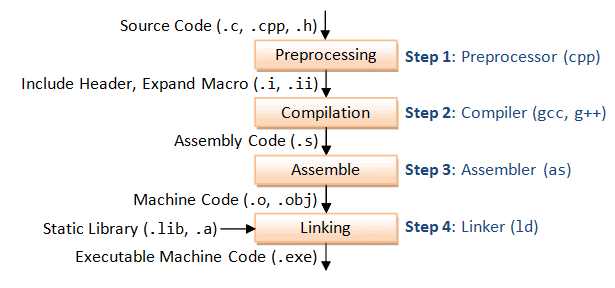
**Compilation Process**



### **1. Preprocessing: The First Step**

At the outset, the C preprocessor comes into play. It handles preprocessor directives, which begin with a '#' symbol, and includes header files in the code. When you use [hashtag #include](https://www.linkedin.com/feed/hashtag/?keywords=include) <stdio.h>, for example, the preprocessor replaces it with the content of the **stdio.h** file. This step generates an intermediate code with all the necessary declarations and macros.

### **2. Compilation: Translating to Assembly**

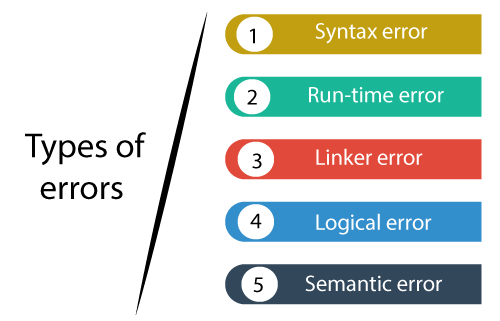
### Once preprocessing is complete, the actual compilation commences. The compiler takes the C code and translates it into assembly code, specific to the target platform. Assembly code represents a low-level view of the program, consisting of instructions that the processor can execute.

### **3. Assembly: From Assembly to Machine Code**

### Next, the assembler converts the assembly code into machine code, also known as object code. The object code consists of binary representations of instructions and data. This step is a crucial bridge between human-readable code and the language the computer understands.

### **4. Linking: Bringing It All Together**

### Now that we have object files with machine code, the linker takes center stage. It resolves external dependencies, such as functions from libraries or other object files, and combines all the object files into a single executable. The linker ensures that the final program is self-contained and ready to run.

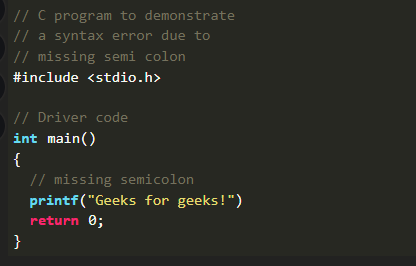
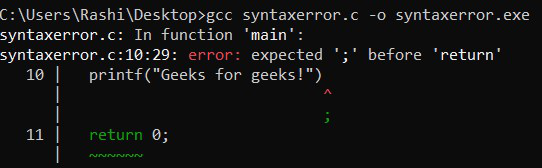
**Types of Errors**

عند حدوث خطأ في البرنامج، يُطلق عليه "بج" (Bug) أو "فولت" (Fault). عملية اكتشاف هذا الخطأ وتصحيحه تُعرف بـ "تصحيح الأخطاء" أو "Debugging".

**Syntax Errors**

المسؤول عن اكتشاف هذا الخطأ هو المترجم (Compiler)، ويحدث عندما يُكتب شيء يخالف قواعد اللغة البرمجية أو عند نسيان أي من الرموز الضرورية مثل الفاصلة المنقوطة (;) أو الأقواس. يُطلق على هذا النوع من الأخطاء "خطأ في بناء الجملة" (Syntax Error) أو "خطأ في الترجمة" (Compilation Error). يحدث هذا الخطأ في وقت الترجمة (Compilation Time) ويمكن تمييزه من خلال وحدة التحكم (Console) حيث يظهر قبل وصف الخطأ كلمة "Error:".

These are also referred to as compile-time errors. These errors have occurred when the rule of C writing techniques or syntaxes has been broken. These types of errors are typically flagged by the compiler prior to compilation.



**Runtime Errors**

الأخطاء أثناء وقت التشغيل (Runtime Errors) تحدث بعد إنشاء ملف .exe وأثناء تنفيذ البرنامج. يمكن تمييز هذا النوع من الأخطاء من خلال مخرجات وحدة التحكم (CMD) حيث تظهر أرقام عشوائية بعد كلمة "returned"، مما يشير إلى حدوث خطأ أثناء وقت التشغيل.

### **بعض الأسباب المحتملة لحدوث أخطاء وقت التشغيل:**

1. **القسمة على صفر:** محاولة قسمة عدد على صفر، مما يؤدي إلى خطأ رياضي.
2. **فتح ملف غير موجود:** محاولة الوصول إلى ملف لا يوجد في المسار المحدد.
3. **نفاذ الذاكرة:** عدم توفر ذاكرة كافية لتنفيذ العمليات المطلوبة، مما يؤدي إلى فشل البرنامج.
4. **خطأ في تقسيم الذاكرة (Segmentation Fault):** محاولة الوصول إلى جزء من الذاكرة غير مصرح به أو غير صالح.

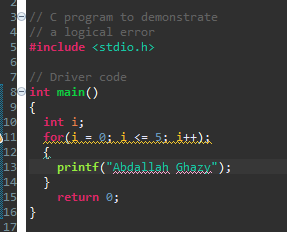
This type of error occurs while the program is running. Because this is not a compilation error, the compilation will be completed successfully. These errors occur due to segmentation fault when a number is divided by division operator or modulo division operator.

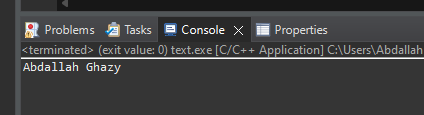
**Logical Errors**

المسؤول عن اكتشاف الأخطاء المنطقية هو المستخدم، حيث أن هذه الأخطاء تؤدي إلى نتائج غير صحيحة أو غير متوقعة. غالبًا ما يكون السبب هو عدم ترتيب الكود بشكل صحيح أو استخدام نوع بيانات غير متوافق.

**أسباب حدوث الأخطاء المنطقية:**

* + **خلل في المنطق البرمجي:** مثل استخدام عمليات حسابية غير صحيحة أو ترتيب غير صحيح للتعليمات.
  + **استخدام متغيرات غير مناسبة:** مثل استخدام نوع بيانات غير متوافق مع العمليات المطلوبة.
  + **شروط غير صحيحة:** كتابة شروط خاطئة في العبارات الشرطية أو حلقات التكرار.

Even if the syntax and other factors are correct, we may not get the desired results due to logical issues. These are referred to as logical errors. We sometimes put a semicolon after a loop, which is syntactically correct but results in one blank loop. In that case, it will display the desired output.



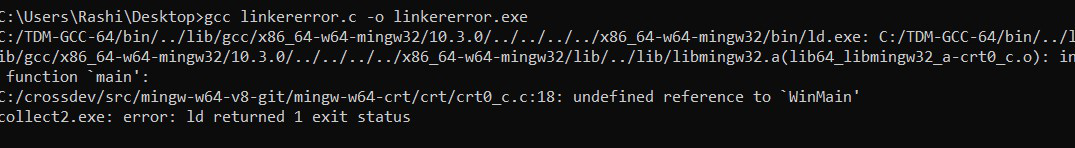
**Linker Errors**

المسؤول عن الإعلان عن أخطاء الربط هو اللينكر (Linker)، وتحدث هذه الأخطاء أثناء عملية توليد الملف التنفيذي (Executable File) حيث يقوم اللينكر بدمج جميع الأكواد في ملف .exe وربطها معًا. يمكن تمييز الخطأ بوجود رسائل تبدأ بـ "error: ld" في شاشة الكونسول.

**أسباب حدوث أخطاء الربط:**

* **تعارض التعريفات (Definition Conflict):** عندما يوجد تعريفات متعددة لنفس المتغير أو الدالة في ملفات مختلفة.
* **تعريف مفقود (Missing Definition):** عندما لا يمكن للينكر العثور على تعريف دالة أو متغير تم الإشارة إليه في الشفرة الأخرى.
* **نوع بيانات غير متوافق (Incompatible Data Types):** عندما تكون هناك تباينات في أنواع البيانات بين التعريفات والاستخدامات المختلفة للمتغيرات أو الدوال.

When the program is successfully compiled and attempting to link the different object files with the main object file, errors will occur. When this error occurs, the executable is not generated. This could be due to incorrect function prototyping, an incorrect header file, or other factors. If main() is written as Main(), a linked error will be generated.



**Semantic Errors**

أخطاء الدلالة تحدث عندما يتم تنفيذ إجراءات غير مفهومة للمترجم (Compiler)، وهذه الأخطاء لا تتمثل في صياغة البرنامج بشكل نحوي صحيح ولا تظهر رسائل خطأ أثناء مرحلة الترجمة. بدلاً من ذلك، قد تؤدي هذه الأخطاء إلى سلوك غير متوقع أو أداء غير صحيح للبرنامج عند تنفيذه.

**تعريف متغير محلي ثم جمع قيم عشوائية:** في هذه الحالة، يمكن أن تكون القيم المخزنة في المتغير المحلي عشوائية، مما يؤدي إلى نتائج غير متوقعة.

**الوصول إلى عنصر غير موجود في مصفوفة:** يمكن أن يؤدي الوصول إلى عنصر خارج نطاق المصفوفة إلى سلوك غير متوقع أو حتى إلى تعطل البرنامج.

**تخزين نوع بيانات غير متوافق في متغير:** يمكن أن يؤدي تخزين قيمة من نوع بيانات غير متوافق في متغير إلى تشغيل غير صحيح للبرنامج أو إلى فشل في التنفيذ.

When a sentence is syntactically correct but has no meaning, semantic errors occur. This is similar to grammatical errors. If an expression is entered on the left side of the assignment operator, a semantic error may occur.

**ما الفرق بين Logical Errors و Semantic Errors**

### **أخطاء الدلالة (Semantic Errors):**

1. **التعريف:**
   * أخطاء الدلالة تحدث عندما يتم تنفيذ إجراءات غير مفهومة للمترجم (Compiler)، حيث يؤدي هذا التصرف إلى سلوك غير متوقع للبرنامج عند تشغيله.
2. **سبب الحدوث:**
   * تحدث هذه الأخطاء نتيجة لاستخدام تعليمات أو أوامر لا تتناسب مع بنية البرنامج، مما يؤدي إلى تنفيذ عمليات غير صحيحة أو غير متوقعة.
3. **أمثلة:**
   * مثال على أخطاء الدلالة هو محاولة تخزين قيمة من نوع بيانات غير متوافق في متغير، أو استخدام متغيرات بدون تعريفها بشكل صحيح، أو الوصول إلى عناصر خارج نطاق المصفوفة.

