

The image shows the front cover of a book. The top portion is a solid gold band. Below this, the background is a dark navy blue. A large, elegant gold 'C' is positioned on the left, followed by two gold '+' signs to its right. Below the 'C++' text, the text 'Ch2' is written in a smaller, gold serif font. In the bottom right corner, there is a small, dark grey circular logo containing the white lowercase letters 'ai'.

C++

Ch2

Header

```
#include <iostream>
```

```
int main()
```

```
{
```

```
    std::cout << "Hello World!\n";
```

```
}
```

#include 1

. هذا السطر بنسميه header file أو ملف الترويسة.

. #include معناها بنطلب من الكمبيوتر يجب مكتبة جاهزة (فيها أوامر جاهزة).

. `iostream` مكتبة مسؤولة عن الإدخال والإخراج، يعني عشان نقدر نطبع أو نقرأ من الكمبيوتر.

. بدون هذا السطر ما راح يتعرف الكمبيوتر على `std::cout` أو `std::cin`.

int main()2

- أي برنامج بلغة C++ لازم يكون فيه دالة رئيسية اسمها `main`.
- `main` هي النقطة اللي بيبدا منها تنفيذ الكود.
- بيرجع `int` لأنه لازم يرجع قيمة عددية (عادة 0) بعد انتهاء البرنامج.

- ```
std::cout << "Hello World!\n";
```
- `std::cout` هو أمر للطباعة على الشاشة.
  - `<<` بنستخدمه عشان نرسل النص لـ `cout`.
  - `"Hello World!"` هو النص اللي بدنا نطبعه.
  - `\n` معناها ينزل سطر جديد بعد الطباعة.
  - `std::` معناها بنستخدم `cout` الموجود في الـ `standard namespace`.

## Cout (Printing Statement)

- في لغة C++ بنستخدم جملة الطباعة `cout` عشان نطبع نصوص أو قيم على الشاشة `cout`. اختصار لـ **character output** يعني إخراج الأحرف. بنكتبه بهذا الشكل:
- ```
std::cout << "Hello World!\n";
```

او

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

بنستخدم الرمز << عشان نرسل النص أو القيمة لـ cout ليتم طبعها.

Endline and \n

في نهاية جملة الطباعة لو بدنا نطبع سطر جديد بنستخدم:

std::endl

بنضيفه بعد جملة الطباعة عشان ينزل المؤشر لسطر جديد :

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

\n (Backslash n)

بنستخدمه عشان ينزل سطر جديد بدون ما يعمل flush:

```
std::cout << "Hello World!\n";
```

الفرق بينهم:

• std::endl ينزل سطر جديد ويعمل flush.

• \n ينزل سطر جديد بدون flush.

std:: Namespace

كل ما نكتب cout, endl، أو أي شيء من مكتبة C++ لازم نحدد

إنه تابع للـ namespace اسمه std. بنكتبه هيك:

std::cout

std::endl

الـ std::معناها بنستخدم الكائن من داخل الـ std namespace،
والـ namespace هو زي مجلد أو مساحة داخلها الأوامر الجاهزة
للغة.

Namespace

الـ namespace في C++ هو طريقة لتنظيم الكود ومنع تعارض
الأسماء.
لما نكتب std::cout بنحدد للمترجم إنه يستخدم الـ cout اللي داخل
الـ std namespace.

لو ما بدنا نكرر std:: كل مرة بنكتب:

```
using namespace std;
```

وبصير نقدر نكتب:

```
cout << "Hello World!\n" ;
```

بدون الحاجة نكتب std:: في كل مرة.

Comments

الكومنت (Comment) هو شرح للكود وما بيتنفذ أثناء تشغيل
البرنامج.

Single-line comment

بنستخدمه لسطر واحد:

// Tips for Getting Started:

Multi-line comment

بنستخدمه لأكثر من سطر:

/* Tips for Getting Started :

1. Use the Solution Explorer window to
add/manage files*/

بنستخدم الكومنت عشان نوضح الكود أو نشرح الغرض منه.

Escape Sequences in

Escape Sequences	Meaning
\'	Single Quote
\"	Double Quote
\\	Backslash
\0	Null
\a	Bell
\b	Backspace
\f	form Feed
\n	Newline
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab

\n

بنتزل سطر زي endl

\t

بتزيح المؤشر بمقدار Tab

\b

بترجع المؤشر بمقدار حرف واحد

\r

برجع المؤشر لبداية السطر الحالي

\\

Backslash \ بطبع

\"

Double Quote " بطبع

\'

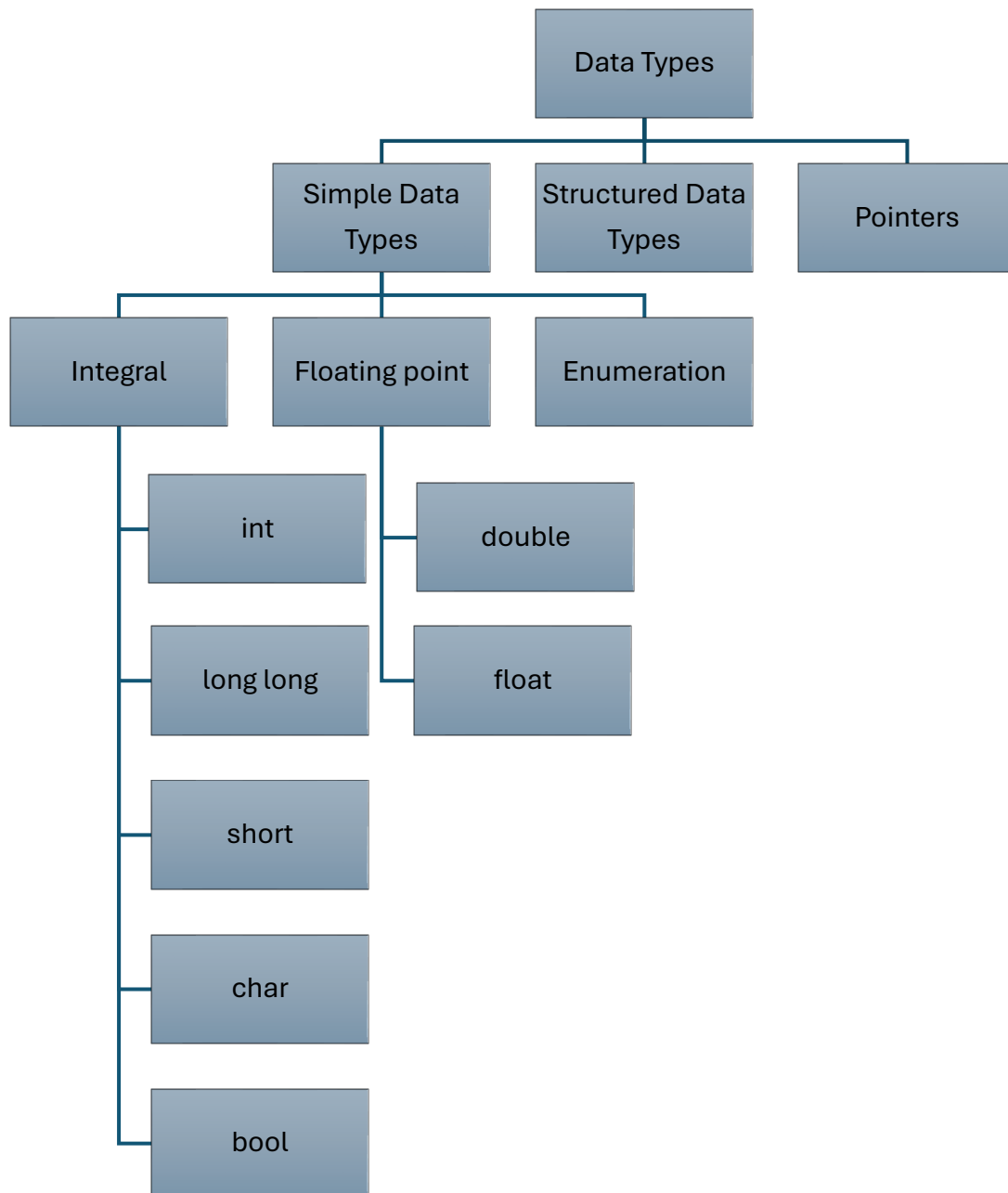
Single Quote ' بطبع

Variable

Variable is like a container that holds a value (data) in the computer's memory. You give it a name so you can use and change its value while your program runs.

المتغير (Variable) هو زِي الحاوية أو الصندوق بنعطيه اسم، ومنحط فيه قيمة (بيانات) عشان نقدر نستخدمها ونغيرها أثناء تشغيل البرنامج.

Data types in C++



Different types of variables (defined with different keywords)

int

- Whole Number: 712

char

- Single Character: 'A'

float

- Floating Point: 14.5

double

- Floating Point: 14.5

bool

- True or False

string

- Text: "Mohammed"

void

- Empty

The basic data types commonly used to define integers in C++ include:

أنواع البيانات و المتغيرات المستخدمة عادةً لتحديد الأعداد الصحيحة في C++ ما يلي:

- int
- long
- short

The **int** and **long** data types occupy **4 bytes** of memory, and the **short** data types occupy **2 bytes**.

تشغل أنواع البيانات int و long 4 بايتات من الذاكرة، وتشغل أنواع البيانات short 2 بايت.

