة مرات، و لكننا لا نعرف كم مرة بالتحديد.

هذه الحلقة تتوقف عن تكرار نفسها إذا لم يعد الشرط الموضوع فيها يتحقق.

هنا كأننا نقول: "نفّذ الكود و إن كان الشرط لا يزال يتحقق فقم بتنفيذه من جديد".

إذاً الفرق الوحيد بينها و بين الحلقة `while` أنها تنفذ مرة واحدة على الأقل لأنها تتأكد من الشرط بعد تنفيذ الأوامر و ليس قبلهم.

```
initialisation;

do{
    // statements
    increment أو decrement;
}
While( condition );
```

طريقة إستخدامها

```
initialisation;

do{
    // statements
    increment أو decrement;
}
While( condition );
```

- `initialisation` : هي أول خطوة تنفذ في الحلقة و هي تنفذ مرة واحدة فقط على عكس جميع العناصر الموجودة في الحلقة.

في هذه الخطوة نقوم بتعريف متغير (يسمى عداد).

- `statements` : هي الخطوة الثانية, و تعني تنفيذ جميع الأوامر الموجودة في الحلقة و هي تنفذ في كل دورة.

- `decrement` أو `increment` : هي الخطوة الثالثة, و هي تنفذ في كل دورة.

هنا نحدد كيف تزداد أو تنقص قيمة العداد.

- `condition` : هي الخطوة الرابعة و الأخيرة و هي تنفذ في كل دورة.

في هذه الخطوة نقوم بوضع شرط يحدد متى تتوقف الحلقة, في نهاية كل دورة يتم التأكد إذا تحقق الشرط أم لا.

هنا طالما أن نتيجة الشرط تساوي `true` سيعيد تكرار الكود.

تذكر فقط أن جميع هذه الخطوات تتكرر في كل دورة ما عدا أول خطوة, و السبب أننا لا نحتاج إلى تعريف عداد جديد في كل دورة, بل نستعمل العداد القديم و الذي من خلاله نعرف في أي دورة أصبحنا.

مثال حول الحلقة `do while`

في المثال التالي قمنا بتعريف حلقة تطبع جميع الأرقام من 1 إلى 10.

مثال

Main.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    // هنا قمنا بتعريف المتغير الذي استخدمناه كعداد في الحلقة
    int i=1;

    // هنا أنشأنا حلقة while تظل تنفذ الأوامر الموضوعة فيها طالما أن قيمة العدد لا تزال أصغر أو تساوي 10
    do
    {
        // في كل دورة سيتم طباعة قيمة العداد ثم إضافة 1 عليها
        cout << i << endl;
        i++;
    }
    while( i<=10 );

    return 0;
}
```

C++

جملة التحكم `break`

تعريف الجملة `break`

الجملة `break` تستخدم في الحلقات و في الجملة `switch` .

بمجرد ان تنفذ الجملة `break` فإنها توقف الـ scope بأكمله و تخرج منه و تمسحه من الذاكرة ثم تنتقل للكود الذي يليه في البرنامج.

طريقة تعريفها

تتألف هذه الجملة من أمر واحد و يكتب على سطر منفرد.

```
break;
```

في المثال التالي قمنا بتعريف حلقة كانت ستطبع جميع الأرقام من 1 إلى 10 لولا أننا إستخدمنا الجملة `break` لجعل الحلقة تتوقف عندما تصبح قيمة العداد `i` تساوي 6.

مثال

Main.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    // هنا قمنا بإنشاء حلقة for تتألف من 10 دورات. في كل دورة تطبع قيمة العداد المستخدم فيها
    for( int i=1; i<=10; i++ )
    {
        // في كل دورة سيتم فحص قيمة العداد و بمجرد أن تصبح تساوي 6 سيتم إيقاف الحلقة نهائياً
        if( i == 6 ) {
            break;
        }

        cout << i << endl;
    }

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
1
2
3
4
5
```

C++

جملة التحكم `continue`

تعريف الجملة `continue`

نستخدم الجملة `continue` لتجاوز تنفيذ كود معين في الحلقة, إذاً نستخدمها لتجاوز جزء من كود الـ scope.

