

# C++ in values and Types Objects,

Ghazy Yousef

٣١ يوليو ٢٠٢٥

## المحتويات

٢	١	ازاي ناخذ input؟
٦	٢	variables ال
٨	٣	input مع ال types
٩	٤	العمليات وال Operators
١٢	٥	assignment وال initialization
١٣	١٠.٥	مثال: words repeated detect
١٦	٢٠.٥	ال assignment composite
١٦	٣٠.٥	مثال words repeated بعد التحسين
١٧	٦	الاسامي names
١٩	٧	types وال objects
٢٢	٨	ال safety type
٢٣	٩	ال conversions
٢٦	١٠	review
٢٨	١١	exercises
المحاضره دي هتشرح انواع البيانات في ال ++C ال Types Data وازاي ناخذ data من ال standard input stream او ال keyboard ونفهم يعني ايه data و data types و objects و values و variables		

## ١ ازاي ناخذ input؟

لحد الان كل الي برنامج worldhello بتاعنا بيعمله انه يطبع "Hello!" "World" بس، مش بيعمل اي حاجه ثاني، مش مش يقرأ اي حاجه، مش بياخذ input من اليوزر، زي ما انت شايف كدا الموضوع ممل شويه، البرامج الحقيقيه معظم الوقت بتعمل حاجه مختلفه علي حسب ال input الي بتديهولها، بدل من انها مجرد بتعمل نفس الشئ مرارا وتكرارا كل مره تشغل البرنامج فيها

علشان ناخذ داتا من اليوزر ونخزننا علشان نستخدمها بعد كدا في البرنامج، عايزين حاجه زي database نخزن فيها الداتا دي، وال database دي هي ال memory بتاعه الكمبيوتر بتاعك.

لما بنيجي ناخذ input من اليوزر لازم الاول نحجزله مكان في ال memory علشان نخط فيها ال data الي اليوزر هيدخلها، والمكان دا بيتقال عليه object.

ال object هو مكان في ال memory عندك محجوز علشان يتخط فيه نوع معين من البيانات، ولما بندي لل object دا اسم بيتقال عليه variable. علي سبيل المثال ال strings character بتتخط في variable، string وال integers بتتخط في int، variable تقدر تتخيل ال variable دا صندوق في ال memory عندك ليه اسم وحجم ونوع معين نقدر نخط فيه داتا من نفس نوع الصندوق، خلينا ناخذ مثال:

```
#include <iostream>

int main() {
    std::cout << "Please type your age:\n> ";
    int age;
    std::cin >> age;
    std::cout << "your age is " << age << '\n';
    return 0;
}
```

ال #include وال main احنا كدا بالفعل عرفناهم من المحاضرات الي فاتت، وبما اننا كدا كدا هحتاج نعمل <include> #<iostream> في تقريبا كل البرامج في خلال الكام محاضره الجايين، فحنا معدناش هنكتبها افتراضا بأنك كدا كدا عارف انها اساسيه، وبرضوا احنا ساعات كتير هنكتب كود مش هيشغل غير لو حطيته جوا ال main زي مثلا:

```
std::cout << "hello\n";
```

فانا برضو بفترض اني مش محتاج اقولك تحط الكود دا في ال main علشان يشتغل ولو حطيته في اي حته ثانيه مش هيشغل، وكدا كدا لو متعرفش ال compiler هيعرفك.

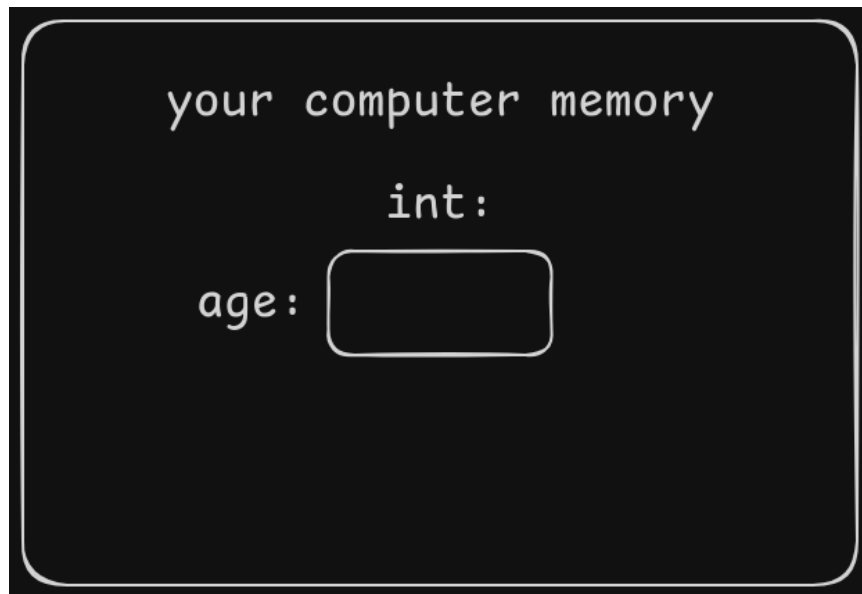
اول سطر عندنا في ال main يطبع رساله "Please type your age:" ودي بيتقال عليها prompt ويبقي شكلها كدا:

```
Please type your age:
> |
```

وزي ما انت شايف بيقولك تعمل ايه، بعدها ينزل سطر ويكتب ">" وبعدها يسبيك تدخل input السطر اللي بعده:

```
int age;
```

دا بيعرف variable من النوع int اسمه age، بمعنى انه يحجز مكان في ال memory عندك (object) ويبدله اسم (age) ويخليه جاهز انه يتخزن فيه قيم من نوع int (ارقام صحيحة)



السطر اللي بعده هتلاحظ انه شبه السطر اللي بيطلع بس مع بعض الاختلافات:

```
std::cin >> age;
```

السطر دا بياخد input من ال standard input stream (ال keyboard في الترمينال) عن طريق object ال cin وده object من ال istream او ال stream input ويحطه في المتغير age عن طريق ال operator << واللي اسمه **from get**

يعني نقدر نقول ان السطر دا بيتقرأ: `cin using stream input standard the from input get` variable. age the in it put and library standard the in is which وبعد تنفيذ السطر دا، بفرض مثلاً اننا دخلنا له 20 كـ input، المنظر هيبقى كذا:



السطر بقا الي بعده:

```
std::cout << "your age is " << age << '\n';
```

دا سطر طباعه عادي، بس هتلاحظ كذا حاجه، اولاً اننا نقدر نطبع كذا حاجه مع بعض (ورا بعض) عادي باستخدام ال operator >> متغير مشا كل. لو كنا بنحب التكرار والكاتبه الكثير كان ممكن نعمله بالشكل دا:

```
std::cout << "your age is ";
std::cout << age;
std::cout << '\n';
```

والاثنين كانوا هيطلعوا نفس ال output عادي، بس احنا دايماً في البرمجه بنحاول نقلل حجم الكود علي قد ما نقدر، وبنحاول نقلل التكرار دايماً لأن التكرار وزياده حجم الكود يعني احتماليه اكبر لظهور ال errors.

تاني حاجه هتلاحظ اننا نطبع القيم اللي ييخزنها ال variable عادي، بس مش بنستخدم بقا " حوالين اسم ال variable، لأننا لو عملنا كذا هيطبع كلمه age حرفياً، واحنا مش عايزين كذا احنا قصدنا علي المتغير age.