The basic data types commonly used to define floating-point numbers or decimal numbers include:

تتضمن أنواع البيانات الأساسية المستخدمة عادةً لتحديد الأرقام العائمة أو الأرقام العشرية ما يلي:

- double
- long double
- float

The **double** and **long double** data types occupy **8 bytes** of memory, while the **float** data types occupy **4 bytes**.

تشغل أنواع البيانات double و long double 8 بايت من الذاكرة،
في حين تشغل أنواع البيانات float 4 بايت.

The following data type is used to identify an alphabetic character or any characters:

- char

يتم استخدام نوع البيانات char لتحديد حرف أبجدي أو أي
أحرف:

Each character occupies **1 byte** of memory.

نظام ASCII CODE وتخزين الأحرف داخل المتغيرات من نوع char :

لما نكتب أي حرف أو رمز على لوحة المفاتيح (keyboard) ،
بيكون لكل حرف أو رمز قيمة عددية خاصة فيه بنظام اسمه

. ASCII CODE

الكومبايلر لما يقرأ المتغير من نوع char (حرف)، هو فعليًا
بيتعامل مع الرقم اللي يمثل هذا الحرف في جدول ASCII ، مش مع
الحرف نفسه.

على سبيل المثال:

. الحرف الكبير (Uppercase letter) زي A إله قيمة عددية
65 =

. الحرف الكبير B = 66

. وهكذا بتزيد القيم تدريجيًا حتى توصل Z = 90

أما الأحرف الصغيرة: (Lowercase letters)

. الحرف a = 97

. الحرف b = 98

. وهكذا بتزيد تدريجيًا حتى توصل z = 122

وبالنسبة للمسافة: (space)

. الـ space إله قيمة عددية = 32

ملاحظة مهمة:

ما في داعي نحفظ كل القيم العددية لكل حرف، يكفي نعرف:

$$A = 65 .$$

$$a = 97 .$$

$$\text{space} = 32 .$$

العلاقة بين الأحرف الكبيرة والصغيرة:

في نظام **ASCII**، الفرق بين الحرف الكبير والحرف الصغير لنفس الحرف هو دائماً **32**.
يعني:

. لو عندنا حرف كبير **A** (قيمته 65) بدنا نحوله لصغير: بنضيف

$$\rightarrow a_{97} = 32 + 65 \rightarrow 97$$

. لو عندنا حرف صغير **a** (قيمته 97) بدنا نحوله لكبير: بنطرح

$$\rightarrow A_{65} = 32 - 97 \rightarrow 65$$

يعني الفرق العددي بين الحرف الكبير والصغير هو نفس قيمة الـ **space (32)**.

كيف يتعامل الكومبايلر مع الأحرف:

. لما نخزن حرف داخل متغير من نوع **char**، الكومبايلر

بيخزن القيمة العددية المقابلة للحرف من جدول **ASCII**.

- ولما نطبع الحرف باستخدام مثلاً `cout`، الكومبايلر يحول الرقم المخزن ويرجعه لحرف ويعرضه.

خلاصة:

- كل حرف أو رمز إله رقم بنظام **ASCII**
- الكومبايلر يتعامل مع الرقم مش الحرف مباشرة
- الفرق بين الحرف الكبير والصغير هو 32
- الـ `space` كمان إله رقم `= 32`

نوع البيانات **bool** وكيفية التعامل معه

نوع البيانات **bool** هو نوع خاص في لغة البرمجة `C++` (وغيرها) يستخدم لتخزين القيم المنطقية. (Boolean values)
يعني بيخزن بس احتماليين:

- **true** (صحيح)
- **false** (خطأ)

كيف يتمثل القيم داخل المتغير من نوع **bool** ؟

• الكومبايلر فعلياً بيخزن **true** كـ **1**

• وبيخزن **false** كـ **0**

يعني لو كتبت:

`bool x = true;`

راح يخزن في الذاكرة الرقم 1.
ولو كتبت:

`bool y = false;`

راح يخزن في الذاكرة الرقم 0.

لو خزننا رقم في متغير **bool** ؟

. لو خزنت أي رقم غير الصفر (زي 5 أو -7 أو 100)،
الكومبايلر بيعتبره **true** وبخزن داخلياً 1

. لو خزنت الرقم 0، بيعتبره **false** وبخزن 0

يعني فعلياً:

. 1 بتمثل كل الأرقام غير الصفر داخل متغير **bool**

. 0 بتمثل قيمة الخطأ أو عدم الصحة

مثال سريع:

`bool a = 7; // 1` راح يخزن 1

`bool b = 0; // 0` راح يخزن 0


`bool c = -3; // 1` راح يخزن 1

خلاصة:

- متغير bool يخزن 1 أو 0 فقط.
- **true** → 1
- **false** → 0
- أي رقم غير 0 يتحول تلقائيًا إلى 1.

You can review the table below for the data types used in C++.

Type	Definition	Size	Limits
int	Integer	4 bytes	-2147483648 to 2147483647
short	Short Integer	2 bytes	-32768 to 32767
long	Long Integer	8 bytes	-2147483648 to 2147483647
float	Floating Decimal Number	4 bytes	1.17549e-038 to 3.40282e+038
double	Double Decimal Number	8 bytes	2.22507e-308 to 1.79769e+308
long double	Long Decimal Number	10 bytes	2.22507e-308 to 1.79769e+308
char	Character	1 byte	-128 to 127
unsigned int	Unsigned Integer	4 bytes	0 to 4294967295
unsigned short	Unsigned Short Integer	2 bytes	0 to 65535
unsigned long	Unsigned Long Integer	8 bytes	0 to 4294967295
unsigned char	Unsigned Character	1 byte	0 to 255
bool	True or False	1 byte	True = 1 and False = 0

unsigned int  Positive integer

How Do You Declare Variable Data Types in C++?

<datatype> <name of variable>;

Here is how you declare a variable data type in C++ code:

```
int age;  
float price;  
char letter;
```

It is possible to change the content of a variable by assigning a specific value anywhere in the program. Often, however, the data type of the

variable is determined from the outset, while it is desirable to have a value as well.

<datatype> <name of variable> = <value>;

Here is how that would look in code:

```
int age = 26;  
float price = 32.95;  
char letter = "f";
```

If we are going to use more than one variable in our program, we can define these variables by writing them side by side, provided that they are of the same type.

تسمية المتغيرات في C++

Identifier : اسم المتغير الذي نستخدمه للوصول للقيمة.

لازم يبدأ بحرف أو _ ، بعده ممكن أرقام أو حروف.

ما بصير الاسم يبدأ برقم.

Case sensitive يعني الحروف الكبيرة والصغيرة بتفرق يعني

مثلا (age ≠ Age)

- ما بصير نستخدم كلمة محجوزة كاسم متغير.

أمثلة كلمات محجوزة:

int, return, while, if

هاي كلمات محجوزة للغة وما بنقدر نسمي متغيرات فيها

الأفضل تختار اسم واضح ومعبر زي score ، total ، name

Here is how you declare multiple variables in C++:

```
int num1, num2;
```

```
int num1=13, num2=14;
```

Lines where the variable data type is declared must end with.“;”

Code:

```
#include  
  
using namespace std;  
  
int main()  
{  
    int smallest=10;  
    int largest=100;  
  
    cout << "Smallest Number: " << smallest << "\n";  
    cout << "Largest Number: " << largest << "\n";  
    return 0;  
}
```

كيف نعرّف متغيرات (Variables) في C++؟

في لغة C++، لما بدنا نعرّف متغير (Variable)، لازم نكتب نوع البيانات (Data Type) تبعه أول إشي، بعدين اسم المتغير. شكل التعريف بيكون بالشكل التالي:

```
<datatype> <variable name>;
```

يعني مثلاً لو بدنا نعرّف متغير من نوع **int** (عدد صحيح) :

```
int age;
```

كيف نعطي المتغير قيمة مباشرة عند تعريفه؟

بنقدر نعطي المتغير قيمة أولية (Initial value) من البداية أثناء تعريفه:

```
<datatype> <variable name> = <value>;
```

مثال:

```
int age = 20;
```

كيف نعرّف أكثر من متغير من نفس النوع؟

لو بدنا نعرّف أكثر من متغير بنفس نوع البيانات، بنقدر نكتبهم ورا بعض، مفصولين بفواصل (,) :

```
int num1, num2;
```

وإذا بدنا نعطيهم قيم مباشرة:

```
int num1 = 13, num2 = 14;
```

ملاحظة مهمة:

. كل سطر بنعرّف فيه متغير لازم ينتهي بـ (semi-colon) ; .

مثال :

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

```
int smallest = 10;

int largest = 100;

cout << "Smallest Number: " << smallest << "\n";
cout << "Largest Number: " << largest << "\n";

return 0;
}
```

في هذا المثال:

- عرّفنا متغيرين من نوع **int** وهما **smallest** و **largest**، وعطيناهم قيم أولية 10 و 100.
- طبعنا قيمهم باستخدام **cout**.