و نستخدمها تحديداً لإيقاف الدورة الحالية و الإنتقال إلى الدورة التالية في الحلقة, لا تقلق ستفهم المقصود من المثال.

طريقة تعريفها

تتألف هذه الجملة من أمر واحد و يكتب على سطر منفرد.

```
continue;
```

أمثلة حول جملة التحكم `continue`

في المثال التالي قمنا بتعريف حلقة تطبع جميع الأرقام من 1 إلى 10 ما عدا الرقم 3.

إستخدمنا الجملة `continue` لجعل الحلقة تتجاوز الدورة الثالثة في الحلقة. أي لن يتم تنفيذ أمر الطباعة عندما تصبح قيمة العداد `i` تساوي 3.

المثال الأول

Main.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    // هنا قمنا بإنشاء حلقة for تتألف من 10 دورات. في كل دورة تطبع قيمة العداد المستخدم فيها
    for (int i=1; i<=10; i++)
    {
        // في كل دورة سيتم فحص قيمة العداد، عندما تصبح تساوي 3 سيتم الإنتقال إلى الدورة التالية في الحلقة بدون تنفيذ أمر الطباعة الموضوع بعدها
        if (i == 3) {
            continue;
        }

        cout << i << endl;
    }

    return 0;
}
```

```
1
2
4
5
6
7
8
9
10
```

في المثال التالي قمنا بتعريف حلقة تطبع جميع الأرقام المفردة من 1 إلى 10.

إستخدمنا الجملة `continue` لجعل الحلقة تتجاوز كل دورة تكون فيها قيمة العداد `i` عبارة عن عدد زوجي.

المثال الثاني

Main.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    // هنا قمنا بإنشاء حلقة for تتألف من 10 دورات. في كل دورة تطبع قيمة العداد المستخدم فيها
    for (int i=1; i<=10; i++)
    {
        // في كل دورة سيتم فحص قيمة العداد. في حال كانت مزدوجة سيتم الإنتقال إلى الدورة التالية في الحلقة بدون تنفيذ أمر الطباعة الموضوع بعدها
        if (i%2 == 0) {
            continue;
        }

        cout << i << endl;
    }

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
1
3
5
7
9
```

هنا قمنا بتجربة الجملة `continue` مع الحلقة `for` فقط، لكن المبدأ هو نفسه تماماً مع الحلقات `while` و `do while`.

مفهوم المصفوفات في C++

المصفوفة (**Array**) عبارة عن متغير واحد يتألف من عدة عناصر (**Elements**) من نفس النوع.

و كل عنصر في المصفوفة يمكن تخزين قيمة واحدة فيه.

عناصر المصفوفة تتميز عن بعضها من خلال رقم محدد يعطى لكل عنصر يسمى **index**.

أول عنصر في المصفوفة دائماً يكون رقمه `0` .

الآن, عليك معرفة أن عدد عناصر المصفوفة ثابت, أي بمجرد أن قمت بتحديده لا يمكنك تغييره من جديد, مع الإشارة إلى أنك تستطيع تغيير قيم هذه العناصر متى شئت.

تعريف مصفوفة في C++

هناك ثلاث طرق يمكنك اتباعها لتعريف مصفوفة (Declare Array) جديدة سنتعرف عليها تباعاً.

```
// الأسلوب التالي يستخدم لتعريف مصفوفة مع تحديد عدد عناصرها
datatype arrayName[size];

// الأسلوب التالي يستخدم لتعريف مصفوفة مع تحديد قيمها الأولية
datatype arrayName[] = {value1, value2, ..};

// الأسلوب التالي يستخدم لتعريف مصفوفة مع تحديد عدد عناصرها و قيمة بعض عناصرها
datatype arrayName[size] = {value1, value2, ..};
```

- `datatype` : هو نوع القيم التي يمكن تخزينها في عناصر المصفوفة.
- `size` : هو عدد عناصر المصفوفة.
- `arrayName` : هو إسم المصفوفة.
- `[]` : هذا الرمز يمثل من كم بعد تتألف المصفوفة.