### **الأخطاء المنطقية (Logical Errors):**

1. **التعريف:**
   * الأخطاء المنطقية تحدث عندما يكون المنطق البرمجي غير صحيح، حيث يؤدي هذا إلى إنتاج نتائج غير متوقعة أو غير صحيحة عند تنفيذ البرنامج.
2. **سبب الحدوث:**
   * يحدث هذا النوع من الأخطاء بسبب خلل في التفكير البرمجي أو استخدام خوارزميات أو تسلسلات غير صحيحة للعمليات البرمجية.
3. **أمثلة:**
   * مثال على أخطاء المنطق هو حساب متوسط القيم لمصفوفة بطريقة خاطئة، أو استخدام شرط غير صحيح في تنفيذ الحلقات، أو استخدام عمليات حسابية غير صحيحة.

**Toolchain**

### **What is Tool Chaining?**

Tool chaining involves integrating multiple tools to create a pipeline or workflow. Each tool in the chain performs a specific function, and the output of one tool becomes the input for the next. This method is commonly used in software development, including:

* **Compilers and Assemblers:** Converting source code to machine code.
* **Linkers:** Combining multiple object files into a single executable.
* **Debuggers:** Analyzing and fixing code issues.
* **Version Control Systems:** Managing changes to source code over time.
* **Continuous Integration/Continuous Deployment (CI/CD) Tools:** Automating the build, test, and deployment processes.

**Native Tool Chaining**

**Native Tool Chaining** refers to the process of compiling and building software directly on the target system for which the software is intended to run. The toolchain used in this process is designed to work on the same architecture and operating system as the target system. This approach is commonly used in desktop and server environments where the development and execution environments are the same.

#### Characteristics of Native Tool Chaining:

* **Same Architecture:** The tools (compiler, linker, etc.) and the target executable run on the same type of CPU and operating system.
* **Simpler Setup:** No need to configure a separate build environment or handle compatibility issues between different architectures.
* **Direct Testing:** Immediate testing on the target system is possible, simplifying debugging and validation.

**Cross Tool Chaining**

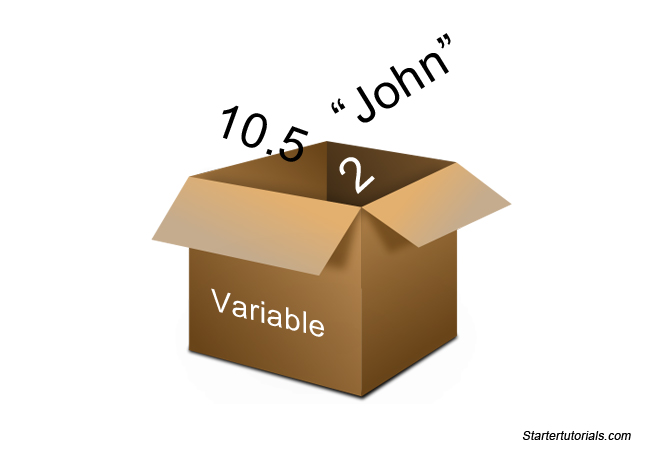
**Cross Tool Chaining** refers to the process of compiling and building software on a host system that is different from the target system where the software will run. The toolchain used in this process is specifically designed to generate code for a different architecture and/or operating system than the one it is running on. This approach is common in embedded systems development, where the target hardware is often different from the development workstation.

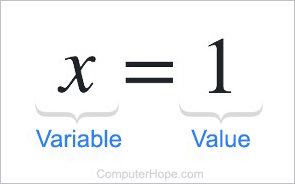
#### Characteristics of Cross Tool Chaining:

* **Different Architectures:** The host system (development machine) and the target system (execution environment) have different architectures.
* **Cross Compiler:** A cross compiler is used to generate binaries for the target architecture.
* **Complex Setup:** Requires setting up a development environment that includes the cross compiler and other necessary tools.
* **Deployment Step:** The compiled code needs to be transferred and flashed onto the target system for testing.

**Variables**

المتغيرات هي أسماء تستخدم للإشارة إلى مواقع أو مساحات في الذاكرة الحاسوبية التي تستخدم لتخزين البيانات أثناء تشغيل البرنامج. يمكن أن تكون هذه البيانات أرقاماً، نصوصاً، أو قيم من أنواع بيانات أخرى.





**Variable Name**



**Variable Definition, Declaration, initialization in C**

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows −



Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows –



**What is the difference between declaring a variable and defining a variable?**

**Declaration** of a variable in C hints the compiler about the type and size of the variable in compile time. Similarly, declaration of a function hints about type and size of function parameters. No space is reserved in memory for any variable in case of declaration.

* **Example**: int a;

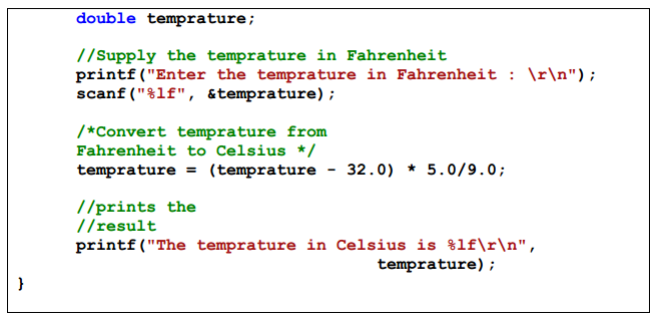
Here variable 'a' is declared of data type 'int'

**Defining** a variable means declaring it and also allocating space to hold it.

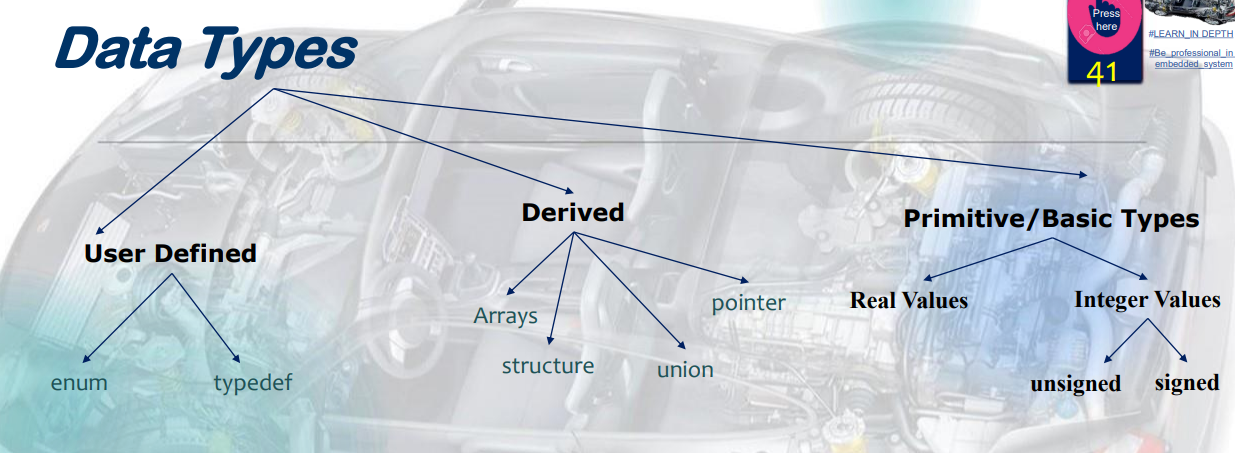
* **We can say "Definition = Declaration + Space reservation".**
* **Example**: int a = 10;

Here variable "a" is described as an int to the compiler and memory is allocated to hold value 10

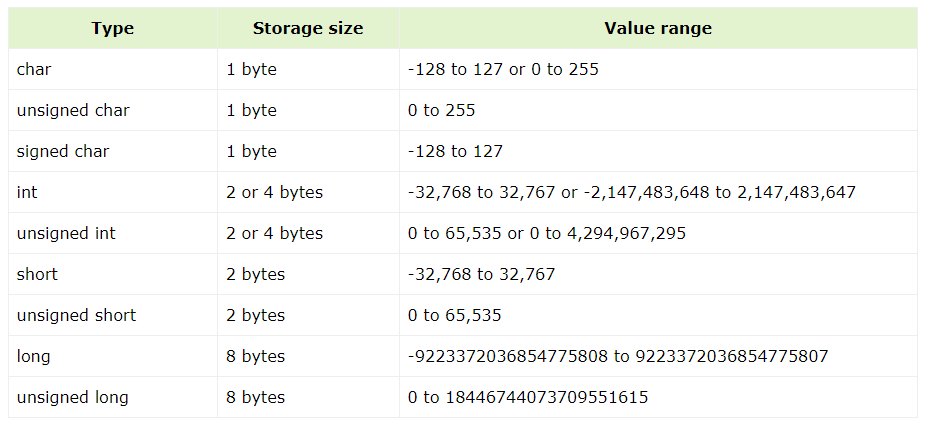
**Comments**



**Data Types**



## **Integer Data Types in C**



To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator. The expressions **sizeof(type)** yields the storage size of the object or type in bytes.

* Sizeof is a unary operator and not a function
* **Unary Operator**: sizeof is considered a unary operator because it operates on a single operand, which can be a type, a variable, or an expression.
* Sizeof int depends on the machine (4Bytes or 2Bytes)

**Range: 2Bytes**

* Unsigned range: 0 to 65535 (2N -1)
* Signed range: -32768 to +32767 ( (2N-1) to (2N-1 – 1) )

**Modifiers used for integer**

**Short, long, signed, unsigned**

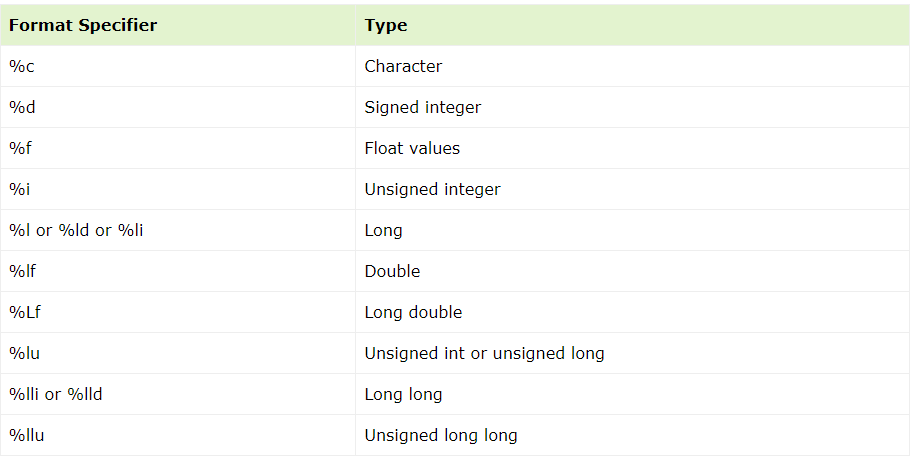
يمكنك استخدام المحدِّدات (modifiers) مع نوع البيانات int في لغة C لتعديل حجمه ونطاق الأرقام التي يمكنه تخزينها.