خلاصة:

✓ التعريف عن متغير في C++ يكون بالشكل:

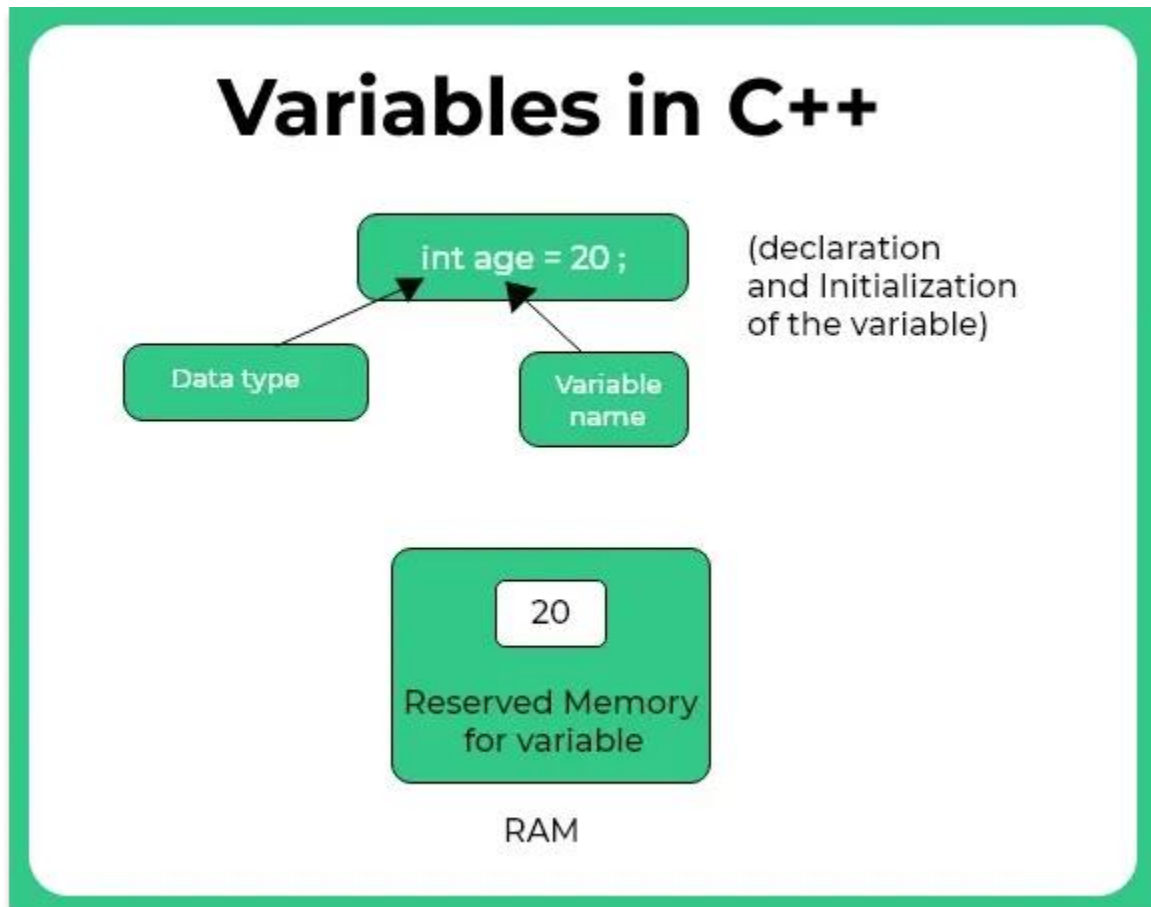
<datatype> <variable name>;

أو لو بدك تعطيه قيمة مباشرة:

<datatype> <variable name> = <value>;

✓ لو المتغيرات نفس النوع، بنقدر نكتبهم ورا بعض بفاصلة.

✓ لازم كل سطر ينتهي بـ ;



المتغير (Variable) هو مكان محجوز في ذاكرة الجهاز (RAM) بنخزن فيه قيمة معينة خلال وقت تنفيذ البرنامج.

لما نعرف متغير، الكمبيوتر بيحجز مكان خاص لإله في **RAM** (الذاكرة المؤقتة) وبيخزن فيه القيمة اللي عطيناها إياها (زي 20 في المثال).

✓ كل متغير إليه:

- نوع بيانات (Data type)
- اسم (Name)
- قيمة (Value)
- مكان مخصص في الذاكرة (Memory)

نوع البيانات string في C++

نوع البيانات **string** بنستخدمه لتخزين سلسلة من الأحرف (يعني نصوص أو جمل). بنقدر نخزن فيه أي مجموعة من الأحرف، زي كلمة، جملة، أو حتى فراغات.

مثال:

```
string name = "Mohammad";  
string message = "Welcome to C++!";
```

إيش بيميز **string** عن باقي أنواع البيانات؟

✓ بنقدر نخزن فيه أكثر من حرف أو رمز (مش زي **char** اللي بيخزن حرف واحد فقط)

ملاحظة مهمة:

عشان نقدر نستخدم نوع **string** في برنامجنا، لازم نضم مكتبة

string:

```
#include <string>
```

في معظم الاستخدامات، لأن **string** مش جزء أساسي من أنواع البيانات البدائية مثل (**int** و **char**)

تعريف متغير من نوع **string**

بنعرف متغير من نوع **string** زي أي متغير ثاني:

```
string <variable name> = "<text>";
```

مثال :

```
string city = "Amman";
```

المتغيرات الثابتة (*const variables*)

نوع متغير ثابت، يعني قيمته ما بتتغير بعد تعريفه.

بنعرفه زي المتغير العادي بس بنضيف كلمة **const** ✓

```
const int maxScore = 100;
```

✓ لو حاولت تغيّر قيمته لاحقاً، البرنامج رح يعطي خطأ.

✓ بنستخدمها لما نحتاج قيمة ثابتة زي: النسب، الحدود العليا، القيم

الثابتة.

Cin

`cin >> variableName;`

`cin` : هي اختصار لـ **Console Input**، وتُستخدم لإدخال البيانات من المستخدم عن طريق لوحة المفاتيح خلال وقت تشغيل البرنامج، وتخزينها داخل متغيرات.

تركيب أمر: `cin` 

`cin >> variable;`

- ◆ السهم `>>` معناها "أدخل القيمة في المتغير".
 - ◆ بقدر أستقبل أكثر من قيمة باستخدام `>>` أكثر من مرة.
-

ملاحظة مهمة جداً: 

"لازم نكون معرفين المتغير قبل ما نستخدمه مع `cin`. البرنامج لازم يعرف شو نوع المتغير (`int`, `string`, `float`...) قبل ما نخزن فيه أي قيمة، غير هيك رح يطلع `Error`."

◆ مثال خاطئ:

```
cin >> age; // Error: age
```

مش معرف

```
int age;
```

◆ مثال صحيح:

```
int age;
```

```
cin >> age;
```

■ جمل الـ Prompt – رسائل للمستخدم:

عادةً بنكتب رسالة توضيحية قبل cin نطلب فيها من المستخدم يدخل قيمة، وهذا الاشئ بنسميه **Prompt**، وهي مهمة جداً حتى المستخدم يعرف شو لازم يعمل.

◆ مثال مع Prompt:

```
cout << "Enter your age: ";
```

```
cin >> age;
```

◆ (غير واضح) Prompt مثال بدون:

```
cin >> age; //
```

المستخدم مش رح يعرف شو لازم يدخل

دائمًا اكتب رسالة قبل الإدخال توضح للمستخدم المطلوب منه. 

أمثلة على استخدام cin: 

مثال 1: إدخال عدد صحيح 

```
#include <iostream>
using namespace std;
int main() {
    int age;
    cout << "Enter your age: ";
    cin >> age;
    cout << "You entered: " << age << endl;
    return 0;
}
```


◆ لو المستخدم كتب 20 :

◆ الإخراج:

Enter your age: 20

You entered: 20

■ مثال 2: إدخال أكثر من قيمة

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {
```

```
    int x, y;
```

```
    cout << "Enter two numbers: ";
```

```
    cin >> x >> y;
```

```
    cout << "Sum = " << x + y << endl;
```

```
    return 0;
```

```
}
```

◆ لو المستخدم كتب 5 7 :

◆ الإخراج:

Sum = 12

جدول توضيحي: 

السطر في الكود	القيمة التي دخلها المستخدم	القيمة المخزنة داخل المتغير
cin >> age;	20	age = 20
cin >> x >> y;	5 7	x = 5, y = 7


مثال 3 : إدخال رقم واحد

```
int age;
```

```
cout << "Enter your age: ";
```

```
cin >> age;
```

البرنامج يطلب من المستخدم يدخل عمره، وبخزن القيمة داخل المتغير age

مثال 4: إدخال رقمين 

```
int x, y;
```

```
cout << "Enter two numbers: ";
```

```
cin >> x >> y;
```

البرنامج يطلب من المستخدم يدخل رقمين (يفصل بينهم بمسافة)،
وبيخزنهم داخل المتغيرين x و y

مثال 5: إدخال نص (كلمة وحدة) 

```
string name;
```

```
cout << "Enter your name: ";
```

```
cin >> name;
```

البرنامج يطلب من المستخدم يدخل اسمه، ويخزن أول كلمة فقط
(لحد أول فراغ) داخل المتغير name.

مثال 6 : إدخال عدد عشري 

```
float temp;
```

```
cout << "Enter the temperature: ";
```

```
cin >> temp;
```

البرنامج يطلب من المستخدم يدخل قيمة عشرية (مثل 23.5)،
ويخزنها داخل temp من نوع float.

مثال 7 : إدخال حرف (char) 

```
char grade;
```

```
cout << "Enter your grade (A-F): ";
```

```
cin >> grade;
```

البرنامج يطلب من المستخدم يدخل حرف واحد مثلاً A أو B أو C
وبيخزنه داخل متغير من نوع char.

✓ مثال 8 : إدخال قيمة منطقية (bool)

```
bool isPassed;
```

```
cout << "Did you pass? (1 = yes, 0 = no): ";
```

```
cin >> isPassed;
```

البرنامج يطلب من المستخدم يدخل 1 أو 0، وبيخزن القيمة داخل
متغير isPassed من نوع bool

True = 1

False = 0

✗ أخطاء شائعة عند استخدام cin

1. ● استخدام cin قبل تعريف المتغير

الخطأ: الطالب يكتب cin >> x; بدون ما يعرف x قبلها.

cin >> x; // ✗ Error: 'x' was not declared

int x;

الحل: 

```
int x;
```

```
cin >> x;
```

2. Prompt نسيان جملة ال- (الإيضاح للمستخدم)

الخطأ: ما بحكي للمستخدم شو لازم يدخل.