أمثلة حول طريقة تعريف مصفوفة أحادية (One Dimensional Array).

أمثلة

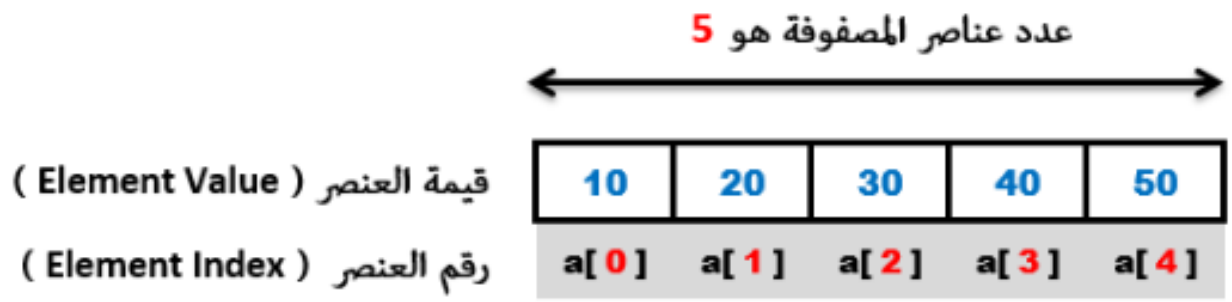
```
// هنا قمنا بتعريف مصفوفة ذات بعد واحد إسمها arr, نوعها int و تتألف من 5 عناصر
int arr[5];

// هنا قمنا بتعريف مصفوفة ذات بعد واحد إسمها arr, نوعها int و وضعنا فيها 6 عناصر, و هذا يعني أن عدد عناصرها أصبح 6 لأننا لم نحدد عدد عناصرها
int arr[] = {1, 2, 3, 4, 5, 6};

// هنا قمنا بتعريف مصفوفة ذات بعد واحد إسمها arr, نوعها int و تتألف من 5 عناصر, و قمنا بوضع قيم أولية في أول 3 عناصر فيها
int arr[5] = {1, 2, 3};
```

الوصول لعناصر المصفوفة في C++

لنفترض الآن أننا قمنا بتعريف مصفوفة نوعها `int` , إسمها `a` , و تتألف من 5 عناصر.

```
int a[] = { 10, 20, 30, 40, 50 };
```

يمكنك تصور شكل المصفوفة `a` في الذاكرة كالتالي.



بما أن المصفوفة تتألف من 5 عناصر, تم إعطاء العناصر أرقام indexes بالترتيب من 0 إلى 4.

إذاً هنا أصبح عدد عناصر المصفوفة يساوي 5 و هو ثابت لا يمكن تغييره لاحقاً في الكود.

و للوصول لقيمة أي عنصر نستخدم index العنصر الذي تم إعطاؤه له.

مثال

```cpp
#include <iostream>

using namespace std;

int main()
{
    // هنا قمنا بتعريف مصفوفة تتألف من 5 عناصر
    int arr[] = {10, 20, 30, 40, 50};

    // هنا قمنا بتغيير قيمة العنصر الأول و العنصر الأخير في المصفوفة
    arr[0] = 1;
    arr[4] = 5;

    // هنا قمنا بعرض قيم جميع عناصر المصفوفة
    cout << "arr[0] = " << arr[0] << endl;
    cout << "arr[1] = " << arr[1] << endl;
    cout << "arr[2] = " << arr[2] << endl;
    cout << "arr[3] = " << arr[3] << endl;
    cout << "arr[4] = " << arr[4] << endl;

    return 0;
}
```