**Sizeof (int) = 4Bytes or 2Bytes** depends on the machine

sizeof(short int) = 2Bytes

sizeof(long int) = 8Bytes

### **Format Specifiers**



## **What Happen When We Exceed Valid Range of Built-in Data Types in C?**

### **Signed Integers (signed int)**

### Signed integers can hold both positive and negative values. The range of a signed int typically depends on the compiler and the system architecture, but it is commonly from -2,147,483,648 to 2,147,483,647 (assuming 32-bit int).

* **Exceeding the Upper Limit**: If you exceed the maximum positive value (INT\_MAX), the behavior is undefined. This means the compiler or runtime environment may handle it in unexpected ways. It could wrap around (**become negative**), crash the program, or produce incorrect results.
* **Exceeding the Lower Limit**: Similarly, if you go below the minimum negative value (INT\_MIN), the behavior is undefined. It could wrap around (**become positive**), crash the program, or produce incorrect results.

### **Unsigned Integers (unsigned int)**

### Unsigned integers can only represent non-negative values (including zero). The range of an unsigned int is typically from 0 to 4,294,967,295 (assuming 32-bit unsigned int).

* **Exceeding the Upper Limit**: If you exceed the maximum value (UINT\_MAX), the behavior wraps around. This means the value will start again from zero and continue counting up. For example, UINT\_MAX + 1 **results in 0.**
* **Exceeding Zero**: Unsigned integers cannot represent negative values. If you attempt to assign a negative value or perform operations that result in a negative value (like subtracting a larger number from a smaller one), the result is still well-defined within the modulus of UINT\_MAX + 1.

## **Floating-Point Data Types in C**

يبدو أنك تشير إلى الفروق بين النوعين float و double في لغة البرمجة، حيث تختلف في الحجم والدقة التي يمكن تمثيلها. إليك شرح موجز لكل نوع:

### **Float (float)**

* **حجم الذاكرة**: عادةً ما يشغل float 4 بايت في الذاكرة.
* **دقة التمثيل**: يوفر تمثيلًا بدقة تصل إلى حوالي 7 أرقام بعد الفاصلة العشرية.
* **مثال**: إذا كان لدينا float x = 3.14159265358979;، قد يتم تقريب القيمة المخزنة لتصبح قريبة من 3.141593.

### **Double (double)**

* **حجم الذاكرة**: عادةً ما يشغل double 8 بايت في الذاكرة، أي ضعف حجم float.
* **دقة التمثيل**: يوفر تمثيلًا بدقة أكبر، تصل إلى حوالي 15-16 رقمًا بعد الفاصلة العشرية.
* **مثال**: إذا كان لدينا double y = 3.14159265358979;، يمكن أن يحتفظ بالقيمة بدقة أكبر مقارنة بـ float.

### التمثيل الرمزي:

* **Float** يمكن تمثيله بمثل 3.141593 حيث يكون العدد المكون من 32 بت.
* **Double** يمكن تمثيله كذلك ب 3.14159265358979

**What is the output?**

**#include** <stdio.h>

**#include** <limits.h>

**int** **main**() {

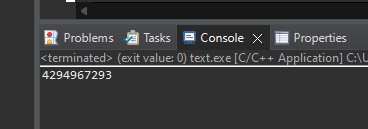
**unsigned** **int** i = 1;

**int** j = -4;

**printf**("%u",i+j);

**return** 0;

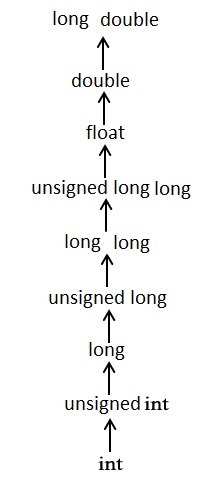
}



ناتج عملية i+j هو 3- لكن بسبب وجود %u و معناها اطبع الأرقام unsigned حيث ان هناك خانة للرقم السالب لكن مع استخدام %u ترجم الخانة علي انها من الأرقام و بدل مايطلبع 3- طبع 4294967293

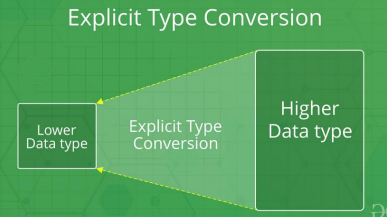
# Type Conversion in C

## **Implicit Type Conversion in C**

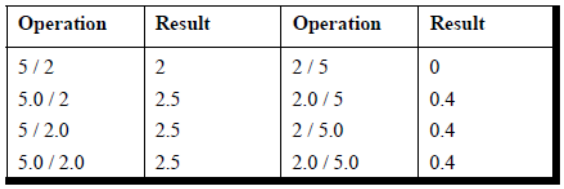
In C, implicit type conversion takes place automatically when the compiler converts the type of one value assigned to a variable to another data type. It typically happens when a type with smaller byte size is assigned to a "larger" data type. In such implicit data type conversion, the data integrity is preserved.

## **Explicit Type Conversion in C**

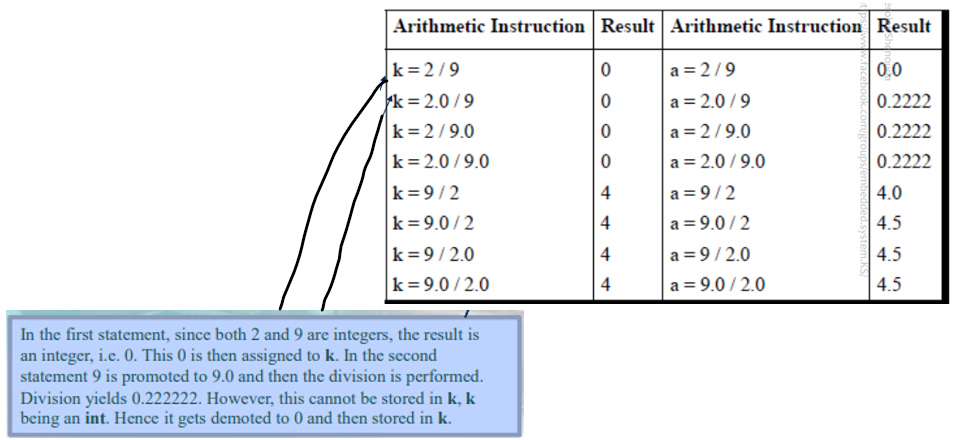
When you need to covert a data type with higher byte size to another data type having lower byte size, you need to specifically tell the compiler your intention. This is called explicit type conversion.



**integer and Float Conversions**

****



k is an integer variable and a is a real variable.

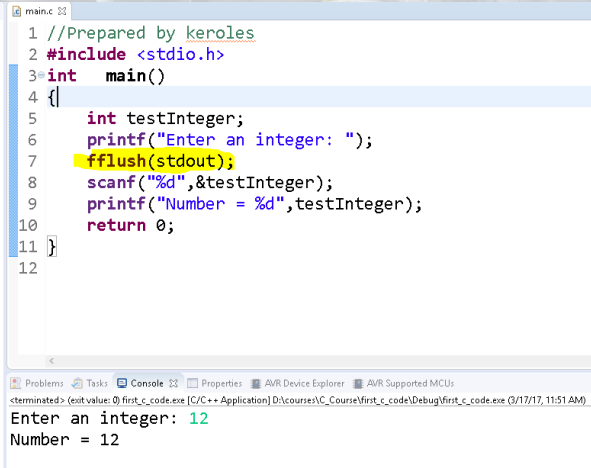
**C Programming Input Output (I/O): printf() and scanf()**

The scanf() function reads formatted input from standard input (keyboard) whereas the printf() function sends formatted output to the standard output (screen).

Eclipse's terminal emulator might be different and do more buffering.

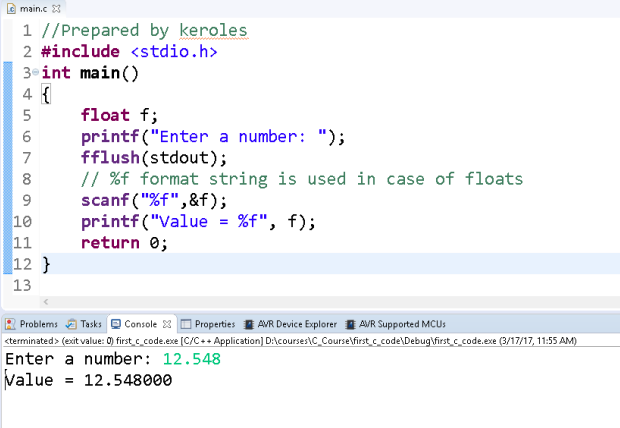
Try calling fflush(stdout); between the printout and the call to scanf().

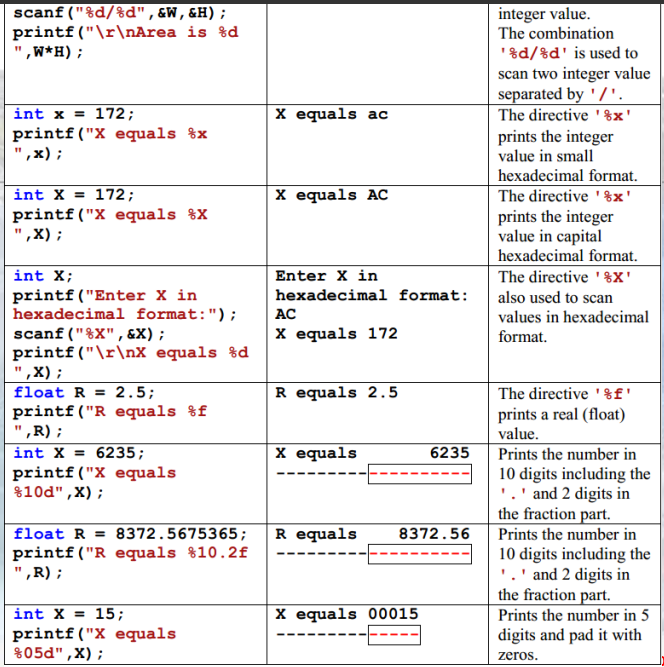
scanf()