ثالث حاجه هتلاحظ اننا نستخدم ' ' حوالين ال n\ بدل " " وده لأن زي ما قلنا قبل كذا n\ حرف واحد مش string، وفي ال ++C ومعظم لغات البرمجه، الأفضل انك تستخدم مع الحروف ' ومع ال string، "، مثلاً 'c' دا حرف، و "hello" دا string. فأحنا مثلاً لو نفذنا البرنامج دا هيحصل المنظر الآتي:

```
~/code $ ./main
Please type your age:
> 20
your age is 20
~/code $
```

هتلاحظ اننا بنشغل البرنامج، بيديلنا ال prompt الحلو بتاعتنا، بنكتب 20 وبنضغط enter والي ساعات بيتقال عليها، return وبيقولي 20 is age your. خلتنا نبص بصفه علي برنامج ثاني برضو بيعمل حاجه مشابه:

```
#include <iostream>
#include <string>

int main() {
    std::cout << "Please type your name:\n> ";
    std::string name;
    std::cin >> name;
    std::cout << "Hello, " << name << "!\n";
    return 0;
}
```

اول حاجه هتلاحظ اننا علشان نقدر نتعامل مع ال strings هحتاج نستخدم المكتبة او ال header ال الي اسمه string ودا برضو header تبع ال C++ standard library وبرضو علشان نستخدم ال strings جوا الكود لازم نخط قبلهم std::: غير كذا البرنامج تقريبا زيه زي الي فات بظبط، وتنفيذه هيبقي عامل كذا:

```
~/code $ ./main
Please type your name:
> yousef
Hello, yousef!
~/code $
```

## ٢ ال variables

ببساطة، ما نقدرش نعمل أي حاجة مفيدة على الكمبيوتر من غير ما نخزن بيانات في ال memory، زي ما عملنا مع ال input statement في المثال اللي فوق. "الأماكن" اللي بنخزن فيها البيانات دي بنسميها كائنات (objects). علشان نوصل ونستخدم object لازم يكون له اسم. ال object اللي له اسم بنسميه متغير (variable) ويكون له نوع محدد (type) زي int أو string. النوع ده هو اللي يحدد إيه اللي ممكن نخطه جوه المتغير (زي مثلاً: 123 ممكن يخط في int، و"Hello"، "World!" ممكن يخط في string) وكان يحدد إيه العمليات اللي ممكن نعملها عليه (زي إتنا نضرب أعداد صحيحة أو integers باستخدام ال operator \*، أو نربط بين نصوص --نعمل concatenation باستخدام ال operator +). القيم اللي بنخطها جوه المتغيرات دي بنسميها قيم (values). الجملة اللي بتعرف variable جديد اسمها (مش مفاجأة) تعريف (definition)، وغالباً (ومن الأفضل) يكون فيها قيمة ابتدائية (initial value). (مثلاً:

```
std::string name = "yousef";
int number_of_steps = 33;
```

القيمة اللي بتيجي بعد علامه اليساوي = بيتقال عليها initializer او قيمه ابتدائية. تقدر تخيل ال variables دي لك objects في ال memory بالشكل دا:



ولاحظ اننا مينفعش نخط نوع غلط من البيانات جوا variable، لازم نخط الـ data بشكل يكون according to the variable type، مثلاً:

```
int age = "yousef";    // error: "yousef" is not an integer
std::string name = 20; // error: 20 is not a string
```

الـ compiler بياخد باله من نوع كل variable، ويبدأ كد إنك بتستخدمه بالطريقة اللي تناسب نوعه، النوع اللي انت اختارته وانت بتـ define او بتعرف الـ variable.

عندنا في الـ C++ في عدد كبير جداً من الـ primitive types بتيجي مع اللغة، بس في الأول كدا خيلنا نعرفك علي 5 بس منهم:

```
int number_of_steps = 33;    // int for integer numbers
double flying_time = 3.5;    // double for floating-point numbers
char decimal_point = '.';    // char for individual characters
std::string name = "yousef"; // string for character strings
bool tap_on = true;          // bool for logical variables
```

السبب في اسم الـ double دا تاريخي شويه، الـ double هنا معناها floating-point double، point، والـ floating-point هي طريقة الكمبيوتر في التعبير عن الأعداد الحقيقية. ولاحظ ان كل نوع من دول له الستايل المميز بتاعه في طريقه كُتابته واللي بيتقال عليه literal:

```
33          // int: an integer
3.5         // double: a floating-point number
'.'         // char: an individual character enclosed in single quotes
"yousef"    // string: a sequence of characters delimited by double quotes
true        // bool: either true or false
```

يعني إيه؟ يعني تسلسل من الأرقام (زي 1234 أو 2 أو 976) معناه عدد صحيح (integer)، وحرف واحد جوا quotes single (زي '1' أو '@' أو 'x' أو '\n') معناه حرف، وتسلسل أرقام فيه نقطة عشرية (زي 234.1 أو 12.0 أو 0.92) معناه رقم عشري (floating-point)، وتسلسل من الحروف جواه quotes double (زي "1234" أو "Hello!" أو "yousef") معناه نص (string).

### ٣ ال input مع ال types

عملية انك تاخذ input باستخدام ال operator from get << بتبقي حساسه لل type بتاع القيمه اللي داخله ونوع ال variable الي انت بتدخل فيه، وبتقرأ بنائاً علي ال type بتاعهم، بص كذا المثال دا مثلاً:

```
int main() {
    // read name and age
    std::cout << "Please enter your first name and age\n> ";
    std::string first_name;           // string variable
    int age = 44;                     // integer variable with arbitrary value
    std::cin >> first_name >> age;    // read a string followed by an integer
    std::cout << "Hello, " << first_name << " (age " << age << ")\n";
}
```

زي ما انت شايف نقدر اننا نقرأ كذا قيمه مره واحده زي ما نقدر نطبع كذا قيمه مره واحده. تعالي نجرب نلعب شويه مع البرنامج دا ونشوف هيتصرف ازاي في حالات مختلفه:

```
~/code $ ./main
Please enter your first name and age
> yousef 20
Hello, yousef (age 20)
~/code $ ./main
Please enter your first name and age
> 20 yousef
Hello, 20 (age 0)
~/code $
```

زي ما انت شايف اول مره شغلناه، لما بنكتب "yousef 20" ال operator << بيقرأ "yousef" في first\_name بعد كذا 20 في age، طب ليه ميقرأش "yousef 20" كلها في first\_name؟ علشان قرائه ال strings بتنتهي بال whitespace، اللي هي ال space أو newline أو tab، انما غير كذا ال whitespaces في الطبيعي بيتم تجاهلها من قبل <<. جرب مثلاً تدخله ال input دا: " 20 yousef"، هتلاقيه بيقولك "hello, yousef (20)" عادي منغير whitespaces بس لو جيت تكتب 20 وبعدها yousef بالشكل اللي انت شفته لما شغلنا البرنامج مره ثانيه هتلاقيه قالك "hello, 20 (age 0)"، ليه؟ علشان هو هيقراً 20 في first\_name عادي، لأن في الأول وفي



الآخر "20" عبارة عن سلسلة من الحروف عادي ينفع تتقرأ في، string انما "yousef" مينفعش تتقرأ في int فمش هيعرف يقرأها في، age فييحط 0 ويديشيل ال 44 الي كانت موجوده.  
زي ما انت شفت، عملية القرائه لل strings بتنتهي بال whitespace بمعنى انه مش هيعرف يقرأ غير كله واحده، بس افرض احنا عليز نقرأ اكتر من كلمه؟ في طرق كتير تقدر تعمل بيها كدا، مثلاً ممكن نقرأ اسم من كلمتين بالشكل دا:

```
int main() {
    std::cout << "Please enter your first and second names\n> ";
    std::string first;
    std::string second;
    std::cin >> first >> second;    // read two strings
    std::cout << "Hello, " << first << " " << second << '\n';
}
```

ببساطه بنستخدم << مرتين لكل اسم، ولو عليز نطبع الاسامي دي لازم نخط مسافه بينهم.  
لاحظ ان مفيش initializer لل two variables بتوعنا first و second مع اننا قلنا ان المفروض دايماً نخط initializers، وده لأن default by ال strings بيتعملها initialization ل empty string، بمعنى ان:

```
std::string first;           // initialized to "" or empty string
std::string second = "";    // initialized to "" or empty string
// so basically both are the same
```

---

جرب دي: جرب تكتب برنامج ال name وال age بتاعنا دا، وعدله بحيث انه يطبع العمر بالشهور، يعني لو شخص دخل عمره 20 سنه يقوله انه عمره 240 شهر، فانت كدا هتحتاج تضرب العمر في 12، واستخدم double بدل int علشان الأطفال الي ممكن بكل نغري باقي عمرهم 6 سنين ونص.

