Tabella degli operatori di C con precedenze e associatività

Livello	Operatore(s)	Descrizione	Associatività
1	() [] -> .	Accesso a funzione, array, puntatore, membro	Da sinistra a destra
2	++ -- + - ! ~ * & sizeof cast (INTENDE I PREFISSI)	Incremento, Decremento, Positivo, Negativo, Negazione logica, Complemento, Dereferenziamento, Indirizzo, Dimensione	Da destra a sinistra
3	* / %	Moltiplicazione, Divisione, Modulo	Da sinistra a destra
4	+ -	Addizione, Sottrazione	Da sinistra a destra
5	<< >>	Shift a sinistra e a destra	Da sinistra a destra
6	< <= > >=	Confronto minore, minore o uguale, maggiore, maggiore o uguale	Da sinistra a destra
7	== !=	Uguaglianza, Diversità	Da sinistra a destra
8	&	AND bit a bit	Da sinistra a destra
9	^	XOR bit a bit	Da sinistra a destra
10	|	OR bit a bit	Da sinistra a destra
11	&&	AND logico	Da sinistra a destra
12	||	OR logico	Da sinistra a destra
13	?:	Operatore ternario	Da destra a sinistra
14	= += -= *= /= %= <<= >>= &= ^= |=	Assegnamento e assegnamento con operazione	Da destra a sinistra
15	,	Separatore di espression	Da sinistra a destra

1. Informatica e Programmazione

- **Informatica**: Disciplina scientifica che studia l'informazione e il suo trattamento automatico.
- **Computer**: Macchina elettronica programmabile in grado di eseguire diverse funzioni a seconda delle istruzioni ricevute.
- **Algoritmo**: Insieme finito e ordinato di passi eseguibili in tempo finito e non ambigui, che definiscono un processo che termina. Le proprietà fondamentali di un algoritmo sono:
    - **Finitezza**: L'algoritmo deve essere composto da un numero finito di passi.
    - **Non ambiguità**: Ogni passo deve essere definito in modo univoco, senza lasciare spazio a interpretazioni.
    - **Terminazione**: L'esecuzione dell'algoritmo deve terminare dopo un numero finito di passi.
- **Programma**: Espressione di un algoritmo in un linguaggio comprensibile all'esecutore (**computer**) senza ulteriori spiegazioni. Un **programma** è un'entità concreta, legata a un particolare esecutore, mentre un **algoritmo** è un concetto astratto.

2. Dal Linguaggio Macchina ai Linguaggi ad Alto Livello

- **Macchina di Von Neumann**: Architettura di calcolatore che prevede una singola unità di memoria per istruzioni e dati.
- **Linguaggio Macchina**: Linguaggio di programmazione composto da istruzioni in formato binario direttamente eseguibili dal processore.
- **Assembler**: Linguaggio di programmazione che utilizza codici mnemonici per rappresentare le istruzioni del linguaggio macchina, rendendo la programmazione più facile rispetto all'uso del codice binario puro.
- **Linguaggi ad Alto Livello**: Linguaggi di programmazione che offrono astrazioni più vicine al ragionamento umano, indipendenti dall'architettura specifica del processore.
- **Compilatore**: Programma che traduce un intero programma scritto in un linguaggio ad alto livello in un programma equivalente in linguaggio macchina.
- **Interprete**: Programma che simula l'esecuzione di un programma scritto in un linguaggio ad alto livello, traducendo ed eseguendo le istruzioni una alla volta.

- **Linker**: Programma che combina il codice oggetto generato dal compilatore con le librerie e altre risorse necessarie per creare un programma eseguibile completo. Il linker collega il codice del programma con funzioni e risorse esterne, risolvendo riferimenti a simboli (come variabili e funzioni) non definiti nel codice sorgente, ma richiesti per l'esecuzione.