```
cin >> age; //  Error: 'cin' was not declared
```

الحل: 

```
cout << "Enter your age: ";
```


```
cin >> age;
```

3. نسيان `#include <iostream>`

الخطأ: الكود ما فيه المكتبة المسؤولة عن cin و cout.

```
int main() {
```

```
    int x;
```

```
    cin >> x; //  Error: 'cin' was not declared
```

```
}
```

الحل: 

```
#include <iostream>

using namespace std;
```

```
int main() {


    int x;

    cin >> x;

}
```

4.  استخدام = بدل >>

الخطأ: الطالب يكتب cin = x; بدل cin >> x;

cin = x; //  هذا مش إدخال، هذا محاولة تخزين!

الصحيح: 

cin >> x;

5.  إدخال نوع غير متوافق

الخطأ: المستخدم بيدخل قيمة مش من نوع المتغير (مثلاً يكتب كلمة
int) داخل

```
int age;
```

```
cin >> age;
```

◆ لو كتب المستخدم twenty :

◆ البرنامج ما رح يفهم القيمة، وبيضل المتغير بدون قيمة صحيحة.

الحل: 

تأكد إنك تدخل قيمة مناسبة لنوع المتغير (int = أرقام، string = كلمات...)

6. نسيان استخدام `using namespace std;`

الخطأ: نسيان سطر `using namespace std;` يخلي `cin` و `cout` يعطوا Error.

```
#include <iostream>
```

```
int main() {
```

```
    int x;
```

```
    cin >> x; //  Error
```

```
}
```

الحل: 

```
#include <iostream>
```



```
using namespace std;
```



```
int main() {
```

```
    int x;
```

```
    cin >> x; //  شغال تمام
```

```
}
```

هل بقدر أأزن قيمة من نوع معين داخل متغير من نوع ثاني؟

الإجابة:  أحيانًا ممكن وببشتغل عادي
لكن  أحيانًا ما بيمشي الحال، أو بيعطي نتيجة غير متوقعة.

ملاحظات أساسية:


1. بعض أنواع البيانات بتقدر تخزن فيها أنواع ثانية بدون مشاكل
مثلاً: تخزن `int` داخل `float`

2. بعض الحالات الثانية بتأدي لفقدان جزء من القيمة، أو تغيير معناها زي تخزين `float` داخل `int`

3. أنواع زي `string` ما بتخزن فيها أنواع ثانية زي `int` أو `float` مباشرة.

4. `char` لما نخزن فيه رقم، بتم اعتباره رمز من جدول `ASCII`.

5. بعض أرقام الـ `ASCII` ما بتمثل رموز مفهومة، فممكن الطباعة تطلع رموز غريبة أو ما تطبع شيء نهائيًا.

6.  مسموح نخزن `char` داخل `string`، لأنه `string` بيقبل نصوص حتى لو كانت حرف واحد.

أمثلة مهمة :

مبدأ:

"أي محاولة تخزين فيها قيمة بنوع مختلف عن نوع المتغير، هي بشكل أو بآخر اسمها **Casting**، سواء تمت تلقائيًا أو يدويًا".

جدول شامل أمثلة على التخزين بين أنواع البيانات

الشرح	هل يُسمح؟	المثال	نوع القيمة	نوع المتغير
يتم تقريب العدد تلقائيًا إلى 5 (يفقد الجزء العشري).	✓	int x = 5.6;	float	int
يتحول إلى 3.0 تلقائيًا.	✓	float f = 3;	int	float
يتم توسيع القيمة تلقائيًا.	✓	double d = 4.5f;	float	double
يتم تحويل الرقم إلى رمز من ASCII ('B').	✓	char c = 66;	int	char

int	char	int x = 'A';	✓	يتم تحويل الحرف إلى كوده الرقمي (65).
bool	int	bool b = 0;	✓	0 = false ، وأي رقم ≠ true.0
bool	int	bool b = 8;	✓	يتحول إلى true.
string	int	string s = 5;	✗	خطأ: لا يمكن تخزين رقم داخل string مباشرة.
string	char	string s = 'A';	✓	مقبولة: يتم تحويل الحرف إلى نص بطول حرف واحد.
short	int (كبير)	short s = 100000;	⚠	ممكن تحدث مشكلة أو خسارة بيانات لو الرقم أكبر من قدرة short.
long	int	long l = 7;	✓	مسموح وبدون مشاكل.

float	string	float f = "3.2";	✗	خطأ: لا يمكن تحويل نص إلى float مباشرة.
-------	--------	---------------------	---	---

ملاحظات مهمة :

1. ✗ ممنوع تخزين string داخل أنواع مثل int أو float مباشرة.

◦ في طريقة بتحول النص إلى رقم أو العكس، لكنها مش مطلوبة في هذا الكورس.

2.  القيم المخزنة في char بتعتمد على جدول: ASCII

◦ الرقم 65 يعني الحرف 'A'

◦ الرقم 66 يعني 'B'

◦ وهكذا...

3. ⚠ بعض الأرقام في char بما يتمثل رموز معروفة:

◦ ممكن تطلع رموز غريبة

◦ أو ما تطبع شيء نهائيًا

 مثال يوضح نقطة: ASCII

```
char ch = 65;
```

```
cout << ch << endl; // الناتج: A
```

لكن:

```
char ch = 7;
```

```
cout << ch << endl; // الناتج: صوت أو رمز غريب أو لا شيء
```

● مش كل رقم بنخزنه داخل char رح يطلع معنا واضح أو مفهوم عند الطباعة.

كل اللي شفناه لحد الآن هو عبارة عن حالات تخزين بين أنواع بيانات مختلفة. مرات C++ بتسمحك تعمل هيك تلقائيًا، ومرات لازم تحول النوع بنفسك. وهون بيجي دور إشي اسمه "Casting."

Casting in C++

1. تعريف Casting.

Casting هي عملية تحويل قيمة من نوع بيانات (Data Type) إلى نوع بيانات آخر، مثل التحويل من `int` إلى `float` أو من `double` إلى `int...` إلخ.

2. لماذا نستخدم الـ Casting؟

نستخدم الكاستنج حتى:

- لمنع فقدان جزء من القيمة
 - نتحكم بالنتائج (خاصة في العمليات الحسابية)
 - نحول القيم حسب النوع المتوقع للمتغير
-

3. طرق كتابة الكاستنج (Casting Syntax) المختصرة :

في C++ في طريقتين رئيسيتين لكتابة الكاستنج:

◆ 3.1 C-style Casting (الطريقة الكلاسيكية)

(dataType) value;

◆ مثال:

```
int x = 5;
```

```
float y = (float)x;
```

◆ 3.2 Functional-style Casting (الطريقة المختصرة)

```
dataType(value);
```

◆ مثال:

```
int x = 5;
```

```
float y = float(x);
```

الطريقتين يعطوا نفس النتيجة.

4. ✓ أنواع الكاستنج (Types of Casting)

◆ 4.1 Implicit Casting (التحويل التلقائي)

C++ أحيانًا بتعمل تحويل تلقائي بدون ما نكتب أي أمر، من نوع أصغر إلى نوع أكبر.

```
int a = 7;
```

```
double d = a; // int → double
```

التحويل تم تلقائيًا لأن double أوسع من int.

◆ 4.2 Explicit Casting (التحويل الصريح)

هون إحنا بنكتب أمر التحويل بشكل واضح في الكود، إذا كنا بدنا نتحكم بالتحويل بأنفسنا.

◆ مثال بالصيغتين:

```
double d = 5.7;
```

```
int x = (int)d; // C-style
```

```
int y = int(d); // Functional-style
```

(x = 5، y = 5) رح يفقد الجزء العشري

5.  أمثلة على الكاستنج داخل `cout`

مثال 1: بدون كاستنج (نتيجة غير متوقعة)

```
cout << 5 / 2; // الناتج: 2
```

فالنتيجة بتكون عدد صحيح `int / int` لأنهم

مثال 2: مع كاستنج (نتيجة عشرية)