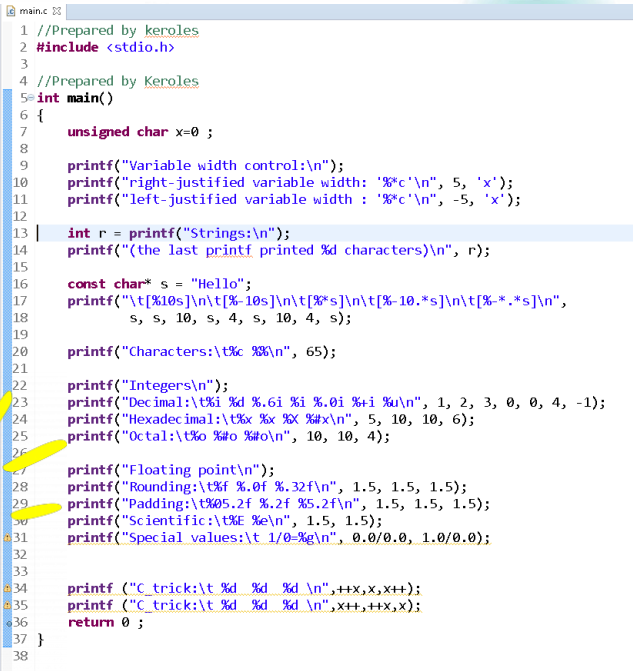


قمنا باستخدام الدالة scanf وداخلها ستجد وسيطين، الأول نقوم فيه بتحديد نوع القيمة التي سيدخلها المستخدم حيث هنا وضعنا الرمز %d سابقا في الدالة printf حيث قلنا أنها خاص بالأعداد الصحيحة، وفي الوسيط الثاني يوجد testInteger ، والرمز& يعني وضع القيمة التي أدخلها المستخدم في عنوان المتغير testInteger

أما بالنسبة لباقي أنواع المتغيرات فسنستعمل نفس الطريقة فقط نقوم بتغير الرمز %d إلى نوع المتغير الذي نريد إستقباله، فمثلا إذا أردنا من المستخدم أن يقوم بإدخال رمز بدل رقم نضع الرمز %c في الدالة scanf

C Floats Input/Output





# 

# C - Operators

* **Unary operators**

++ (increment)

-- (decrement)

! (NOT)

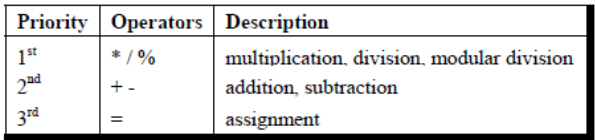
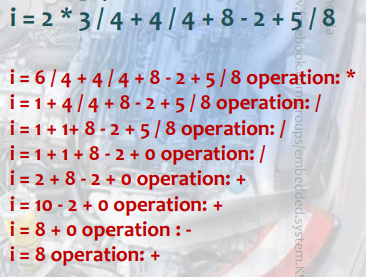
~ (compliment)

& (address of)

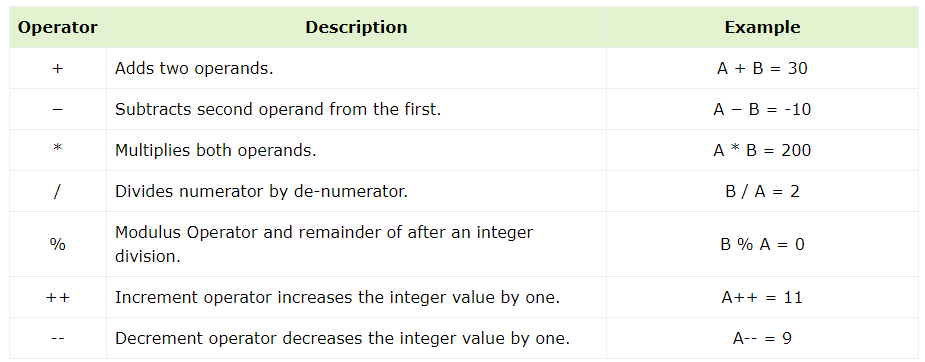
\* (dereference)

* **Binary operators** − arithmetic, logical and relational operators except !
* **Ternary operators** − The ? operator

**Hierarchy of Operations**



## **Arithmetic Operators**



**Pre-increment and Post-increment in C**

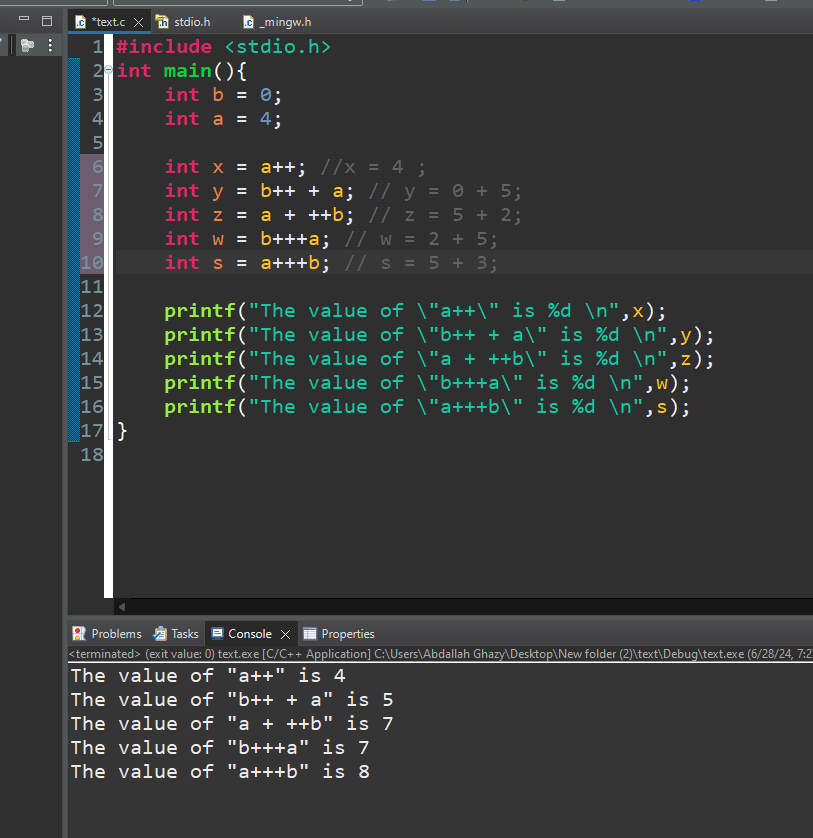
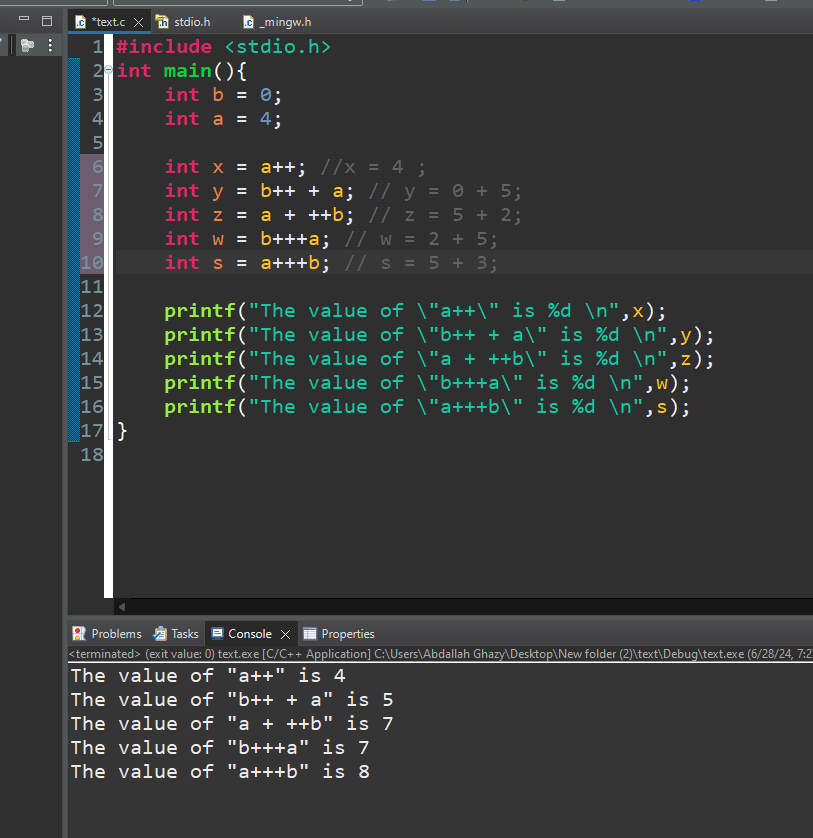
 Increment operators are used to increase the value of a variable by 1. This operator is represented by the ++ symbol.

The increment operator can either increase the value of the variable by 1 before assigning it to the variable or can increase the value of the variable by 1 after assigning the variable.

Thus it can be classified into two types:

1. Pre-Increment Operator (a = ++x;)
2. Post-Increment Operator (a = x++;)

**Ex**

****

**Token Generation and Lexical Analysis in Compilers**

Lexical analysis is the first phase of a compiler, often referred to as the scanning phase. This stage involves the conversion of the source code into a sequence of tokens, which are the fundamental building blocks for syntactic analysis in the subsequent phases of compilation.

**Steps Involved in Lexical Analysis:**

1. **Scanning**:

* The lexical analyzer scans the input source code character by character.
* It groups sequences of characters into lexemes, which represent the smallest meaningful units in the source code.

1. **Token Generation**:

* Once a lexeme is identified, the lexical analyzer converts it into a token.
* A token typically consists of a token name and an attribute value.
* **Example of Token Generation:**



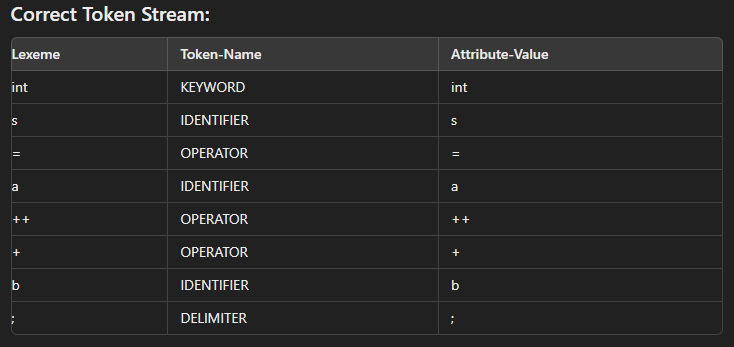
#### Summary:

* **Lexical Analysis** is the first phase of the compiler.
* The **lexical analyzer (scanner)** scans the entire source program to identify meaningful sequences of characters, which are then converted into tokens.
* **Tokens** are pairs consisting of a token-name and an attribute-value.
* The **scanner** facilitates the subsequent phases of compilation by generating a stream of tokens from the source code.