3. Programmazione Strutturata

- **Programmazione Strutturata**: Paradigma di programmazione che prevede l'utilizzo di tre strutture di controllo fondamentali: **sequenza**, **selezione** e **iterazione**.
- **Teorema di Böhm-Jacopini**: Teorema che dimostra che qualsiasi programma esprimibile tramite istruzioni di salto (**goto**) può essere riscritto utilizzando solo le tre strutture di controllo fondamentali.
- **Sequenza**: Struttura di controllo che prevede l'esecuzione delle istruzioni nell'ordine in cui compaiono nel codice.
- **Selezione**: Struttura di controllo che permette di scegliere tra due o più alternative in base al valore di una condizione logica.
- **Iterazione**: Struttura di controllo che permette di ripetere un blocco di istruzioni un numero definito o indefinito di volte.

4. Sintassi e Semantica dei Linguaggi

- **Sintassi**: Insieme di regole grammaticali che definiscono come scrivere correttamente le frasi di un linguaggio di programmazione.
- **Semantica**: Significato delle frasi di un linguaggio di programmazione, ovvero cosa l'esecuzione di un programma produce.
- **BNF (Backus-Naur Form)**: Formalismo per descrivere la sintassi di un linguaggio di programmazione, rappresentando simboli terminali e non terminali, e generando frasi sintatticamente corrette.
- **Grafi Sintattici**: Formalismo grafico alternativo al **BNF** per rappresentare la struttura sintattica, dove i rettangoli indicano simboli non terminali e gli ovali simboli terminali.

5. Il Linguaggio C

- **C**: Linguaggio di programmazione ad alto livello che coniuga efficienza, portabilità e flessibilità.

- **Alfabeto e parole riservate**: In **C**, l'alfabeto è **Unicode**, con parole riservate che non possono essere usate con altri significati (es. `if`, `int`, `float`).
- **Preprocessing**: Fase di elaborazione che precede la compilazione, in cui vengono gestite direttive come `#include` e `#define`.
- **Direttive di Preprocessing**: Istruzioni speciali che forniscono indicazioni al preprocessore. Le principali direttive sono:
    - **`#include`**: Direttiva che permette di includere il contenuto di un altro file nel codice sorgente.
    - **`#define`**: Direttiva che permette di definire macro, ovvero associare un nome a un valore o a un frammento di codice.
- **Funzioni**: Blocchi di codice riutilizzabili che permettono di suddividere un programma in moduli.
- **Commenti**: Porzioni di codice ignorate dal compilatore, utilizzate per documentare e spiegare il codice.
- **Operatori**: Simboli che permettono di eseguire operazioni su dati.
- **Side Effect**: Effetto collaterale di un'operazione che modifica lo stato del programma, come ad esempio l'assegnamento di un valore a una variabile.

6. Caratteristiche del C

- **Efficienza**: Derivazione diretta dagli scopi per cui nasce, offrendo grande velocità di esecuzione e un utilizzo ottimale delle risorse.
- **Portabilità**: La disponibilità di compilatori per tutte le piattaforme e la standardizzazione del linguaggio ne garantiscono l'utilizzo su diverse macchine.
- **Flessibilità**: **C** è stato usato per programmare un'ampia gamma di applicazioni, grazie alla sua diffusione tra i programmatori e alla possibilità di compiere operazioni impensabili con altri linguaggi.
- **Semplicità Sintattica**: Linguaggio con sintassi essenziale e limitato numero di istruzioni, che si avvale in gran parte di funzioni di libreria.
- **Accesso diretto alla memoria**: Possibilità di accedere alla memoria del computer, una caratteristica essenziale per lo sviluppo di sistemi operativi.

7. Limiti del C

- **Manutenzione**: La mancanza di strumenti per una suddivisione razionale del codice rende la manutenzione complessa, soprattutto in progetti di grandi dimensioni.
- **Scarsa Leggibilità**: **C** permette di scrivere codice estremamente conciso, ma anche poco comprensibile, rendendo difficile la collaborazione tra diversi sviluppatori.
- **Error Proneness**: La flessibilità implica un rischio maggiore di errori dovuto alla mancanza di controlli di consistenza rigorosi, frequenti in altri linguaggi.
- **Obfuscation**: La sintassi di **C**, se usata in modo complesso, può generare codice di difficile lettura, con comandi e operatori combinabili in modo denso e poco trasparente.