```
cout << (float)5 / 2; // الناتج: 2.5
```

مثال 3: باستخدام Functional-style

`cout << float(5) / 2; //` الناتج: 2.5

أمثلة متنوعة على Casting


نوع الكاستنج النتيجة الكود		
<code>(float)3 / 2</code>	1.5	Explicit
<code>double d = 4;</code>	4.0	Implicit
<code>int x = (int)4.9;</code>	4	Explicit
<code>cout << (char)65;</code>	A	Explicit
<code>bool b = 5;</code>	true	Implicit
<code>int x = 'A';</code>	65	Implicit


جدول أمثلة على استخدام static_cast في C++

التفسير	النتيجة أو المكافئ	السطر باستخدام static_cast
تم تحويل العدد العشري إلى عدد صحيح بفقدان الجزء العشري	cout << 7;	cout << static_cast<int>(7.9);
الرقم 66 يقابل الحرف 'B' حسب ASCII	cout << 'B';	cout << static_cast<char>(66);
'C' الحرف يتحول إلى كوده الرقمي 67	cout << 67;	cout << static_cast<int>('C');
'D' الحرف يتحول إلى كود 68، ثم إلى double	cout << 68.0;	cout << static_cast<double>('D');
ASCII في 33 هو الرمز '!'	cout << '!';	cout << static_cast<char>(33);

<code>cout << static_cast<bool>(0);</code>	<code>cout << 0;</code>	يتم تفسيره كـ 0 false
<code>cout << static_cast<bool>(123);</code>	<code>cout << 1;</code>	أي رقم $0 \neq$ true يعتبر
<code>cout << static_cast<int>(true);</code>	<code>cout << 1;</code>	يتحول true إلى 1
<code>cout << static_cast<int>(false);</code>	<code>cout << 0;</code>	يتحول false إلى 0
<code>cout << static_cast<char>('A' + 2);</code>	<code>cout << 'C';</code>	'A' = 65 ، + 2 = 67 = 'C'

ملاحظات مهمة

1.  التحويل التلقائي (Implicit) بيصير من نوع أصغر لنوع أكبر.

2.  التحويل اليدوي (Explicit) بنكتبه لما:

◦ نخاف نفقد جزء من القيمة (زي حذف الجزء العشري)

◦ أو النوعين مش بيدعمو التحويل التلقائي

3.  التحويل من وإلى string بشكل مباشر مش مسموح،
وبدّه طرق خاصة مش مطلوبة في هاد الكورس.

سؤال شامل عالکامل:

أكتب برنامج بلغة C++ يقوم بما يلي:

1. يطلب من المستخدم إدخال البيانات التالية:

- الاسم الأول (First Name)
- الاسم الأخير (Last Name)
- عدد صحيح (Integer)
- عدد عشري (Float)
- حرف واحد (Character)
- كلمة واحدة فقط من نوع String

2. يخزن القيم المدخلة في متغيرات مناسبة.

3. يطبع البيانات المدخلة بالشكل التالي:

- الاسم الكامل (Full Name)
- العدد الصحيح
- العدد العشري
- الحرف
- الكلمة (String)

4. يقوم بتنفيذ التحويلات التالية باستخدام `static_cast`:

- تحويل العدد الصحيح إلى عدد عشري.
- تحويل العدد العشري إلى عدد صحيح.
- تحويل الحرف إلى كوده الرقمي. (ASCII code)

5. يعرف متغيرين من نوع `bool`:

- الأول يحتوي القيمة 0.
- الثاني يحتوي القيمة 100.
- ثم يطبع القيمتين.

6. يعرف متغير جديد من نوع `string` ويخزن فيه الحرف المدخل مباشرة، ثم يطبع هذا المتغير.

💡 ملاحظة: لا تستخدم `getline()`، أدخل كلمة واحدة فقط في المتغير النصي.

Input Example:

Ahmed

Salameh

7

4.2

B

World

 **Expected Output:**

Full name: Ahmed Salameh

Integer: 7

Float: 4.2

Character: B

String: World

Casting operations:

Integer to float: 7.0

Float to integer: 4

Character to ASCII: 66

Boolean values:

bool b1 (0): 0

bool b2 (100): 1

Character as string: B

الحل النموذجي : 

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
int main() {
```

```
    // تعريف المتغيرات
```

```
    string firstName, lastName;
```

```
    int myInt;
```

```
    float myFloat;
```



```
char myChar;
```

```
string myString;
```

```
// الإدخال من المستخدم
```

```
cout << "Enter your first name: ";
```

```
cin >> firstName;
```

```
cout << "Enter your last name: ";
```

```
cin >> lastName;
```

```
cout << "Enter an integer: ";
```

```
cin >> myInt;
```

```
cout << "Enter a float: ";
```

```
cin >> myFloat;
```

```
cout << "Enter a character: ";
```

```
cin >> myChar;
```

```
cout << "Enter a string (one word): ";
```

```
cin >> myString;
```

```
// الطباعة
```

```
cout << "\nFull name: " << firstName << " " <<  
lastName << endl;
```

```
cout << "Integer: " << myInt << endl;
```

```
cout << "Float: " << myFloat << endl;
```

```
cout << "Character: " << myChar << endl;
```

```
cout << "String: " << myString << endl;
```

```
// الكاستنج
```

```
cout << "\nCasting operations:\n";
```

```
cout << "Integer to float: " <<  
static_cast<float>(myInt) << endl;
```

```
cout << "Float to integer: " <<  
static_cast<int>(myFloat) << endl;
```

```
cout << "Character to ASCII: " <<  
static_cast<int>(myChar) << endl;
```

// المتغيرات البوليانية

```
bool b1 = 0;
```

```
bool b2 = 100;
```

```
cout << "\nBoolean values:\n";
```

```
cout << "bool b1 (0): " << b1 << endl;
```

```
cout << "bool b2 (100): " << b2 << endl;
```

// تخزين الحرف داخل متغير سترينج مباشرة

```
string charAsString = myChar;//error
```

```
cout << "\nCharacter as string: " << charAsString  
<< endl;//error
```

```
return 0;
```

```
}
```

Mathematical Operations in C++

هي العمليات الرياضية الأساسية (+ - * /) والتي بنستخدمها داخل برامج C++ لحساب القيم.

ملاحظات على العمليات:

1. الـ bool يتحول تلقائيًا لـ 0 أو 1، ويتعامل كأنها int.
 2. الـ char يتحول لقيمتها في جدول ASCII ويتصير int.
 3. إذا العملية بين int و int → الناتج int.
 4. إذا العملية بين int و double → الناتج double.
 5. النوع الأصغر دائمًا يتحول تلقائيًا للنوع الأكبر (promotion).
-

أمثلة على العمليات:

التفسير	الناتج	الكود
الأولوية للضرب	20	<code>cout << 10 + 5 * 2;</code>
int + doubl e → الناتج doubl e	9.5	<code>cout << 7 + 2.5;</code>
قسمة int → بدون كسور	2	<code>cout << 9 / 4;</code>
ناتج doubl e	2.2 5	<code>cout << 9.0 / 4;</code>
B = 66 في ASCII	67	<code>cout << 'B' + 1;</code>

<code>cout << true + 6;</code>	7	true = 1
<code>cout << static_cast<double>(15) / 2;</code>	7.5	تحويل صريح إلى double
<code>cout<<static_cast<double>(15/2) ; //= cout<<static_cast<double>(7)=7. 0</code>	7	

● العملية الخاصة % Modulus :

رمز % يعني: "أعطني باقي القسمة فقط".

📌 لازم تنتبه:

✓ الرقم اللي بدنا نعرف باقي قسمته (المقسوم) → على اليسار

✓ الرقم اللي بدنا نقسم عليه (المقسوم عليه) → على اليمين

✗ ما بصير تعكس الترتيب إلا لو كنت فاهم شو بتسوي

أمثلة واضحة:

الكود	النتج	التفسير
19 % 3	1	$19 \div 3 = 6$ والباقي 1
20 % 5	0	$20 \div 5 = 4$ بدون باقي
34 % 2	0	كل عدد زوجي باقي قسمته على 2 = 0
33 % 2	1	عدد فردي \rightarrow باقي = 1
5 % 8	5	لأن 5 أصغر من 8، فالجواب هو نفسه 5
-5 % 4	-1	الإشارة تتبع الرقم الأول
5 % -4	1	الإشارة بتضل موجبة لأن المقسوم موجب
7 % 0	 Error	القسمة على صفر غير مسموحة

معلومة مهمة:

. العملية % لازم تكون بين نوعين int فقط.

. ما بصير تستخدم % مع double.

ملخص سريع:

الناتج	الحالة
✓ مسموح	int % int
✗ Error	int % 0
✗ Error	double % double أو double % int

دائماً راقب نوع المتغيرات لما تستخدم العمليات الرياضية.

لو كنت مش متأكد من نوع الناتج، استخدم static_cast لتحديد النوع بشكل صريح.




العمليات بين int و double ممكن تعطيك ناتج غير اللي متوقعه لو ما انتبهت.


✓ Increment (++)
✓ Decrement (--)

هي عمليات لزيادة أو نقصان قيمة المتغير بمقدار واحد فقط.