---

## ٤ العمليات وال Operators

بالأضافه للقيم الي ينفع نخطها في ال variable، نوع ال variable ايضاً يحدد العمليات الي نقدر نعملها عليه ومعناها ايه، علي سبيل المثال:

```
int age = -1;
std::cin >> age;           // >> reads an integer into age
std::string name;
```

```
std::cin >> name;           // >> reads a string into name
int a2 = age + 2;           // + adds integers
std::string n2 = name + " Jr. "; // + concatenates strings
int a3 = age - 2;           // - subtracts integers
std::string n3 = name - " Jr. "; // error: - isn't defined for strings
```

لما نقول error فاحنا قصدنا ان ال compiler مش هيرض ي compile البرنامج دا وهيطلعلك error ان ال operator - مش متعرف لل strings، ال compiler عارف كويس ايه العمليات اللي تنفع علي المتغيرات من النوع الفلاني دي مثلا بعض ال operators لبعض الأنواع المشهوره:

operation	bool	char	int	double	string
assignment	=	=	=	=	=
addition			+	+	
concatenation					+
subtraction			-	-	
multiplication			*	*	
division			/	/	
remainder (modulo)			%		

operation	bool	char	int	double	string
increment by 1			++	++	
decrement by 1			--	--	
increment by n			+= n	+= n	
add to end					+=
decrement by n			-= n	-= n	
multiply and assign			*=	*=	
divide and assign			/=	/=	
remainder and assign			%=		
read from s into x	s >> x	s >> x	s >> x	s >> x	s >> x
write x to s	s << x	s << x	s << x	s << x	s << x
equals	==	==	==	==	==
not equal	!=	!=	!=	!=	!=
greater than	>	>	>	>	>
greater than or equal	>=	>=	>=	>=	>=
less than	<	<	<	<	<
less than or equal	<=	<=	<=	<=	<=

لو مكان ال operation فاضي دا معناه ان ال type مش بي support ال operation دي بشكل مباشر.  
 احنا هنشرح ال operations دي واكثر علي مدار الكورس، ولكن الهدف هنا من اني اوريك الجدول دا هو انك تعرف ان في operations كتير ب operators خاصه بيها وغالباً بيشتروا في المعني وسط كتير من الأنواع.  
 تعالي مثلاً نشوف بعض ال operations الي ممكن تتعمل علي ال floating-point numbers من النوع: double

```
#include <cmath>

int main() {
    // simple program to exercise operators
    std::cout << "Please enter a floating-point value: ";
    double n = 0;
    std::cin >> n;
    std::cout << "n == " << n
              << "\nn+1 == " << n+1
              << "\nthree times n == " << 3*n
              << "\ntwice n == " << n+n
              << "\nn squared == " << n*n
              << "\nhalf of n == " << n/2
              << "\nsquare root of n == " << std::sqrt(n)
              << '\n';
}
```

طبعا، العمليات الحسابية العادية ليها نفس الشكل والمعنى الي اتعلمناه في المدرسة. الاستثناء الوحيد هو ان علامة المساواة بتكون == مش = ، لأن = في البرمجة معناها "assignment" او انك تعين قيمه للمتغير او ال variable مش مقارنة. يعني بنستخدمها عشان نخط قيمة في متغير.  
 طبيعي برضو ان مش كل حاجة ممكن نعملها على الأرقام (زي الجذر التربيعي مثلاً) تكون متاحة كـ "عملية مباشرة" باستخدام operator. عشان كده في عمليات او operations بنستخدم فيها functions ليها أسماء. في الحالة دي، لو عايزين نجيب الجذر التربيعي لعدد n، بنستخدم function اسمها sqrt من ال standard C++ library، عشان كذا اضطررنا نعمل include ل file header اسمه cmath وكتبنا قبلها std:: ، ، وبنكتبها كده: sqrt(n) ، ودي طريقة معروفة في الرياضيات.

جرب دي اكتب البرنامج الصغير ده وخليه يشتغل. بعد كده عدله عشان يقرأ عدد صحيح (int) بدل ما يقرأ عدد عشري (double). كان جرب عليه شوية عمليات أو operations ثانية، زي عملية باقي القسمة او ال % modulo. خد بالك إن لما بنشتغل بـ int، القسمة / بتكون قسمة عددية صحيحة، يعني

النتيجة من غير كسور، و % معناها الباقي بعد القسمة.  
يعني مثلاً: 5 / 2 نتيجتها 2 (مش 5.2 ولا 3) و 5 % 2 نتيجتها 1

ال strings ليهم عدد اقل من ال operations بس زي ما هنشوف بعد كذا ان ليهم كثير من ال operations علي شكل functions. بس ال operations اللي بتعمل عليهم باستخدام operator بتبقي نوعاً ما سهله ومنطقيه، زي كذا مثلاً:

```
int main() {
    // read first and second name
    std::cout << "Please enter your first and second names\n";
    std::string first;
    std::string second;
    std::cin >> first >> second;           // read two strings
    std::string name = first + ' ' + second; // concatenate strings
    std::cout << "Hello, " << name << '\n';
}
```

في حالة ال strings ال + معناها concatenation، يعني لو s1 و s2 دول two strings، فا s1 + s2 معناها ان الحروف بتاعه s2 هتكل بعد الحروف بتاعه s1

## ٥ ال assignment وال initialization

واحد من اهم ال operators واكثرهم اثاره للأهتمام هو ال assignment operator = واللي بيعمله انه يحط قيمه جديده في ال variable

int a = 3; // a starts out with the value 3	a: <input type="text" value="3"/>
a = 4; // a gets the value 4 (becomes 4)	a: <input type="text" value="4"/>
int b = a; // b starts out with a copy of a's value ( which is 4)	a: <input type="text" value="4"/> b: <input type="text" value="4"/>
b = a + 5; // b gets the value a+5 (which is 11)	a: <input type="text" value="4"/> b: <input type="text" value="11"/>
a = a + 7; // a gets the value a+7 (which is 11)	a: <input type="text" value="11"/> b: <input type="text" value="11"/>

ركز علي اخر assignment. أولاً، واضح جداً ان هنا علامه ال = تساوي بمعنى المقارنه المتعارف عليه، لأن a = a + 7 دي مستحيله رياضياً، هنا علامه = معناها اتنا عايزين نخط قيمه جديده في a

والقيمة دي هتساوي  $a + 7$  و  $a$  المفروض انها كانت ب 4 و  $4 + 7$  ب 11، فأكنا بنقوله اننا عايزين نغير قيمة  $a$  ل 11.  
ونقدر برضو نعمل نفس الحوار بال strings:

```
std::string a = "alpha"; // a starts out with the value "alpha"
a = "beta"; // a gets the value "beta" (becomes "beta")
std::string b = a; // b starts out with a copy of a's value ( which is "beta")
b = a + "gama"; // b gets the value a+"gama" (which is "betagama")
a = a + "delta"; // a gets the value a+"delta" (which is "betadelta")
```

لاحظ اننا بنستخدم المصطلحين starts out with و gets علشان نفرق بين عمليتين متشابهين نوعا ما بس منطقيا مختلفين:  
احنا بنستخدم "يبدأ بـ" (starts) out (with) و "ياخد" أو "يحتطه" (gets) علشان نفرق بين عمليتين شبه بعض، لكن من الناحية المنطقية مختلفين:  
• ال Initialization: يعني بندي للمتغير لل variable قيمة ليه لما علطول واحنا بنعرفه.  
• ال Assignment: يعني بنغير قيمة المتغير ونديه قيمة جديدة بعد ما اتعرف.