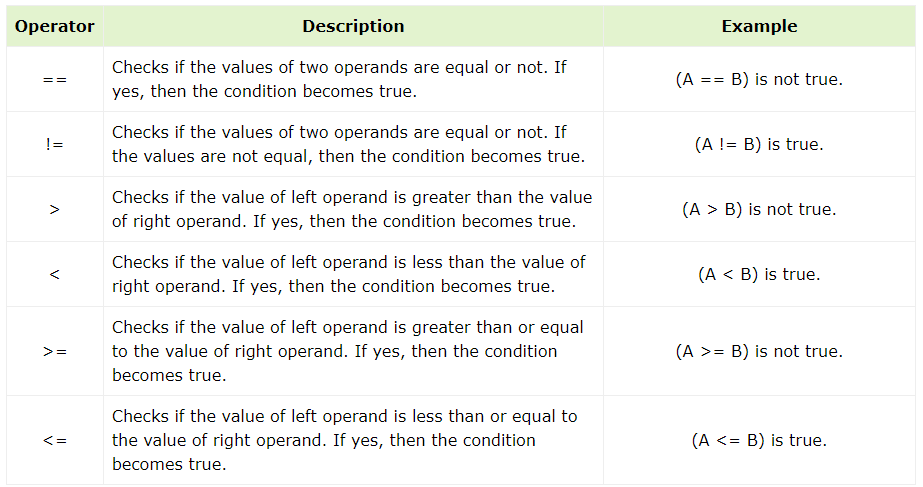
### Understanding the Line int s = a+++b;

This line of code can be confusing due to the sequence a+++b. The compiler interprets this as:

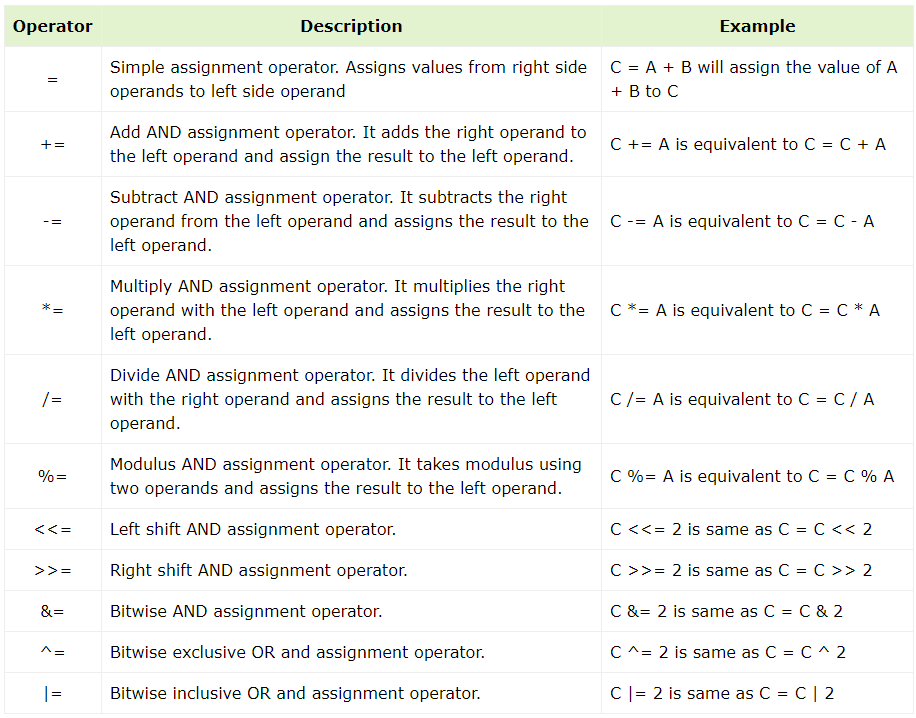
* a++ (post-increment of a)
* + (addition operator)
* b (identifier)



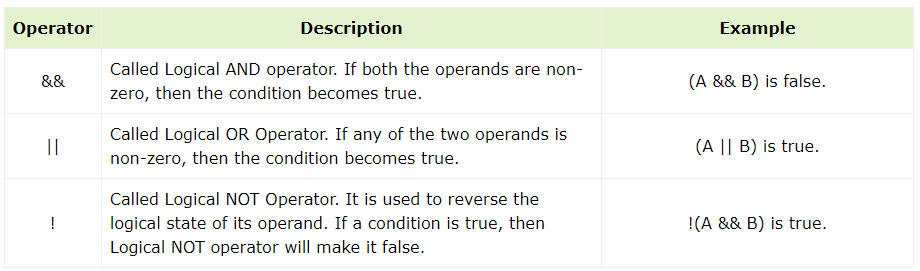
## **Relational Operators**



Assignment Operators



Logical Operators



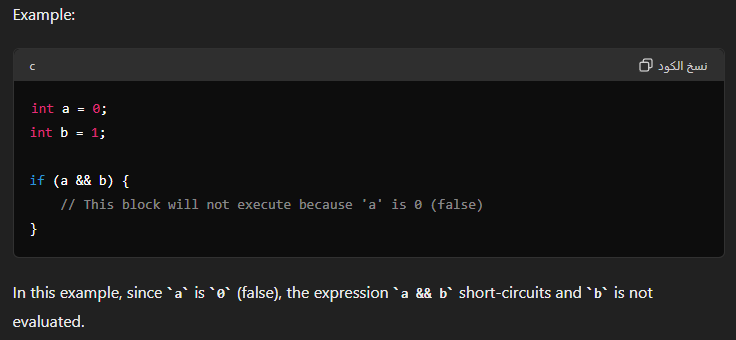
**Concept of short circuit in logical operator**

In C programming, short-circuit evaluation is a concept associated with the logical operators && (logical AND) and || (logical OR). This concept means that the evaluation of logical expressions can be stopped as soon as the outcome is determined. This can improve efficiency and prevent potential errors by avoiding unnecessary evaluations.

**Short-Circuit Evaluation**

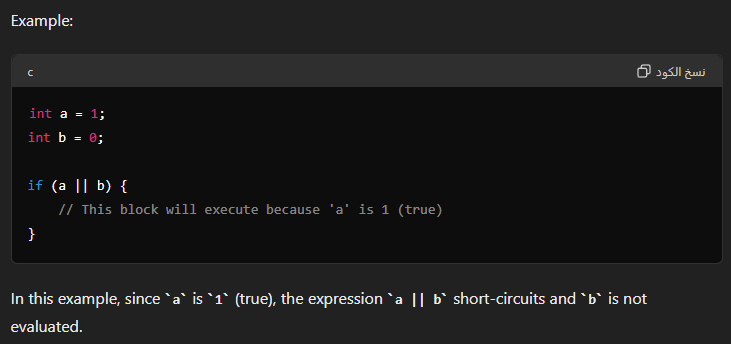
**Logical AND (&&)**

For the && operator, if the first operand evaluates to false (0), the overall expression cannot be true, regardless of the second operand. Therefore, the second operand is not evaluated.



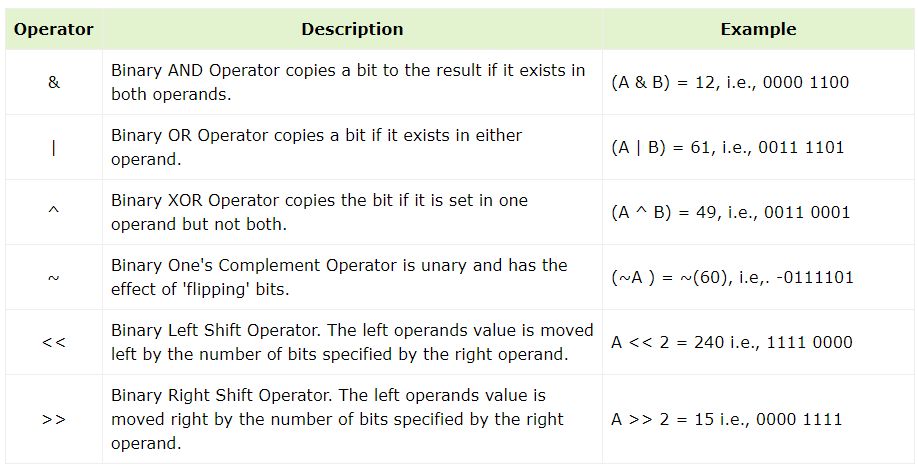
**Logical OR (||)**

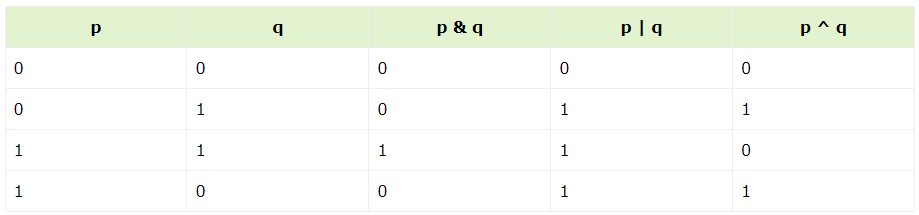
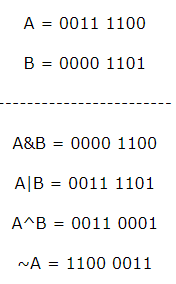
For the || operator, if the first operand evaluates to true (non-zero), the overall expression is true, regardless of the second operand. Therefore, the second operand is not evaluated.



Practical Uses

## **Bitwise Operators**





**Shift left operator (<<)**

First operator << second operator

عدد مرات الازاحة

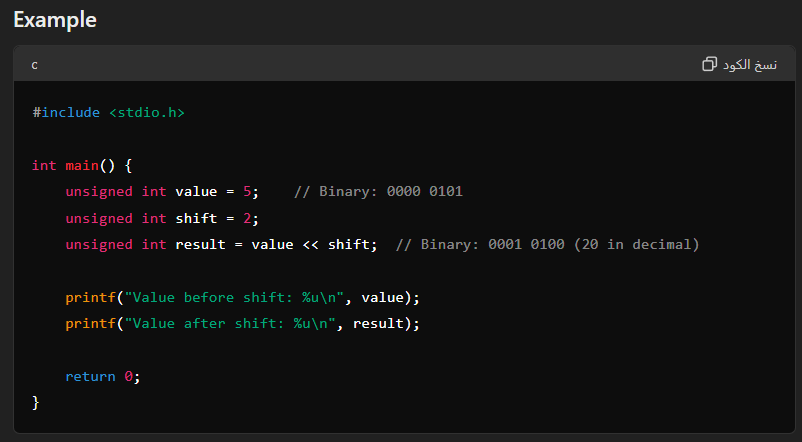
الرقم الذي يتم إزاحة وحدات البت الخاصة به الي اليسار

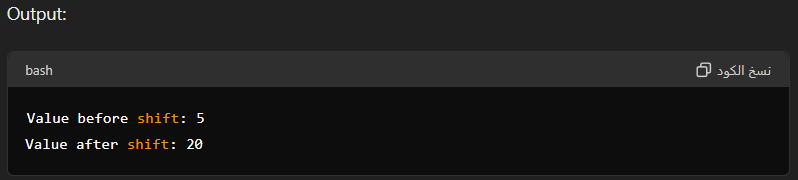
* When bits are shifted left then trailing positions are filled with zero