أنواع البيانات اللي بتشتغل معها:

نوع المتغير	هل يمكن استخدام ++ أو --؟
int	✓ نعم
double	✓ نعم

char	 نعم (يتغير ASCII code)
bool	 لا
string	 لا

 أنواع الاستخدام:

◆ Pre-Increment / Pre-Decrement

--X; أو ++X;

- أول إشي: بزيد أو بنقص قيمة المتغير
- بعدين: بتكتب القيمة الجديدة داخل الجملة

◆ Post-Increment / Post-Decrement

x--; أو ++x; //

- أول إشي: الجملة بتتنفذ باستخدام القيمة الحالية
- بعدين: بزيد أو بنقص المتغير

 أمثلة مع الشرح التفصيلي:

مثال 1 Pre-Increment: في جملة

```
int x = 3;  
cout << ++x << endl;
```

التحليل:

. ++x أول شيء زاد x من 3 إلى 4

. بعدين طبعها

Output:

4

مثال 2 Post-Increment: في جملة

```
int x = 3;  
cout << x++ << endl;
```

التحليل:

. طبع x وهي 3

. بعدين زاد x وصارت 4

Output:

3

مثال 3 Pre-Decrement: داخل جملة

```
int x = 5;  
cout << --x + 2 << endl;
```

التحليل:

. أول إشي $--x$: صارت 4

. بعدين $6 = 2 + 4$

Output:

6

مثال 4 Post-Decrement: داخل جملة

```
int x = 5;  
cout << x-- + 2 << endl;
```

التحليل:

. أول شيء استخدم x وهي 5 $\rightarrow 5 + 2 = 7$

. بعدين نقص x فصارت 4

Output:

7

✓ مثال 5: استخدام داخل معادلة

```
int x = 2;
int y = 3;
int result = ++x + y++;
cout << "x = " << x << ", y = " << y << ", result = " <<
result << endl;
```

التحليل خطوة بخطوة: 🔍

• ++x صارت 3

• ++y استخدم 3، وبعدين y صارت 4

• النتيجة: $6 = 3 + 3$

Output:

x = 3, y = 4, result = 6


✓ مثال 6: مع char

```
char c = 'A';
cout << ++c << endl;
```

🔍 A → ASCII = 65 → ++c = 66 → B

Output:

B

مثال 7: للتأكيد على توقيت التنفيذ 

```
int a = 5;
```

```
cout << "قبل التنفيذ" << a << endl;
```

```
cout << "داخل الجملة" << a++ << endl;
```

```
cout << "بعد التنفيذ" << a << endl;
```

Output:

قبل التنفيذ: 5

داخل الجملة: 5

بعد التنفيذ: 6



ملخص سريع (مهم جداً):

النوع	الترتيب	مثال	النتيجة إذا $x = 2$
Pre-Increment	زيد أولاً، نفذ بعدين	$++x$	3
Post-Increment	نفذ أولاً، زيد بعدين	$x++$	(ثم يصير 3) 2
Pre-Decrement	نقص أولاً، نفذ بعدين	$--x$	1
Post-Decrement	نفذ أولاً، نقص بعدين	$x--$	(ثم يصير 1) 2

جدول تدريبات على Increment/Decrement مع تفسير واضح

الملاحظة أو الشرح	الـ Output	الكود	رقم
Pre-Increment: أول إشـي زاد a من 5 إلى 6، وبعدين طبع القيمة الجديدة.	6	int a = 5; cout << ++a;	1
Post-Increment: طبع a أول إشـي (5)، بعدين زادها لـ 6 بالذاكرة.	5	int a = 5; cout << a++;	2
x++ استخدمت 3 بالجملة، بعدها جمعت مع 2، النتيجة 5. بعد السطر x بصير 4.	5	int x = 3; cout << x++ + 2;	3
x++ زادت x من 3 لـ 4، وبعدين جمعت مع 2، الناتج 6.	6	int x = 3; cout << ++x + 2;	4
y = x++ يعني خزن قيمة x الأصلية (4) في y، ثم زاد x لـ 5.	5 4	int x = 4; int y = x++; cout << x << " " << y;	5

6	<pre>int x = 4; int y = ++x; cout << x << " " << y;</pre>	5 5	<p>++x زادت x أولاً لـ 5، ثم خزنها في y، فصارت القيمتين نفس الشيء.</p>
7	<pre>int a = 5; int b = a++ + ++a; cout << a << " " << b;</pre>	⚠ غير موصى به	<p>تم استخدام ++ لنفس المتغير مرتين بنفس الجملة، وهذا ممكن يعطي نتائج غير متوقعة (سلوك غير معرف).</p>
8	<pre>char c = 'D'; cout << c++; cout << c;</pre>	D E	<p>طبع أول إشي D (قبل التغيير)، وبعدها زاد القيمة لـ E وطبعها.</p>
9	<pre>int a = 3; int b = 4; int r = ++a + b++; cout << a << " " << b << " " << r;</pre>	4 5 8	<p>++a زادت a لـ 4، b++ استخدمت 4 وبعدين صارت 5. النتيجة: 8=4+4.</p>
10	<pre>int x = 10; cout << x--; cout << --x;</pre>	10 8	<p>--x طبع 10 ثم نقصه لـ 9، بعدين --x نقص كمان مرة لـ 8 وطبعها.</p>

11	<pre>int a = 1; cout << a++ + ++a + a;</pre>	7	<p>،++a = 3 ،a++ = 1 بعدها .a = 3. المجموع: 1 7 = 3 + 3 +</p>
12	<pre>int x = 0; cout << x++ << " " << ++x << " " << x;</pre>	0 2 2	<p>أول شيء طبع 0، ثم زاد لـ 1، بعدها ++x صارت 2، وآخر شيء طبع x وهي 2.</p>

✓ أولوية تنفيذ العمليات الحسابية (Precedence of Operations) في C++

لما نكتب تعبير (expression) يحتوي على أكثر من عملية حسابية، مش كل العمليات بتننفذ بنفس الترتيب، في أولوية محددة بتمشي عليها C++ علشان تعرف مين تنفذ أول.

جدول ترتيب الأولويات (من الأعلى إلى الأقل):

الترتيب	اسم العملية	الرمز / الشكل	مثال
1	Increment / Decrement	++, --	x++, --y
2	الأقواس	()	(a + b)
3	ضرب / قسمة / باقي قسمة	*, /, %	a * b, x % y
4	جمع / طرح	+, -	x + y - z
5	التخزين (Assignment)	=	x = y + 3

ملاحظات مهمة:

. إذا كانت عمليتين بنفس المستوى زي + و - نمشي من اليسار لليمين.

. الأقواس () بتتحكم في الترتيب وبتخلي اللي بداخلها يتنفذ أول.

أمثلة تدريبية مع التفسير:

✓ مثال 1:

```
int result = 2 + 3 * 4;
```

Output:

14

الشرح:

الضرب أول ($12 = 4 * 3$)، بعددين الجمع. ($2 + 12 = 14$)

✓ مثال 2:

```
int result = (2 + 3) * 4;
```

✓ **Output:**

20

الشرح:

الأقواس أول ($5 = 3 + 2$)، بعددين ضرب. ($5 * 4 = 20$)

مثال 3: ✓

```
int result = 10 / 3;
```

✓ **Output:**

3

الشرح:

لأنهم int، القسمة رح تعطي عدد صحيح فقط (بدون كسور).

مثال 4: ✓

```
int result = 13 / 6 - 2;
```

Output:

0

الشرح:

$13 / 6 = 2 \rightarrow$ بعدين $2 - 2 = 0$

مثال 5: ✓

```
int x = 5, y = 3;
```

```
int result = x - (--y) + x;
```

Output:

8

الشرح:

. --y: قلل y لـ 2 أول إشي

. العملية $5 - 2 + 5 = 8$

مثال 6: 

```
int x = 2;
```

```
x = x++ + 1;
```

Output of x:

3

الشرح:

. $x++ = 2$ ، وبعدين x صار 3

. الجملة صارت $x = 2 + 1 = 3$

مثال 7: 

```
char ch = 'A';
```

```
int x = 2;
```

```
double d = 99.2;
```

```
char result = ch + static_cast<char>(x +  
static_cast<int>(d));
```

Output (as char):

A + 101 = char(166) → ASCII حسب الجدول

الشرح:

int(d) = 99 → x + 99 = 101 .

static_cast<char>(101) = .
101
ASCII المقابل للحرف

A = 65 → 65 + 101 = 166 → char(166) .

مثال 8: 

```
int x = 6, y = 2;
```

```
int result = -x - y + x;
```

Output:

-6 - 2 + 6 = -2

الشرح:

العمليات تمشي من اليسار لليمين بنفس المستوى (الطرح)، فبطلع
النتائج 2-

 ملخص سهل للحفظ:

الأولويات :

1. ++ -- ← أول إشي

2. () ← أي إشي داخل أقواس

3. % / * ← ضرب، قسمة، باقي القسمة

4. + - ← جمع وطرح

5. = ← تخزين

السؤال الأول: حساب مساحة مستطيل ✓

اكتب برنامج بلغة C++ يطلب من المستخدم إدخال الطول والعرض
لمستطيل، ثم يقوم بحساب المساحة باستخدام القانون:
المساحة = الطول × العرض:

مثال: (Output)	مثال: (Input)
The area of the rectangle is: 28	Enter length: 7 Enter width: 4

الحل (الكود):

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

int main() {
    int length, width;
```



```
cout << "Enter length: ";  
cin >> length;  
  
cout << "Enter width: ";  
cin >> width;  
  
int area = length * width;  
cout << "The area of the rectangle is: " << area  
<< endl;  
  
return 0;  
}
```

السؤال الثاني: حساب مساحة دائرة

اكتب برنامج بلغة C++ يطلب من المستخدم إدخال نصف القطر
لدائرة، ثم يقوم بحساب مساحتها باستخدام القانون:

$$\text{المساحة} = \pi \times r^2$$

حيث أن قيمة π تساوي 3.14، ويجب تعريفها داخل البرنامج
كمتغير ثابت const.

مثال:(Input)	مثال:(Output)
Enter radius: 5	The area of the circle is: 78.5

الحل (الكود):

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

int main() {
    const double PI = 3.14;
    double radius;
```

```
cout << "Enter radius: ";  
  
cin >> radius;  
  
double area = PI * radius * radius;  
  
cout << "The area of the circle is: " << area <<  
endl;  
  
return 0;  
}
```

Compound assignment

Operator	Example	Longer Expression	Description
+=	a += b	a = a + b	Add, then assign
-=	a -= b	a = a - b	Subtract, then assign
*=	a *= b	a = a * b	Multiply, then assign
/=	a /= b	a = a / b	Divide, then assign
%=	a %= b	a = a % b	Compute remainder, then assign

Concatenation in C++

الـ Concatenation هي عملية دمج نصوص (Strings) مع بعض باستخدام علامة الجمع +. وهي العملية الوحيدة المسموحة على الـ string باستخدام الـ + في C++.