من الناحية المنطقية، ال initialization وال assignment مختلفين. مبدئياً، ال initialization يحصل وال variable لسه فاضي. أما ال assignment فهو لازم (من حيث المبدأ) يشيل ال value القديمة من ال variable قبل ما يحط الجديدة.  
تقدر تخيل ال variable كأنه علبة صغيرة، وال value اللي بتتحط فيه كأنها عملة معدنية. قبل ال initialization، العلبة فاضية، لكن بعد ما بنعمل ال initialization، العلبة دايما بيكون فيها عمله. فلما نتيجي تحط عملة جديدة (يعني نعمل ال assignment) لازم الأول تشيل العملة القديمة — أو بمعنى ثاني "نتخلص من القيمة القديمة"، ويمكن حتي نستخدمها كمرجع لل value الجديد زي ما شفنا في مثال  $a + 7 =$  طبعا في ال memory الموضوع مش بالتبسيط المخل دا، بس دي طريقة كويسة تساعدك تخيل اللي يحصل.

## ١.٥ مثال: words repeated detect

لو تلاحظ هتلاقي ال assignment بتبقي مفيدة اكرت حاجه لما نبقى عايزين نغير قيمة نفس ال variable كذا مره، تعالي مثلا نبص علي البرنامج دا اللي بي detect الكلمات المتكرره:

```

int main() {
    std::string previous;
    std::string current;
    while (std::cin >> current) {
        if (current == previous) {
            std::cout << "word: " << current << " repeated\n";
        }
        previous = current;
    }
}

```

تعالى نبص على البرنامج دا سطر سطر ونحاول نفهم هو بيعمل ايه.

```

std::string previous;
std::string current;

```

اول حاجه هتلاحظ اننا زي ما قلنا مفيش initializer لل strings علشان هما default by بيتعملهم initialization default ل string empty او ""

```

while (std::cin >> current) {
    // statements here
}

```

الجملة دي بنسميها while-statement، مثيره للاهتمام في حد ذاتها، وهنشرحها أكثر المحاضره الجايه علي طول.

ال while معناها إن التعليمات اللي بعد std::cin >> current (الي عاده بتكون جوا ال {} ) هتفضل تتكرر طالما عملية ال input بتاعه cin >> current بتنجح، و std::cin >> current هتنجح طالما فيه حروف لسه ممكن تتقري من ال input. standard فهو هيفضل ينفذ std::cin >> current ولو العملية دي نجحت هينفذ الي جوا {}

افتكر إن في حالة string، ال << بتقرا كلمات مفصولة بمسافات. بتنتهي ال loop دي عن طريق إنك تدي للبرنامج character end-of-input (الي غالباً بيتقال عليه file). of end  
على Windows، ده بيكون بالضغط على Ctrl+Z وبعدها Enter، أما على Linux، فيكون بالضغط على Ctrl+D.

```

if (current == previous) {
    std::cout << "word: " << current << " repeated\n";
}

```

هنا احنا بختصار بنقارن القيمة بتاعه current والي هي الكلمه اللي اليوزر لسه مدخلها، بالقيمة بتاعه previous والي هي اخر كلمه اليوزر دخلها، ولو هم نفس الكلمه، بنطبع دا

```
previous = current;
```

بعد كذا قبل ما `std::cin >> current` تنفذ ثاني ونرجع ناخذ `input` ثاني من اليوزر وال `fi-statement` تنفذ ثاني، لازم نخزن قيمه ال `current` في ال `previous` بحيث ان اخر كلمه اليوزر دخلها تبقي هي الكلمه اللي فاتت ونستعد اننا ناخذ كلمه جديده.

طريقة من طرق فهم سير البرنامج (program) (flow إنك "تلعب دور الكمبيوتر"، يعني تمثلي ورا البرنامج سطر بسطر، وتعمل اللي مكتوب فيه خطوة بخطوة. ارسم مربعات على ورقة واكتب فيها ال `values` بتاعه المتغيرات. وغير ال `values` دي زي ما البرنامج بيغير فيهم.

---

جرب دي  
نفذ البرنامج ده بنفسك باستخدام ورقة وقلم. استخدم ال `input`:

```
The  
cat  
cat  
jumped.
```

حتى المبرمجين المحترفين ساعات يستخدموا الطريقة دي علشان يتخيلوا اللي يحصل في جزء صغير من الكود، خصوصاً لو مش واضح ليهم بالضبط هو بيعمل إيه.

---

جرب دي  
خلي برنامج `"repeated word detection"` يشتغل. جربه بالجملة دي: `he "he laughed she "She`  
`good" good very very look not did did he what because he!`  
خد الجملة `paste copy` او اكتبها مره واحده في ال `terminal` ومتدخلهاش كلمه كلمه

• كام كلمه مكررة لقيتها؟

• ليه؟

• يعني إيه "كلمه" هنا؟

• ويعني إيه "كلمه مكررة"؟

(يعني مثلاً، هل `"She"` `she` تعتبر تكرار؟)

---

## ٢.٥ ال assignment composite

في البرمجة، انك تغير قيمة ال variable بنائاً على قيمته القديمة زي مثلاً:  $a = a + 7$  دي حاجة بتحصل كتير جداً، وال ++C بتديك syntax خاص علشان تعمل كدا:

```
a += 7; // means a = a + 7
b -= 9; // means b = b - 9
c *= 2; // means c = c * 2
```

زي ما انت شايف بدل ما نكتب  $a = a + 7$  ممكن نكتب  $a += 7$  ومعناها زود 7 على قيمة a الحالية  
في العموم لو op دا operator binary فا  $var = var$  بتعادل  $var = var$  expression op  
اهم حاجة دلوقتي هي ال operators دي: += و -= و \*= و /= و %  
في حاله انك عايز تزود القيمة بتاعه المتغير بواحد بظبط، ونظراً لأن ده هيجحصل كتير، فال ++C بتسمحلك انك تكتب حاجة زي كدا:  $var++$  ودي تعتبر زيها زي  $var = var + 1$   
1

## ٣.٥ مثال words repeated بعد التحسين

في مثال ال words repeated اللي فوق احنا ممكن نضيف تعديل بسيط يخلينا نعرف الكلمه رقم كام بظبط اللي اتكررت باستخدام ال assignment: composite

```
int main() {
    int number_of_words = 0;
    std::string previous; // previous word; initialized to ""
    std::string current;
    while (std::cin >> current) {
        ++number_of_words; // increase word count each time after reading a word
        if (previous == current)
            std::cout << "word number " << number_of_words << " repeated: " << current;
        previous = current;
    }
}
```

اول حاجة بنبدأ مع number\_of\_words ب 0، المتغير دا هيكون زي العداد او ال counter بتاعنا، كل مره هنقرأ فيها كلمه جديده، هنزود المتغير دا بواحد  $number\_of\_words++$   
لاحظ قد إيه البرنامج ده شبه اللي البرنامج الي فات. واضح إننا خدنا نفس البرنامج وعدلناه شوية علشان يخدم الهدف الجديد بتاعنا. ودي طريقه شائعة جداً: لما نكون عايزين نحل مشكلة، بندور على مشكلة



شبهها ونستخدم الحل بتاعها مع شوية تعديلات مناسبة. ما تبدأش من الصفر إلا لو مضطر. استخدام نسخة سابقة من برنامج كأساس للتعديل بيوفر وقت كثير، وكان بنستفيد من المجهود اللي اتبذل في النسخة الأصلية.