**Syntax**

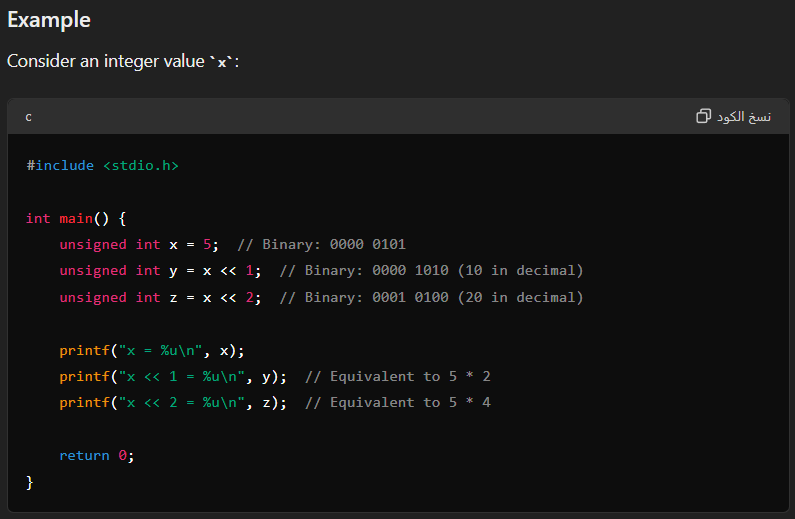
result = value << shift;

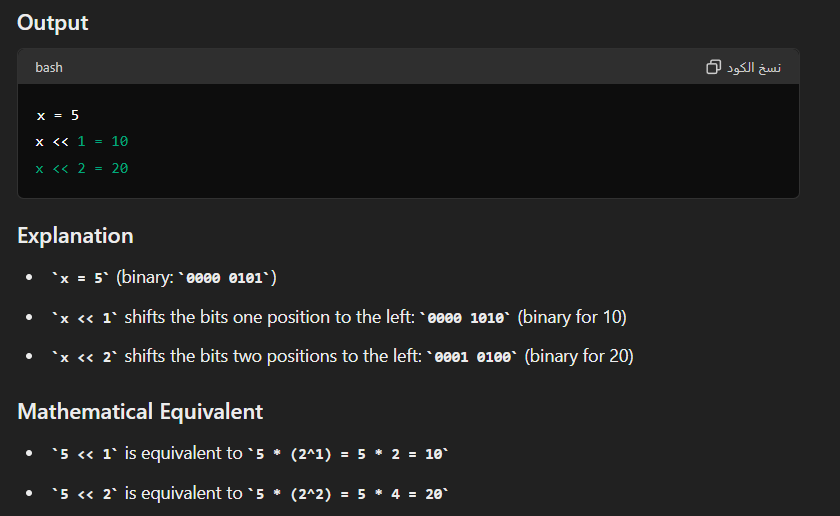
* **value**: The integer value whose bits are to be shifted.
* **shift**: The number of positions to shift the bits to the left.
* **result**: The result of the left shift operation.





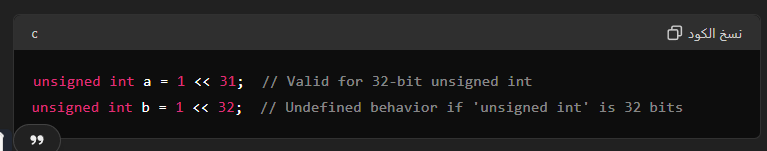
* Left shifting is equivalent to multiplication by 2 **right operands**

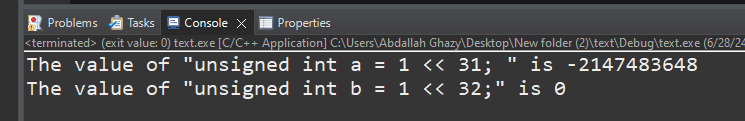




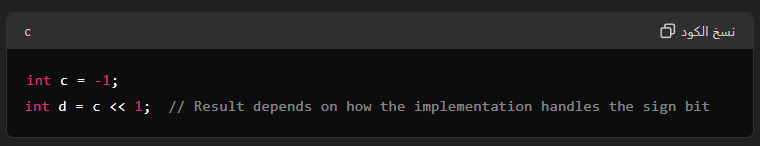
**Important Points**

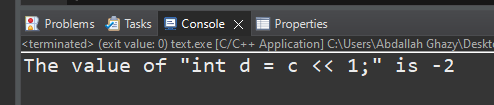
**Overflow**: Be cautious with large values, as left shifting can lead to overflow if the resulting value exceeds the maximum value that can be stored in the data type.





**Signed Integers**: When left shifting signed integers, be careful with the sign bit. If the sign bit is shifted into the value, the result might become negative or undefined, depending on the implementation.





**Comma ( , ) operator**

the comma operator (,) is a binary operator that evaluates two expressions and returns the value of the second expression. This operator is typically used in situations where multiple expressions need to be evaluated in a single statement.

**Syntax**

result = (expression1, expression2);

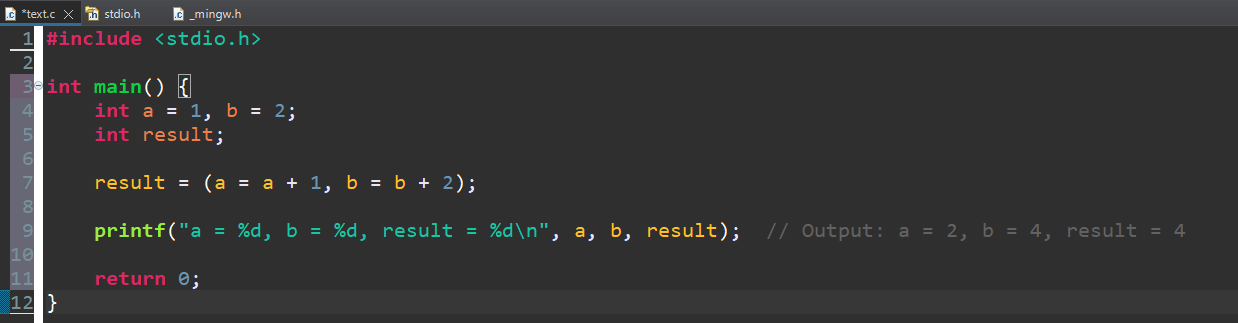
* expression1: The first expression to be evaluated.
* expression2: The second expression to be evaluated, whose value is returned.

**الاستخدام**

1. يستخدم لفصل المتغيرات عن بعضها البعض



1. اذا تم إعطاء اكثر من قيمة للمتغير بين اقواس a = (3,4,5,8) فامه يختار القيمة الاولي من اليمين ولكن يقوم بتنفيذ كل الأوامر التي داخل القوس يعطي خطا اذا كتبتها int a=3,4,5,8 لانه بيبقي شايفها int a = 3 , int 4, int 5 , int 8 عشان كدا نحطها في اقواس

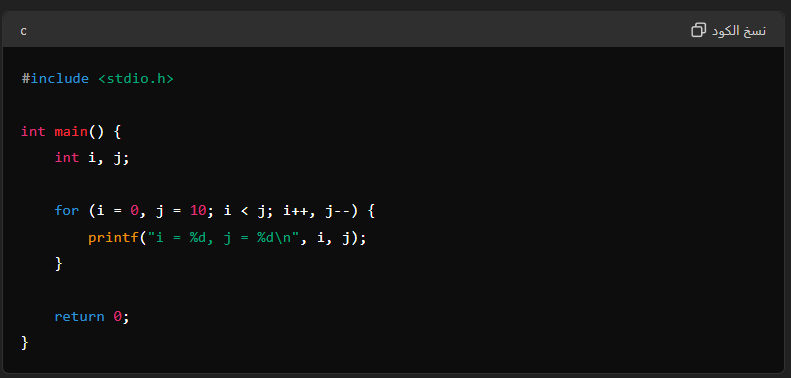
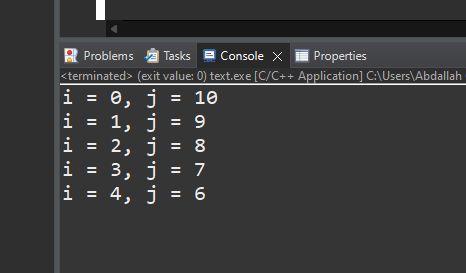


**In this example:**

* a = a + 1 is evaluated first, setting a to 2.
* b = b + 2 is evaluated next, setting b to 4.
* The value of b (which is now 4) is assigned to result.

#### **Using Comma Operator in For Loop**

The comma operator is commonly used in for loops to handle multiple expressions within the loop header.