لكن لازم يكون في بينهم على الأقل متغير واحد من نوع string ، ومش مسموح تجمع قيمتين نصيتين ثابتتين مباشرة باستخدام +.

الكود (Code)	الشرح	الناتج (Output)
<pre>string word = "Uni"; cout << word + "versity";</pre>	<p>هون عندنا متغير word نوعه string، وجمعنا عليه نص ثابت باستخدام .+ العملية مسموحة.</p>	University
<pre>string part1 = "Hashe", part2 = "mite"; cout << part1 + part2;</pre>	<p>دمجنا متغيرين من نوع string. العملية صحيحة وتنتج النص المدموج.</p>	Hashemite
<pre>string middle = "vers"; cout << "\"Uni" + middle + "ity\"";</pre>	<p>استخدمنا متغير بالنص مع نصوص ثابتة، وحطينا علامات اقتباس باستخدام \"</p>	"University"

<code>cout << "Hello" + "World";</code>	✗ خطأ! لأنه حاولنا نجمع قيمتين نصّيتين ثابتتين بدون استخدام متغير	خطأ وقت الترجمة (Compilation Error)
---	---	--

ملاحظات مهمة:

- ✓ مسموح تجميع `string + string` أو `string_variable` + نص ثابت
- ✗ غير مسموح: نص ثابت + نص ثابت بدون متغير

Error Types في C++

بالواقع الأخطاء (Errors) في البرمجة مش نوع واحد، هم ثلاث أنواع رئيسية، وكل نوع منهم بكون بسبب خلل مختلف عن الثاني

الشرح الكامل

الكود /

المثال


<pre>int x = 5; cout << "x is: " << x</pre>	<p>نوع الخطأ Syntax Error: خطأ كتابة شو بصير؟ البرنامج ما يشتغل نهائياً وبيطلع Error من الكمايلر كيف نعرفه؟ الكمايلر بيكتب رسالة زي: 'expected ';' before 'cout' كيف نصلحه؟ نرجع نضيف الفاصلة المنقوطة ؛ بنهاية السطر الأول</p>
<pre>int a = 4, b = 2; cout << a - b * 2;</pre>	<p>نوع الخطأ Semantic Error: خطأ منطقي شو بصير؟ البرنامج يشتغل وبيطبع 0: ليش؟ لأن الضرب إله أولوية أعلى من الطرح، فصار (2 * 4 - 2) كيف نصلحه؟ نضيف أقواس حتى نغير الترتيب * (a - b) << cout: 2;</p>
<pre>int x = 10, y = 0; cout << x / y;</pre>	<p>نوع الخطأ Run-Time Error: خطأ وقت التشغيل شو بصير؟ البرنامج ببش، وبعدين بيوقف فجأة أو بعمل Crash ليش؟ لأنه فيه قسمة على صفر كيف نصلحه؟ نستخدم شرط يمنع القسمة على صفر .</p>

```
int x = 3,
y = 6, z =
    9;

cout <<
"Average:
" << x + y
+ z / 3;
```

نوع الخطأ Semantic Error: شو بصير؟ بيطلع نتيجة غلط لأن القسمة تمت قبل الجمع كيف نصلحه؟ نضيف أقواس لتغيير الترتيب / $(x + y + z)$: 3.0 أو نحول أحد الأطراف لـ double

:

مثال على خطأ منطقي وحله (المتوسط الحسابي): 

✗ الكود الخطأ:

```
int x = 3, y = 6, z = 9;
cout << "Average: " << x + y + z / 3;
```

. بيطلع نتيجة غلط لأن القسمة صارت قبل الجمع (بسبب الأولويات)

✓ الكود الصحيح:

```
cout << "Average: " << (x + y + z) / 3.0;
```

. هون استخدمنا أقواس و Double للتأكد من دقة الحساب

🧠 كيف أميز نوع الخطأ إذا مش عارفه؟

نوع الخطأ	الملاحظة
Syntax Error	البرنامج ما يشتغل نهائياً، وبيطلع رسالة Error من الكمبايلر
Semantic Error	البرنامج يشتغل، بس النتيجة مش اللي متوقعها
Run-Time Error	البرنامج يشتغل وبعدين بيوقف فجأة أو بطلع خطأ خلال التشغيل

أشياء مهمة لازم نحفظها او نكون عارفينها :

المعنى	المصطلح
هو البرنامج المسؤول عن ترجمة كود C++ للغة الآلة، وبفحص إذا فيه Syntax Errors	Compiler
عملية فحص الكود وتحويله لشي جاهز للتنفيذ	Compile

Run / Debug / Start	كلهم يعملوا تشغيل للكود، وبيتم قبلهم عمل Compile تلقائي
ملف .cpp	هو الملف اللي بكتب فيه المبرمج الكود، مثال main.cpp :
ملف .obj	بيتم إنشاؤه بعد الـ Compile ، وهو نسخة جزئية مترجمة من الكود
Runtime Error	هو الخطأ اللي ما بنشوفه إلا لما نعمل تشغيل للكود فعلياً
Semantic Error	الكود يشتغل، لكن بيطلع نتيجة غلط بسبب خلل بالمنطق أو الحساب
Syntax Error	الكود ما يشتغل نهائياً، بسبب خطأ بالكتابة أو التنسيق

أولاً : شرح مفهوم الذاكرة (Memory) في C++

الفكرة الأساسية:

قبل ما ندخل بموضوع Pointers ، لازم نفهم كيف الكمبيوتر
بتعامل مع المتغيرات، وين بيخزنهم، وإيش يعني "ذاكرة".

تخيل الذاكرة كأنها جدول كبير فيه خلايا (memory blocks) ،
كل خلية إلها عنوان مميز (Address) ، والخلية هاي ممكن نخزن
فيها قيمة.

قبل ما نحط أي إشي في الذاكرة:

الذاكرة بتكون مليانة قيم عشوائية اسمها **rubbish data** مثل:

3jd93d, ff9n00, 39hheu, ju839i ...

هاي القيم ما إلها معنى، ومش من شغلنا.

لما نعرف متغير:

لما نكتب:

```
int x = 2;
```

الكمبايلر بروح بحجز خلية أو أكثر من الذاكرة حسب نوع المتغير

مثلاً `int` تحجز `bytes4`

وبيعطيه عنوان خاص مثلاً 4385،

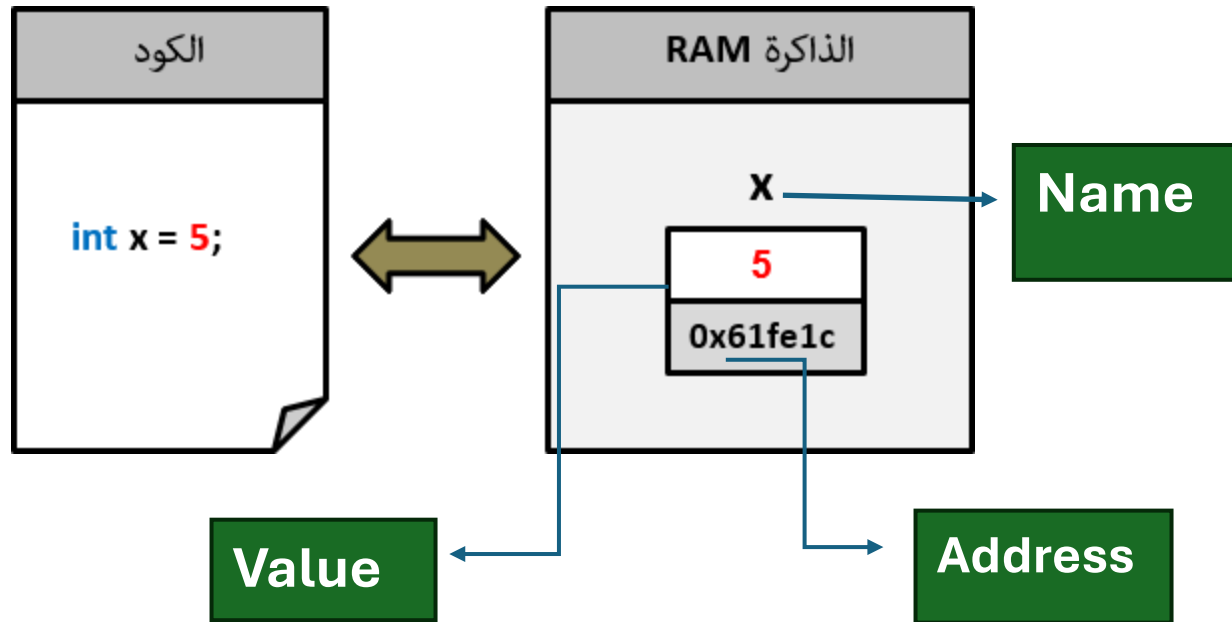
وبخزن داخلها القيمة 2

وإحنا بنصير نقدر نوصل لهاي الخلية عن طريق اسم المتغير `x`.