## ٦ الأسماء names

إحنا بنسمي الobjects علشان نقدر نفتكرها ونرجع لها من أجزاء ثانية في البرنامج. طب إيه اللي ينفع يكون اسم في ++C، في ++C، الاسم لازم يبدأ بحرف، ويمكن يحتوي على حروف وأرقام و"أندرسكور" (-) بس. مثلاً:

```
x
number_of_elements
Fourier_transform
z2
Polygon
```

دي كلها تنفع اسماء عادي، بس اللي جاي دا مينفعش:

```
2x           // a name must start with a letter
time@to@market // @ is not a letter, digit, or underscore
Start menu   // space is not a letter, digit, or underscore
```

ومتنفعش هنا بمعنى ان الcompiler مش هيرضي يعرفهم كأسماء وهيطعلك error. ولا حظ برضو ان الnames بتبقي case sensitive، يعني انك تقدر تعمل variable اسمه one و variable ثاني اسمه One عادي، بس دي حاجه لا ينصح بيها، علي الرغم من انها مش هتلتخط الcompiler بس بسهولة هتلتخط المبرمج. في مجموعه من الnames في ال++C بيتقال عليها keywords، ودي بتبقي names اللغه بتستخدمها زي مثلاً fi, while, int, double وهكذا، لو جربت تستخدمهم هيطعلك error:

```
int if = 7; // error: if is a keyword
```

بس تقدر تستخدم الnames بتاعه الحاجات اللي في الlibrary standard عادي، زي كذا مثلاً:

```
int string = 5; // compiles, but will lead to trouble
double cout = 2.4; // compiles, but will lead to trouble
```

السبب في ده انك مش بتقوله `std::string` او `std::cout` بس مع ذلك لا ننصح بأنك تعمل دا علشان استخدامك لأسامي منتشرة زي دي غالباً هيؤدي لـ `errors` في باقي الكود. لما تيجي تختار أسماء للمتغيرات أو الدوال أو الأنواع، (`types`) اختار أسماء ليها معنى، يعني أسماء تساعد اللي بيقرأ الكود يفهمه. حتى إنت نفسك هتواجه صعوبة في فهم برنامجك لو كنت مليته بمتغيرات أسماءها سهلة في الكتابة بس ملهاش معنى، زي: `x1, x2, s3` و `p7`. الاختصارات والحروف المقطعة (`acronyms`) ممكن تلخبط الناس، لحاول تقلل منها على قد ما تقدر. ممكن تكون كانت واضحة لينا وقت ما كتبناها، بس غالباً إنت هتواجه صعوبة في فهم واحدة منهم على الأقل، وكان هتصعب عليك انك تلاقي الـ `errors` في الكود بتاعك برضو حاول متكتبش اسامي طويله اوي، بتخلي الكود اصعب في القرائه، يعني مثلاً الأسامي دي كويسه:

```
partial_sum
element_count
stable_partition
```

انما الأسامي دي غالباً طويله اوي:

```
the_number_of_elements
remaining_free_slots_in_symbol_tab
```

الـ "ستايل" اللي إحنا ماشيين عليه (يعني الطريقة اللي بنكتب بها الكود) هو إنا بنستخدم `underscore` (-) للفصل بين الكلمات في الاسم، زي `elementCount`، بدل الطرق الثانية زي `elementCount` أو `ElementCount`. وعمرنا ما بنستخدم أسماء كلها حروف كابيتال زي `LETTERSCAPITALALL`، علشان ده تقليدياً يبقى مخصص للماكروز (هنعرف يعني ايه ماكروز بعدين)، ودي حاجة إحنا بنتجنب نستخدمها. بعد كذا هتلاقيك بدأت تعرف انواع او `types` جديده خاصه بيك، إحنا بنبدأ أسماء الأنواع او الـ `types` اللي بنعرفها بحرف كابيتال، زي `Square` و `Graph`. بس لغة `C++` و `library standard` بتاعتها ما بتستخدمش الطريقة دي، فبنلاقي مثلاً `int` مش `Int` و `string` مش `String`. علشان كده، القاعدة اللي إحنا بنمشي عليها بتقلل اللخطة ما بين الأنواع اللي إحنا بنعملها والأنواع اللي موجودة في `C++` أصلاً.

وبمناسبه الأنواع اللي موجوده في الـ `C++` اصلاً، ساعات كتير الـ `implementation` يستخدم اسامي بتبدأ بـ `_` حاول برضو متعملش اسامي بتبدأ بـ `_` علشان متبشش تلاقي الأسامي بتاعتك بتلخبط مع حاجات في الـ `implementation`.

## ٧ ال types وال objects

مفهوم ال types شيء رئيسي في ++C ومعظم لغات البرمجة الثانية، تعالي نبص بصره متعمقه وتيكنيكال اكثر علي حوار ال types دا:

- ال type يعرف مجموعه من ال values الي ينفع تتخط في object من ال type دا
- ال type يعرف مجموعه من ال operations الي ينفع تتعمل علي object من ال type دا
- ال object هو مكان في ال memory يخزن value من type معين
- ال values هي مجموعه من ال bits في ال memory الي بتقرأ بنائاً علي ال type بتاع ال object في ال memory
- ال variable هو object ليه اسم
- ال declaration هي statement بتدي name و type لل object، تقدر تقول انها بتعرف ال compiler ان ال variable دا موجود
- ال definition هي statement بتدي name و type لل object، زي ما انت شايف ال declaration عادي، بس كان بتجزله مكان في ال memory
- ال definition ممكن يدي ال variable قيمه مبدئية (initial value) في عملية تسمي بال initialization ومعظم الوقت بيقتي دا شيء مستحب انه يحصل
- مفهوم ال declaration والفرق بينه وبين ال definition ممكن مبيقاش واضح دلوقتي بس هنفهمه اكثر بعدين.
- زي ما قلنا احنا ممكن (بشكل غير رسمي) نفكري في ال object علي إنه زي علبة (box) بنخط فيها قيم من نوع معين. يعني مثلاً، علبة من نوع int ممكن تشيل أرقام صحيحة زي 7، 42، و-399. وعلبة من نوع string ممكن تشيل سلاسل من الحروف، زي: "yousef" و "operators: +-\*/%"، و "fun". is programming
- ممكن نتخيل ده كده بشكل مرسوم بالطريقة دي:

<code>int a = 7;</code>	7
<code>int b = 9;</code>	9
<code>char c = 'a';</code>	a
<code>double x = 1.2;</code>	1.2
<code>string s1 = "hello, world!";</code>	13   hello, world!
<code>string s1 = "1.2";</code>	3   1.2

زي ما انت شايف الـ string بيكون تمثيله في الـ memory أعقد شوية من مثلاً الـ int، علشان الـ string يحتفظ بعدد الحروف اللي جواه.