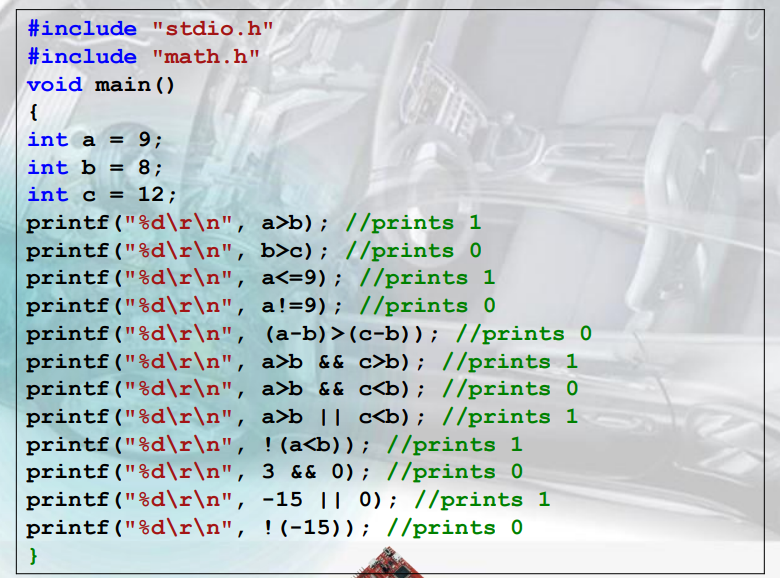
**Example of Combined Expressions**

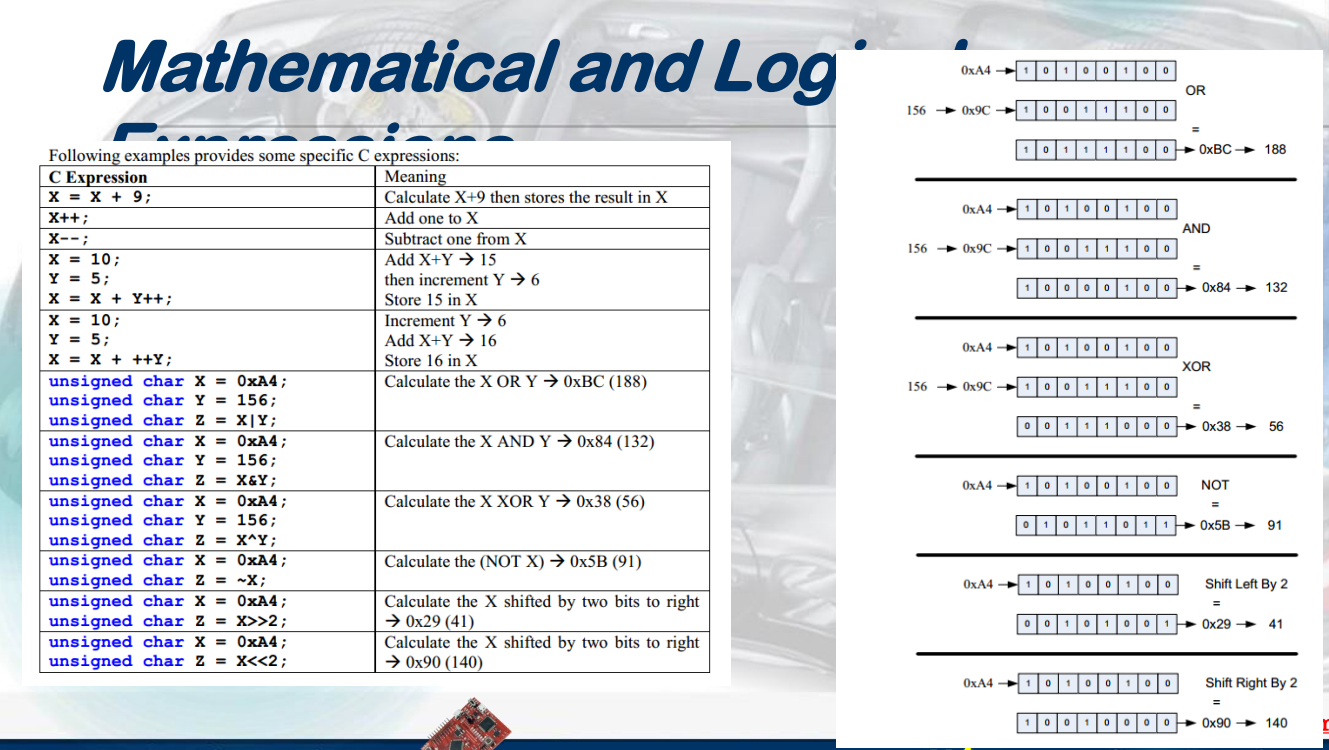


## **Operators Precedence in C**



**Example: Using Conditions**



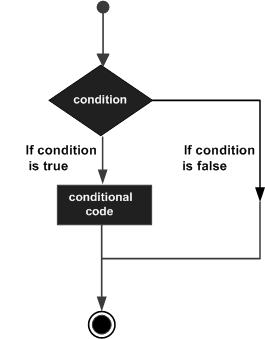


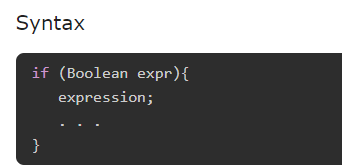
**Identifiers**

Identifiers are the names that are given to various program elements such as variables, symbolic constants and functions.

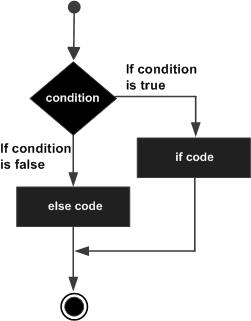
* **Identifier can be freely named, the following restrictions.**
* Alphanumeric characters ( a ~ z , A～Z , 0～9 ) and half underscore ( \_ ) can only be used.
* The first character of the first contain letters ( a ~ z , A～Z ) or half underscore ( \_ ) can only be used.
* **Here are the rules you need to know:**
* Identifier name must be a sequence of letter and digits, and must begin with a letter
* The underscore character (‘\_’) is considered as letter.
* Names shouldn't be a keyword (such as int, float, if, break, for etc)
* Both upper-case letter and lower-case letter characters are allowed. However, they're not interchangeable.
* No identifier may be keyword.
* No special characters, such as semicolon, period, blank space, slash or comma are permitted

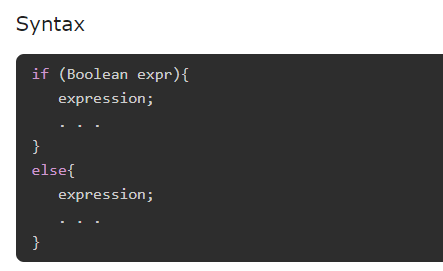
**If Statement in C Programming**



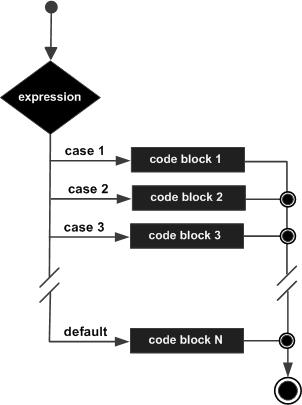


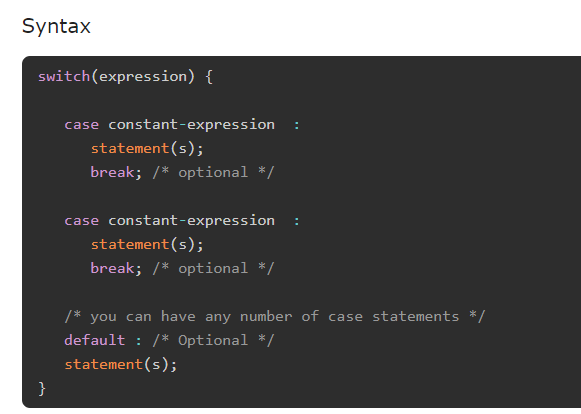
## **If...else Statement in C Programming**





**Switch Statement in C Programming**



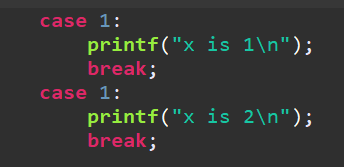
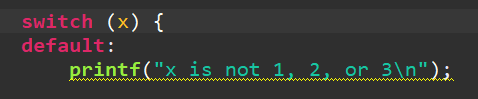


**Differences Between if and switch Statements**

**if Statement**

1. The if statement can handle complex expressions, including relational, logical, and arithmetic operations. Supports conditions involving any data type (integer, floating-point, pointers, etc.).
2. May be slower compared to switch in scenarios involving multiple conditions because each condition is evaluated until a match is found.
3. Conditions are evaluated sequentially from top to bottom.

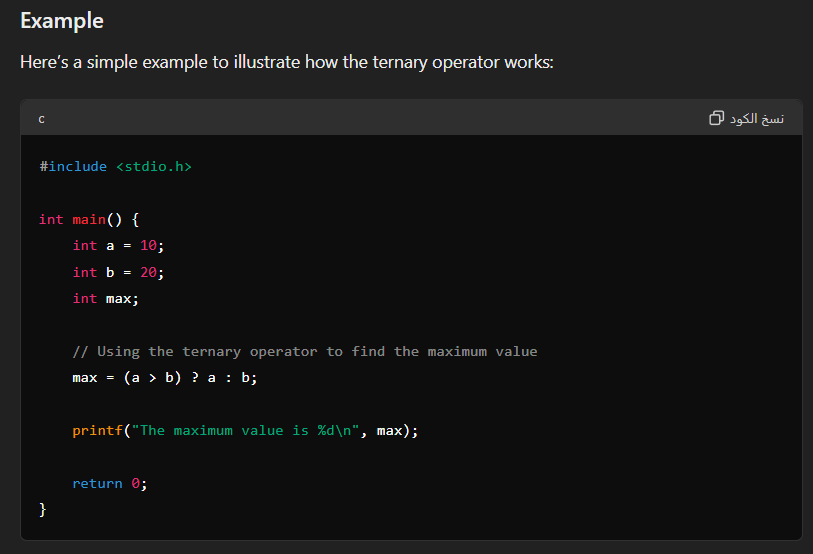
**switch Statement**

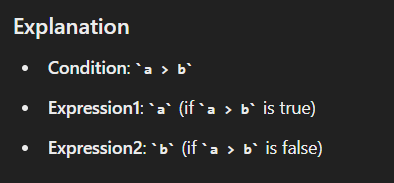
1. faster in execution time for cases involving a large number of discrete values.
2. Utilizes a jump table or binary search for faster lookup.
3. The switch expression must evaluate to an integer or an enumeration type.
4. Does not support floating-point numbers or complex conditions.
5. Faster for large numbers of discrete values because of its optimized handling via jump tables or binary searches.
6. Less flexible compared to if statements in terms of condition complexity.
7. لا يسمح لك بوضع قيم مكررة
8. لا يمكن كتابة المعادلة داخل switch
9. يمكن كتابة معادلة بدل تسمية الحالة
10. يتم تنفيذ كل ال Cases في البداية و بعدها ينفذ ال default حتي لو مكتوبه في بداية ال switch

**The ?: Operator in C Programming**

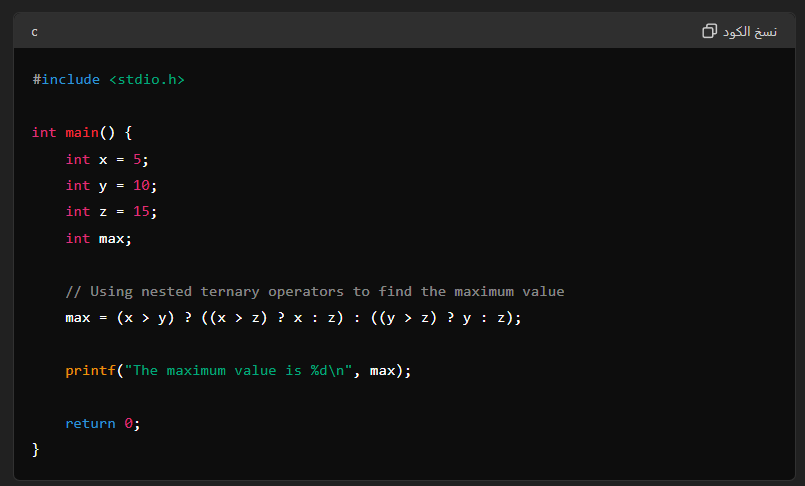
**condition** **?** **expression1** : **expression2**;

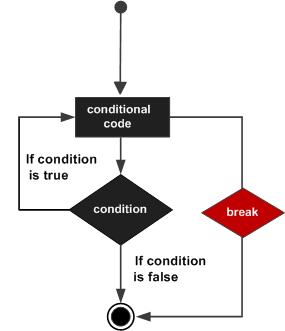
* condition: The expression to be evaluated. If it is true (non-zero), expression1 is evaluated and returned.
* expression1: The expression evaluated and returned if condition is true.
* expression2: The expression evaluated and returned if condition is false.