اسم المتغير هو اسم مستعار (alias) لعنوان الذاكرة.

الشكل التخيلي للذاكرة بعد تعريف مجموعة متغيرات:

المتغير	النوع	القيمة	العنوان (تخيلي)
<i>x</i>	int	2	4385
<i>ch</i>	char	's'	5520
<i>d</i>	double	3.23	6620
<i>b</i>	bool	true	8910
<i>str</i>	string	"cpp"	7032



كل متغير إليه:

. نوع بيانات (Data Type)

. قيمة (Value)

. عنوان بالذاكرة (Address)

ملاحظات مهمة:

1. ترتيب المتغيرات في الذاكرة مش شرط يكون بنفس الترتيب
اللي كتبناهم فيه بالكود.

2. العنوان (Address) هو رقم فريد لكل خلية، لكن أنت ما
بتتعامل معه عادة إلا لما تدخل في مواضيع البوينتر.

3. الوصول للقيمة بيكون من خلال اسم المتغير، لكن ممكن كمان
نوصل للـ Address زي ما رح نشوف بالبوينتر

خلاصة الجزء الأول:

. الذاكرة عبارة عن خلايا، كل خلية إليها عنوان.


. كل متغير بنكتبه بتم حجز مساحة إليها بالذاكرة حسب نوعه.

. القيمة تنحزن داخل الخلية، واسم المتغير هو اسم مستعار لهاي
الخلية.

. قبل تعريف المتغيرات، الخلايا مليانة بيانات عشوائية (rubbish data).

. ترتيب المتغيرات في الذاكرة مش تسلسلي دائماً.

ثانياً :- Pointers

 الفكرة الأساسية:

كل متغير بيتخزن في خلية بالذاكرة، وكل خلية إلها عنوان (Address).

إذا بدنا نخزن عنوان متغير داخل متغير ثاني، بنستخدم إشي اسمه Pointer.

يعني:

 البوينتر = متغير يخزن عنوان متغير ثاني

 تعريف البوينتر:

نستخدم الرمز * عند تعريف البوينتر:

متغير عادي // `int x = 5;`

`int* p;` // بوينتر اسمه p يخزن عنوان متغير `int`

`p = &x;` // خزنا في البوينتر p عنوان المتغير x

✖ مكونات السطر:

• `int* p;` معناها: متغير `p` يحمل عنوان متغير من نوع `int`

• `&x` معناها: عنوان المتغير `x`

• `p = &x;` معناها: خزّن العنوان داخل البوينتر

📌 كيف نستخدم البوينتر؟

النتيجة	السطر	الاستخدام
بيطبّع عنوان المتغير <code>x</code>	<code>cout << p;</code>	طباعة العنوان
بيطبّع القيمة اللي داخل <code>x</code> يعني 5)	<code>cout << *p;</code>	طباعة القيمة اللي بالعنوان
هيك غيرنا قيمة <code>x</code> عن طريق البوينتر	<code>*p = 10;</code>	تعديل القيمة

مبدأ مهم:

الرمز	المعنى
<code>&x</code>	عنوان المتغير <code>x</code>

*p	القيمة الموجودة في العنوان اللي البوينتر p يشير عليه
----	--

شكل الذاكرة بعد تعريف متغير وبوينتر:

العنوان	القيمة	النوع	الاسم
4385	5	int	x
9000	4385	int*	p

يعني:

• p يخزن العنوان تبع x

• *p توصلنا لقيمة x

ملاحظات مهمة:

1. لازم نوع البوينتر يطابق نوع المتغير، يعني:

• `int* p;` لازم يخزن عنوان متغير `int`

• `double* d;` لازم يخزن عنوان متغير `double`

2. لو حاولت تخزن عنوان `double` في `int*` بصير خطأ:

```
double x = 5;
```

```
int* p = &x; // ❌ Error
```

مثال تطبيقي:

```
int x = 10, y = 20;
```

```
int* p1, *p2;
```

```
p1 = &x;
```

```
p2 = &y;
```

p1 يخزن عنوان x

p2 يخزن عنوان y

```
*p1 = 30;
```

```
*p2 = *p1;
```

عدلنا قيمة x لـ 30 عن طريق p1

عدلنا y وخرنا فيه نفس قيمة x يعني 30

✓ النتيجة النهائية:

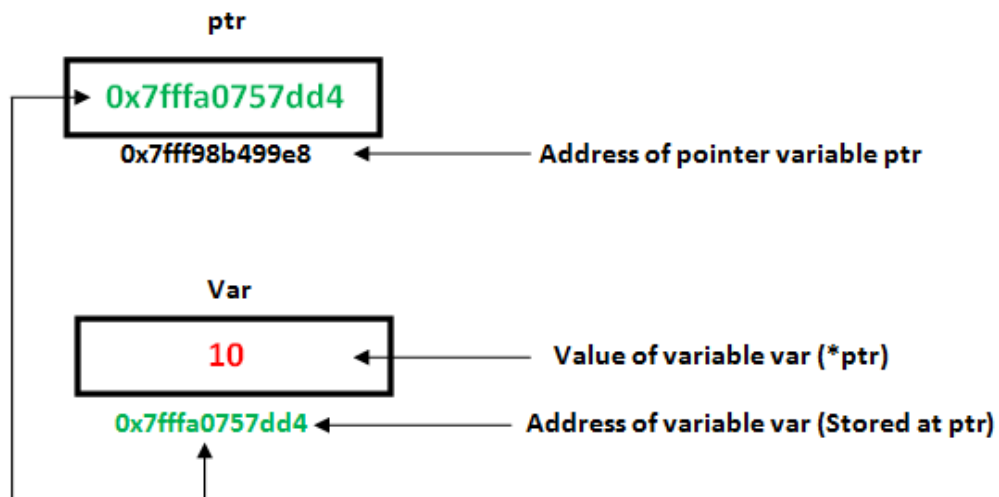
- x = 30

- y = 30

ملخص سريع:

المصطلح	المعنى
int* p;	بوينتر يخزن عنوان متغير int

<code>p = &x;</code>	خزّن عنوان المتغير x في p
<code>*p</code>	القيمة اللي داخل x
<code>*p = 7;</code>	تعديل قيمة x عن طريق البوينتر



Working of Pointer in C++

```
int value = 23 ;
```

value

23

#4590

```
int* ptr=&value;  
*ptr=49;
```

value

49

#4590



Dynamic Memory Allocation

(الحجز الديناميكي للذاكرة)

أولاً: شو يعني “ذاكرة ديناميكية”؟

بشكل طبيعي، لما تكتب متغير بالكود زي هيك:

```
int x = 5;
```

المتغير x بيتتم حجز مكانه في الذاكرة وقت الترجمة (Compile Time)

والمكان هذا اله اسم (x) وبيخزن فيه القيمة (5)

لكن أحياناً، في حالات خاصة، ما بنعرف كم متغير بدنا أو متى

بنحتاج نحجز مساحة إلا أثناء تشغيل البرنامج نفسه Run

Time

مثال:

أنت مش عارف إذا رح تحتاج تخزن رقم واحد أو عشر أرقام إلا لما المستخدم يدخل البيانات!

🔥 هون بنستخدم Dynamic Memory Allocation

يعني:

نحجز خلية في الذاكرة وقت تشغيل البرنامج، مش وقت كتابة الكود، ونخزن فيها أي قيمة بدنا.

✅ ثانياً: كيف نحجز ذاكرة ديناميكيًا؟

1. منعرف بوينتر عادي.

2. منستخدم الكلمة المفتاحية new لحجز خلية بالذاكرة.

3. منخزن العنوان الناتج داخل البوينتر.

4. منخزن قيمة داخل الخلية باستخدام *.

مثال:

<pre>int* p = new int; // *p = 7; // cout << *p; // delete p;</pre>	<p>حجز خلية ديناميكية من نوع int</p> <p>خزنا فيها القيمة 7</p> <p>طباعة القيمة 7</p> <p>بتحذف الخلية اللي البوينتر p بيشار عليها</p>
---	--

هذا الكود بيحجز مكان جديد بالذاكرة، وبيخزن فيه 7، وبعدين بطبعها.

ملاحظة مهمة جدًا:

المكان الذي حجزته باستخدام new، ما يَسمح لحالته بعد انتهاء البرنامج أو الوظيفة.

عشان هيك، لازم أنت تحذف الذاكرة يدويًا باستخدام delete

كود بسيط:

```
#include <iostream>

using namespace std;

int main() {
    int* p = new int; // حجز خلية جديدة
    *p = 10;           // خزن فيها قيمة

    cout << "Value: " << *p << endl;

    delete p;         // حذف الخلية

    return 0;
}
```

ملخص سريع:

الكود	المعنى
<code>int* p;</code>	بوينتر من نوع <code>int</code>
<code>p = new int;</code>	حجز خلية جديدة ديناميكيًا
<code>*p = 5;</code>	تخزين قيمة داخل الخلية
<code>delete p;</code>	حذف الخلية بعد ما نخلص منها