خد بالك إن الـ double يخزن رقم، لكن الـ string يخزن حروف. يعني مثلاً، المتغير x يخزن الرقم 2.1، لكن s2 يخزن الثلاث حروف: '1'، '.'، و'2'.

علامات التنصيص بتاعة الـ chars أو الـ string مش بتخزن في الـ الميموري.

كل متغير من نوع int بيكون له نفس الحجم في الـ memory، يعني الـ compiler يخصص نفس المساحة لكل int.

في كمبيوتر أو موبايل عادي، المساحة دي بتكون 4 bytes (يعني 32 bits)، وبالمثل، الأنواع الثانية زي bool و char و double برضو ليها حجم ثابت.

غالباً، هتلاقي الجهاز بيستخدم 1 byte (يعني 8 bits) لكل من الـ bool أو الـ char، و 8 bytes للـ double.

خد بالك إن أنواع الـ objects المختلفة بتاخذ مساحات مختلفة في الـ memory. يعني مثلاً، الـ char بياخذ مساحة أقل من الـ int، وكان الـ string مختلف عن double و int و char، لأنه ممكن ياخذ مساحات مختلفة حسب طول النص اللي فيه.

نقدر نعرف الـ compiler مخصص مساحه قد ايه بظبط لكل variable احنا بنستخدمه بنائاً علي نوع الـ variable باستخدام الـ operator: sizeof

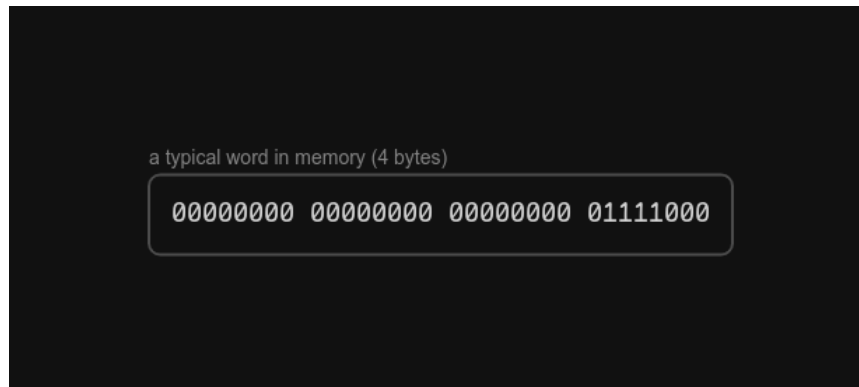
```
int main() {
    int age = 20;
```

```

double pi = 3.14;
char first_letter = 'y';
bool is_male = true;
std::cout << "the variable (age) of type (int) has the size of: " << sizeof age << "\n";
std::cout << "the variable (pi) of type (double) has the size of: " << sizeof pi << "\n";
std::cout << "the variable (first_letter) of type (char) has the size of: " << sizeof first_letter << "\n";
std::cout << "the variable (is_male) of type (bool) has the size of: " << sizeof is_male << "\n";
}

```

معنى bits اللي في الميموري بيعتمد تماماً على النوع اللي بنستخدمه علشان نقرأ أو نكتب في الميموري. يعني تخيلها كده: الميموري بتاعت الكمبيوتر مش عارفة حاجة عن types، هي بس bits وخلاص. bits دي ما بيقاش ليها معنى غير لما إحنا نقرر هنفسرها إزاي. وده شبه اللي بنعمله كل يوم وإحنا بنتعامل مع الأرقام. يعني مثلاً الرقم 5.12، معناه إيه؟ مش معروف كده لوحده. ممكن يكون 5.12 دولار، أو 5.12 سنتيمتر، أو 5.12 لتر. المعنى الحقيقي بيبان بس لما نقول الوحدة. كمثال، نفس bits اللي بتمثل الرقم 120 لما نشوفها كـ int، ممكن تكون بتمثل الحرف 'x' لو بصينا عليها كـ char. ولو حاولنا نفس bits دي نعتبرها string، مش هتفهم خالص وممكن يعمل run-time error لو حاولنا نستخدمها. ممكن نرسم ده بشكل مرئي باستخدام 1 و 0 علشان نوضح شكل bits في الميموري.



المنظر اللي قدامك دا ممكن يتفسر بطريقتين، ممكن يتفسر علي انه int وقيمتة 120، وممكن يتفسر علي انه char وقيمتة 'x' (لو بصينا حصراً علي آخر byte او 8 bits علي اليمين)، مش هيحدد إحنا هنفقراه ازاى غير النوع بتاع الobject.

## ٨ safety type ال

كل object يبقى معمول من type معين وقت ال definition وال type دا عمره ما بيتغير علي مدار البرنامج كله، بنقول علي البرنامج انه safe type لما يبقى كل ال objects بتستخدم بطريقه تناسب مع القوانين بتاعه ال type بتاعها، بمعنى انه بيعمل فقط العمليات المسموحه علي ال type ومش بي mix ال objects الي من types مختلفه بطريقه ممكن تؤدي ل undefined او unsafe او behavior. unpredictable

انت ممكن تسأل نفسك، هو انا اقدر اصلا اعمل كذا؟ المفروض ان كل type يحدد العمليات الي ينفع تتعمل عليه، ولو جربت استخدم object من type معين بطريقه يخالف قوانين ال type زي مثلا اني احط 2.1 في variable من نوع int او اني اطرح strings من بعض s1 - s2 هيطلعلي ايرور، او اني مثلا احاول اجمع int مع string.

ال "Type" safety الكامل هو الهدف والقانون العام في اللغة. بس للأسف، ال ++C compiler لوحده ميقدرش يضمن type "safety" بشكل كامل لكل كود ممكن تكتبه، علشان كده لازم نبعد عن الأساليب الي مش آمنة. يعني لازم نلتزم بشوية قواعد كتابة كود (coding rules) علشان نقدر نحقق type "safety". دلوقتي، ال features الحديثه في ال ++C وأدوات ال analysis الحديثه، بقي ممكن نتأكد من ال safety type في أغلب استخدامات ++C.

الهدف الأساسي هو إننا ما نستخدمش أي ميزة من مميزات اللغة إلا لو نقدر نثبت إنها آمنة من ناحية ال types قبل ما البرنامج يشتغل، وده الي بنسميه "static type". safety وبإستثناء شوية أكواد موجوده في الكتاب علشان تشرح حاجات unsafe فكل الكود الي في الكتاب يطبق قواعد "Core C++ Guidelines" [CG] وارتاجع علشان يكون آمن من ناحية ال types.

فكره ال safety type مهمه جدا لو عايز تبقي مبرمج محترف، علشان كذا احنا بنتكلم عنها بدري اوي كذا في الكورس، لو محترمتش ال safety type هتلاقي عندك مشاكل كتير غامضه في الكود هيقتي صعب انك تتبع مصدرها ومش هتطلعك error واضح صريح يقولك المشكله فين بظبط، علي سبيل المثال شوف الكود دا:

```
int x; // we forgot to initialize x, x's value is undefined
int y = x; // y is initialized to a copy of x's value, which is undefined
double z = 20 + x; // here both the value of x and the meaning of the operation + are undefined
std::cout << "y: " << y << ", z: " << z << '\n';
```

اوعي تنسي تعمل initialization لل variables!!!