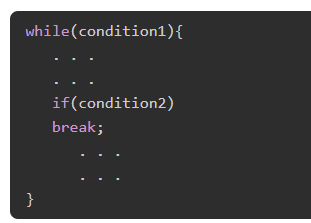


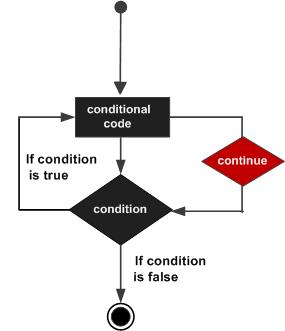


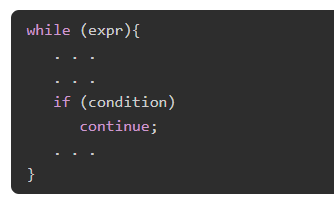
### **Nested Ternary Operators**



**The Break Statement in C Programming**

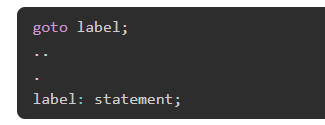
In C, the [break statement](https://www.tutorialspoint.com/cprogramming/c_break_statement.htm) is used in switch–case constructs as well as in loops. When used inside a loop, it causes the repetition to be abandoned.

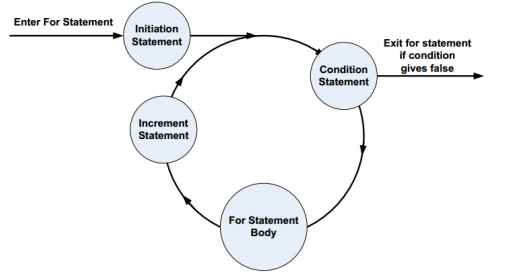
**The Continue Statement in C Programming**

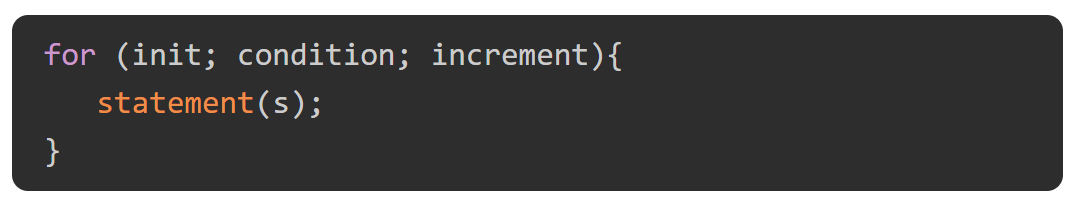
In C, the [continue statement](https://www.tutorialspoint.com/cprogramming/c_continue_statement.htm) causes the conditional test and increment portions of the loop to execute.

## C goto statement**The goto Statement in C Programming**

C also has a [goto keyword](https://www.tutorialspoint.com/cprogramming/c_goto_statement.htm" \t "_blank). You can redirect the program flow to any labelled instruction in the program.

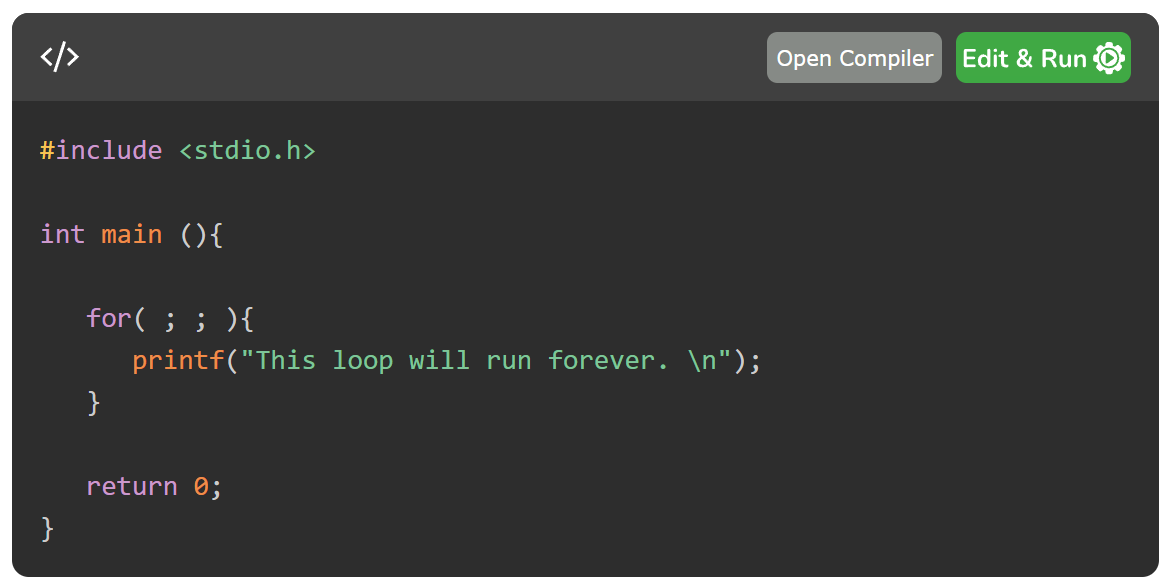


**for Statement**

**Syntax of for Loop**

Using the for loop in C is very useful when you need to execute a block of code a specific number of times. Always be mindful of the counter's size to ensure it has enough space to cover all required iterations without causing an overflow.

**Infinite Loop in C**

****

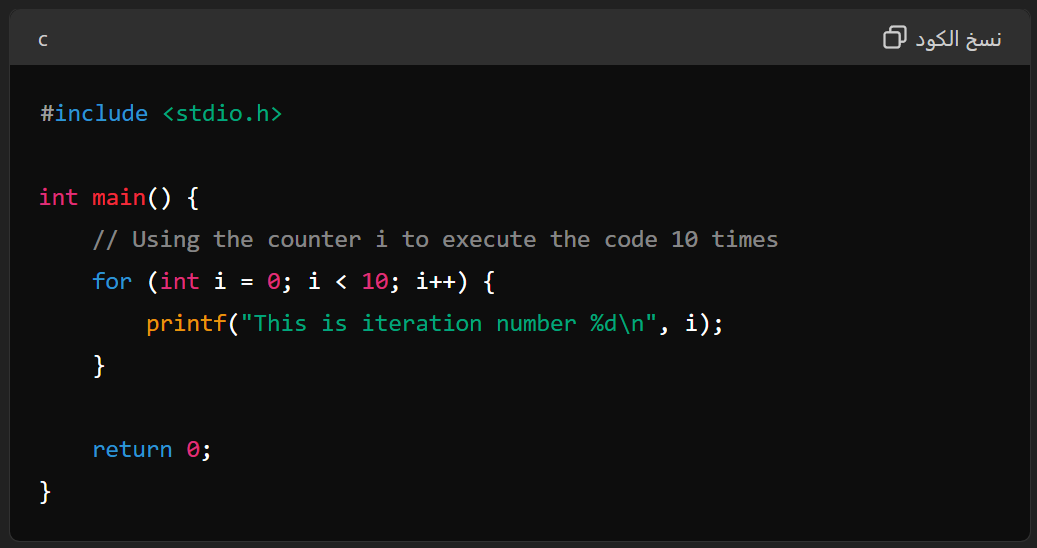
**C - While Loop**

The while loop in C is ideal for situations where the number of iterations is not known in advance. This loop continues to execute the code block as long as the condition remains true.

**Syntax of C while Loop**

****

**Infinite Loop Example**

****

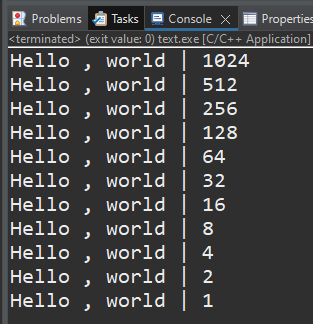
**Do-While Loop in C**

****

The do-while loop in C is used when you need to execute a block of code at least once and then repeat the execution as long as a given condition is true. This loop is similar to the while loop, but with a key difference: the condition is checked after the execution of the loop body, ensuring that the code inside the loop runs at least once.

Loops 🡪 solved problems

Ex **output ?**



**#include** <stdio.h>

**int** **main**() {

**int** i = 1024;

**for** ( ; i ; i>>=1){

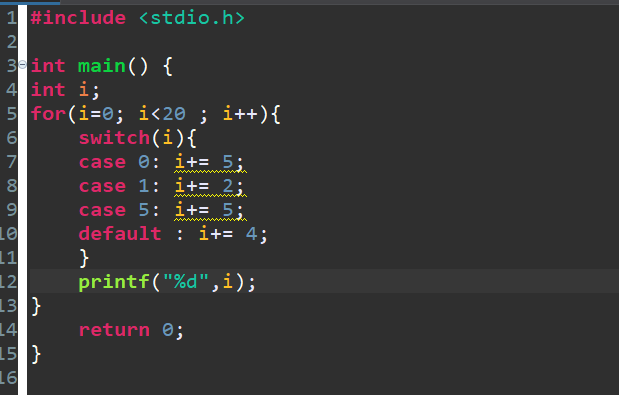
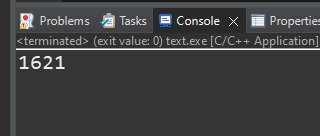
**printf**("Hello , world | %d \n",i);

}

**return** 0;

}

Ex **output ?**



**Special programs**

1. Write a program to check whether a number is an Armstrong number or not.
2. Write a program to check whether a number is a strong number or not.
3. Write a program to 23 and 2-3
4. Write a program to check whether a year is a leap year or not.
5. Write a program to add two numbers without using + operator
6. Write a program convert binary to decimal
7. Swapping Two Variables Without a Temporary Variable
8. Finding the Maximum of Two Numbers Using Ternary Operator
9. Checking if a Number is Even or Odd
10. Printing a Number in Binary Format
11. Write a program to find factorial of the given number.
12. Write a program to swap two numbers using a temporary variable.
13. Write a program to swap two numbers without using a temporary variable
14. Write a program to swap two numbers using bitwise operators.
15. Write a program to find the greatest of three numbers.
16. Write a program to find the greatest among ten numbers.
17. Write a program to check whether the given number is a prime.
18. Write a program to check whether the given number is a palindromic number
19. Write a program to check whether the given string is a palindrome
20. Write a program to generate the Fibonacci series.
21. Write a program to print "Hello World" without using semicolon anywhere in the code.
22. Write a program to print a semicolon without using a semicolon anywhere in the code.



ملحوظة ان if بتنفذ ال بداخلها الأول حتي لو كان x++ هيروح يضيف 1 علي x مش هينزل وبعدين يضيف