- سنحصل على النتيجة التالية عند التشغيل.

```
arr[0] = 1
arr[1] = 20
arr[2] = 30
arr[3] = 40
arr[4] = 5
```

```cpp
#include<iostream>
#include<cmath>
#include<algorithm>
const int size=50;
main(){
//برنامج لحساب مجموع عناصر مصفوفة وطباعته
 int arr[size],n,s=0;
  cout<<" number of element   ";
  cin>>n;
  cout<<" element\n";
  for (int i=0;i<n;i++)
    cin>>arr[i];      //اعطاء قيم لعناصر المصفوفة

   for (int i=0;i<n;i++)  //حساب مجموع عناصر المصفوفة
    s+=arr[i];
   cout<<"sum= "<<s<<endl;   //طباعة المجموع
```

```cpp
 //البحث عن عنصر ضمن مصفوفةوطباعة دليل العنصر
    int arr[10],i,x,k;
    cin>>x;
for(i=0;i<5;i++)
cin>>arr[i];
 for(i=0;i<5;i++)
    if (x==arr[i]) {
    cout<<x<<endl;
     k=i;
    }
    cout<<k<<endl;
```

```cpp
// max and min and sum
int arr[size],n,max,min,s;
cout<<" number of element   ";
cin>>n;
cout<<" element\n";
for (int i=0;i<n;i++)
  cin>>arr[i];      //اعطاء قيم لعناصر المصفوفة

 max=min=s=arr[0];
 for (int i=1;i<n;i++)
 {
  if (arr[i]>max)
  max=arr[i];
  if (arr[i]<min)
  min=arr[i];
  s+=arr[i];
 }
 cout<<"max=  "<<max<<"   min=  "<<min<<"  s= "<<(float)s/n<<endl;

   //odd numbers
int arr[size],n,i;
  cout<<" number of element   ";
  cin>>n;
  cout<<" element\n";
  for ( i=0;i<n;i++)
    cin>>arr[i];
   cout<<"odd\n";
  for (int i=0;i<n;i++)
    if (arr[i]%2!=0)
     cout<<arr[i]<<endl;
```

```
  //تقسيم المصفوفة لمصفوفتين  مصفوفة اعداد فردية ومصفوفة اعداد زوجية
   int arr[size],arr1[size],arr2[size],n,i,k=0,l=0;
cout<<" number of element   ";
cin>>n;
cout<<" element\n";
for ( i=0;i<n;i++)
  cin>>arr[i];
cout<<"-----\n";
 for (int i=0;i<n;i++)
  if (arr[i]%2!=0)
  {
    arr1[k]=arr[i];
   k=k+1;
  }
  else
  {
      arr2[l]=arr[i];
      l++;
  }
  cout<<" odd array\n" ;
 for (int i=0;i<k;i++)    //طباعة مصفوفة الاعداد الفردية
  cout<<arr1[i]<<"  ";
  cout<<endl;

  cout<<" even array\n";
 for (i=0;i<l;i++)    //طباعة مصفوفة الاعداد الزوجية
  cout<<arr2[i]<<"  ";
  cout<<endl;

```

```cpp
    //مجموع الأعداد ذات الترتيب الفردي
int arr[size],i,n,s=0;

cout<<" number of element   ";
cin>>n;
cout<<" element arr\n";

   //اعطاء قيم لعناصر المصفوفة
for (int i=0;i<n;i++)
    cin>>arr[i];

for (int i=0;i<n;i++)
    if (i%2!=0)
     s+=arr[i];
 cout<<"sum "<<s<<endl;

//مجموع مصفوفتين
  int arr[size],arr1[size],arr2[size],n,s=0;
  cout<<" number of element   ";
  cin>>n;
  cout<<" element arr1\n";

   //اعطاء قيم لعناصر المصفوفة
  for (int i=0;i<n;i++)
    cin>>arr1[i];

   cout<<" element arr2\n";
  for (int i=0;i<n;i++)
    cin>>arr2[i];

   for (int i=0;i<n;i++)    //حساب مجموع المصفوفتين
    arr[i]=arr1[i]+arr2[i];
     cout<<" sum arr= \n";

    for (int i=0;i<n;i++)    //طباعة المصفوفة الجديدة
    cout<<arr[i]<<"  ";
    cout<<endl;
}
```