فاكر لما كنت بتيجي تقسم علي الصفر علي الاله الحاسبه في اعدادي وتقولك undefined؟ عندنا هنا في البرمجه في undefined برضو بس مش بالمعني دا، لو جيت تقسم علي الصفر هيقولك error عادي، انما لما بنقول undefined دي معناها حاجه ال ++C مش هتقدر تتنبأ بالتصرف بتاعها، يعني انا لو بصيت دلوقتي علي ال standard ISO بتاع ال ++C وبعد كذا بصيت علي الكود الي فوق دا، مش هقدر اقولك

الكود دا هيعمل ايه بظبط، ليه؟ لأن دا علي حسب كان في ايه في ال memory وانت بتنفذ الكود دا، لانك هنا في السطر الأول لما عملت definition لل variable من غير ما تحط فيه قيمه، انت كدا عينتله مكان في ال memory ومخطتش فيه قيمه، لو جيت تستخدم ال variable دا بعد كدا هيجيبك حاجه احنا بنقول عليها value. garbage  
انا مثلاً لما جربت اشغله طلعي ال output دا:

```
~/code $ ./main
y: -885271328, z: -8.85271e+08
~/code $ |
```

غالباً دا مكانش ال output اللي انت كنت متوقع الكود يعملها، ودا مش بسبب قله معرفتك، حتي المبرمجين المحترفين مش هيكبوا كود زي دا لأنهم مش هيقوا قادرين يتنبؤوا بالتصرف اللي هيعمله، علشان كدا ينتقال عليه، undefined، بمعنى ان التصرف بتاعه شئ غير معرف.  
عاده ال compiler يقدر يطلعك messages warning للحاجات اللي شبه كدا تقدر تشغلها ب Wall-

## ٩ conversions ال

نقدر نحول من نوع لنوع تاني في عملية تسمى بال casting implicit بالشكل دا:

```
char c = 'x';
int i1 = c;           // i1 gets the integer value of c (120)
int i2 = c + 1000;    // i2 gets the integer value of c added to 1000 (1120)
double d = i2 + 7.3;  // d gets the floating-point value of i2 plus 7.3 (1127.3)
```

هنا i1 هتبقى ب 120، ودي ال integer value بتاعه 'x' في ال ascii، table، تقدر تعمل دا مع اي حرف علشان تجيب ال numeric representation بتاعه.  
علشان بقا نجيب ال value بتاعه i2 احنا هنعمل arithmetic normal ونجمع integers، two بس ثانيه، c مش integer دي، char علشان كذا قبل ما هنعمل عملية الجمع احنا هنحول او هن promote او هن convert ال c ل int قبل ما نعمل الجمع.  
وكذلك علشان نجيب القيمة بتاعه d ونجمع بين value floating-point و value integer هن promote ال integer value اللي هي في الحالة دي i2 ل double ودا هيطلعنا النتيجة 3.1127  
ال conversions عندنا ليا نوعين:

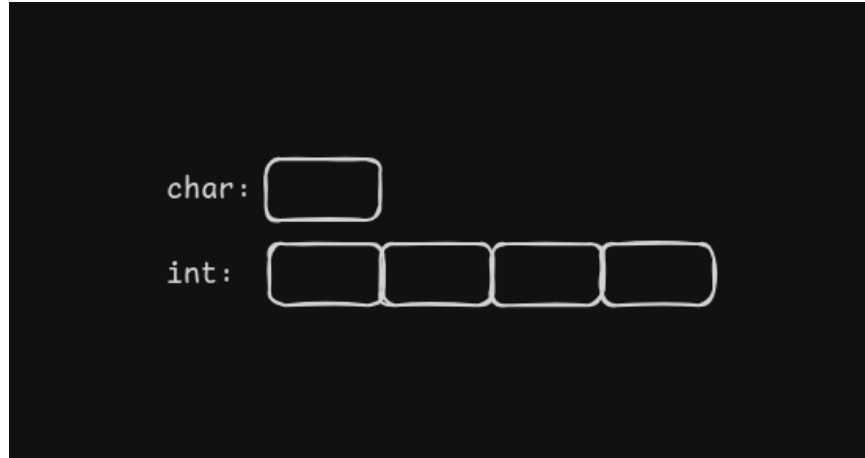
- widening: ودي لما بن convert من حاجه صغيره زي char لحاجه اكبر منها زي int او من int ل double مثلاً

- narrowing: ودي لما بن convert من حاجه كبيره زي int لحاجه اصغر زي char او من double ل int وغالباً هينتج عنها نقص في الداتا

ال conversions widening مفيده لحد كبير زي ما شفنا في المثال اللي فوق، وزى ما انت شفت، ساعات كثير بتحصل بشكل implicit او ال ++C بتعملها scenes the behind ومحتجناش اننا بنفسنا نقوله يحول من int مثلاً ل double قبل ما يجمع.

ال conversions narrowing بقا علي الصعيد الآخر مش مفيده ومعظم الوقت بتؤدي لفقد في الداتا، وللأسف ال ++C برضو ساعات بتعملها بشكل implicit، ولما بنقول narrowing معناها انك بتحاول تحول من نوع لنوع اصغر منه وده ممكن يؤدي لفقد في الداتا، تخيل معايا مثلاً انك عندك كوبايه كبيره فيها ميه وبتحاول تصب منها في كوبايه اصغر، الكوبايه الصغيره هتتلمي وفي ميه كثير هتدلق، نفس الفكره عندنا هنا لما تبجي تحول من نوع كبير او بياخد مساحه كبيره في ال memory لنوع بياخد مساحه اصغر

تخيل معايا مثلاً التحويل من int ل char: زي ما شفنا في المثال بتاعه ال sizeof، ال int بياخد مساحه 4 bytes من ال memory، وال char بياخد 1 byte.



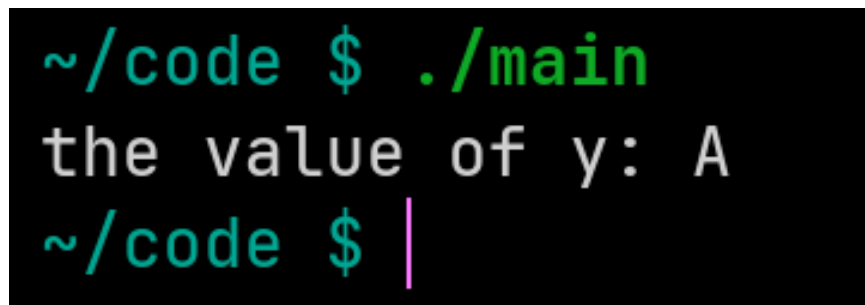
ال 1 byte بتاع ال char دا يقدر يخزن قيم من اول 128 - لحد 127 أو من 0 ل 255 علي حسب ال implementation، علي عكس ال 4 bytes بتوع ال int ممكن يخزنوا من اول -2,147,483,648 لحد 2,147,483,647.

مينفعش نخط رقم كبير زي 321 في char، رقم زي دا ممكن يتخط في int عادي، بس ال char اخره بالكثير 127 او 255 علي حسب ال compiler، والتحويله دي بتؤدي لحاجه بنقول عليها overflow



يعني مثلا لو جربت انك تخط 321 في variable من نوع char هيطلعك output انت ممكن متوقعوش، هيطلعك حرف 'A' بس ازاي؟ ال A في ال table ascii ال representation numeric بتاعها 65 مش 321!

```
int x = 321;
char y = x;
std::cout << "the value of y: " << y << '\n';
```



بص هو الموضوع دا صعب شويه شرحه من غير ما ننزل لتفاصيل level low شويه، بس تقدر تتخيل ان لما تخط رقم زي 321 في variable من نوع char، بيوصل ل 255 وبعد كذا بيبدأ يعد من الأول، فمثلا 256 لو حطيتها في variable من نوع char هتقلب 0، و 257 هتبقى 1 وهكذا، وتقدر تتنبأ بده باستخدام ال operator، module مثلا 321 دي نقدر نشوف هتبقى بكام لو اتخطت في char بالمعادله دي  $(int = c) \% (255 - 1)$  والواحد ده علشان هي بتبدأ تعد من الصفر، بدل الواحد، فمثلا لو بدلنا دي ب 321 هيطلعك c ب 65 وده اللي يخليه يطبعك 'A'

علشان كذا بيتقال عليها conversion narrowing لأن زي ما انت شايف علشان نقدر نحول من int ل char خليناها بدل 321 ل 65، وللأسف برضو زي ما انت شايف ال compiler بيعمل دا عادي، ليه دي مشكله؟ لأن في اوقات كتير احنا اصلا مش بنبقي واخدين بالناس ان في narrowing conversion بتحصل، علي سبيل المثال بص دي:

```
double x = 2.7;
// lots of code....
int y = x; // y becomes 2
```

في الوقت اللي عملنا فيه assign ل x في y، ممكن نكون نسينا ان x دي كانت double وان ال assignment دي هينتج عنها truncation، وال truncation معناه انه هيشيل اي حاجه بعد الفاصله تماما، بدل ما يقربها لأقرب عدد صحيح، اللي حصل دا شئ well-defined في ال standard، بس المشكله ان مفيش حاجه في العمليه بتاعه  $int\ y = x$  دي يفكرنا ان ال 7. دي هتتسأل.

ليه الناس بتقبل مشكلة narrowing conversions؟ السبب الرئيسي هو التاريخ:  
 الـ C++ ورثت conversions narrowing من اللغة الـ C، التي جاية منها، الـ C، فمن أول يوم  
 ظهرت فيه، الـ C كان فيه كود كثير معتمد على conversions narrowing.  
 وكان، كثير من التحويلات دي في الحقيقة ما بتعملش مشاكل، لأن القيم الـ C بتتحول غالباً بتكون  
 جوه الـ range المسموح،  
 وكان فيه مبرمجين كثير مش يحبوا إن الـ compiler "يقولهم يعملوا إيه".  
 خصوصاً إن المشاكل الـ C بتحصل من conversions narrowing بتكون تحت السيطرة لو البرنامج  
 صغير أو المبرمج عنده خبرة. بس في البرامج الكبيرة، الموضوع ممكن يسبب أخطاء كثير، وبالنسبة للمبتدئين،  
 يكون سبب رئيسي للمشاكل.  
 الحلو إن فيه كمبايلات بتطلع warnings عن conversions narrowing - وكثير منها بيعمل  
 كده فعلاً. اسمع كلام الـ compiler لما يحذرك.  
 ولأسباب تاريخية وعملية، الـ C++ بتوفر 4 طرق لكافة الـ initialization.

```
int x0 = 7.8;    // narrows, some compilers warn
int x1 {7.8};   // error : {} doesn't narrow
int x2 = {7.8}; // error : ={} doesn't narrow (the redundant = is allowed)
int x3 (7.8);   // narrows, some compilers warn
```

الـ = و {} = كانوا موجودين من أيام الـ C. إحنا بنستخدم = لما الـ initialization يكون بسيط  
 وينسخ الـ initializer او القيمه المبدئيه. وينستخدم {} أو {} = لما الـ initialization بيتقي معتد  
 شوية أو لما نحب الـ compiler يمنع narrowing في وقت الـ compile-time.

```
int x = 7;
double d = 7.7;
std::string s = "Hello, World\n";
```

```
std::vector v = {1, 2, 3, 5, 8 }; // will explain this in future lectures
std::pair p {"Hello",17}; // will explain this in future lectures
```

اما بالنسبة للـ initialization بال () فده احنا بنستخدمه في حالات محدده جدا

## ١٠ review

١. What is meant by the term prompt?

٢. Which operator do you use to read a variable into?

object? an initialize to use you can notations What .۳

variable a for program your into value integer an input to user the want you If .۴

to user the ask to write could you code of lines two are what number, named  
program? your into value the input to and it do

serve? it does purpose what and called \n is What .۵

string? a into input terminates What .۶

integer? an into input terminates What .۷

code: of line single a as following the write you would How .۸

```

    "; "Hello, << std::cout •
    first_name; << std::cout •
    "!\n"; << std::cout •

```

object? an is What .۹

literal? a is What .۱۰

there? are literals of kinds What .۱۱

variable? a is What .۱۲

double? a and int, an char, a for sizes typical are What .۱۳

as such memory, in entities small of size the for use we do measures What .۱۴

strings? and ints

?== and = between difference the is What .۱۵

definition? a is What .۱۶

assignment? an from differ it does how and initialization an is What .۱۷

C++? in work it make you do how and concatenation string is What .۱۸

int? an to apply you can operators What .۱۹

why legal, not is name a If C++? in names legal are following the of Which .۲۰

not?

```

This_little_pig •
fine This_1_is •

```

```

2_For_1_special •
    thing latest •
        George@home •
            _this_is_ok •
                MineMineMine •
                    number •
                        correct? •
                            stroustrup.com •
                                $PATH •

```

are they because use shouldn't you that names legal of examples five Give .٢١  
 confusion, cause to likely

names? choosing for rules good some are What .٢٢

important? it is why and safety type is What .٢٣

thing? bad a be int to double from conversion can Why .٢٤

safe is another to type one from conversion a fi decide help to rule a Define .٢٥  
 unsafe, or

conversions? undesirable avoid we can How .٢٦

## exercises ١١

lecture, this from exercises THIS TRY the do already, so done haven't you If .١

program Your kilometers, to miles from converts that C++ in program a Write .٢  
 miles, of number a enter to user the for prompt reasonable a have should  
 kilometers, 609.1 is mile A Hint:

variables of number a declares but anything, do does't that program a Write .٣  
 see can you that so ,(;0 = double int as (such names illegal and legal with  
 reacts, compiler the how

Store values, integer two enter to user the prompts that program a Write .٤  
 to program your Write val2, and val1 named variables int in values these  
 these of ratio and product, difference, sum, larger, smaller, the determine  
 user, the to them report and values

and values floating-point enter to user the ask to above program the Modify .6  
 programs two the of outputs the Compare variables. double in them store  
 be? they Should same? the results the Are choice. your of inputs some for  
 difference? the What's

then and values, integer three enter to user the prompts that program a Write .7  
 the fi So, commas. by separated sequence numerical in values the outputs  
 are values two If .10 ,6 ,4 be should output the ,6 4 10 values the enters user  
 should 4 5 4 input the So, together. ordered be just should they same, the  
 .5 ,4 ,4 give

As even. or odd is it fi determine to value integer an test to program a Write .8  
 don't words, other In complete. and clear is output your sure make always,  
 an is 4 value The like alone, stand should output Your no. or yes output just  
 .% operator modulo the use Hint: number. even

"two" and "zero" as such numbers spelled-out converts that program a Write .9  
 program the number, a inputs user the When .2 and 0 as such digits, into  
 4 and ,3 ,2 ,1 ,0 values the for it Do digit. corresponding the out print should  
 doesn't that something enters user the fi know I number a not out write and  
 .99 or computer! stupid as such correspond,

and operands two by followed operation an takes that program a Write .10  
 example: For result. the outputs

14.3 100 + .  
 5 \*4 .

in double that put to tries and input, as double a takes that program a Write .11  
 with program this Run results. the outputs and variables char and int an  
 inputs: of verity

.3 or 2 like values: Small .  
 .1000 or 255 ,127 than larger numbers: Large .  
 values. Negative .  
 .128 and 89 ,56 .  
 .2.56 or 9.65 like values: floating-point .

converted. when results 'unreasonable' produces program your that find will you  
 into liters 4 (about pot pint a into gallon a pour to trying are you basically  
 glass). 500ml