Ordine degli argomenti:

1. il compilatore;
2. la funzione *printf(...)* ;
3. il costrutto di controllo *if* ;
4. le espressioni logiche.

(in questa lezione non faremo gli escape characters)
La scorsa volta, abbiamo visto come è possibile un programma che assegni il risultato di un'espressione ad una variabile di qualsiasi tipo e come dobbiamo fare attenzione al valore e al tipo di ogni espressione e sotto-espressione affinché il C non faccia interagire i dati in modo sbagliato.
Fino a questa lezione, vi sarà sicuramente sorta la domanda: "Come foss... come fa il computer a capire ed eseguire ciò che scrivo quasi in inglese?"

Il compilatore

Beh, ovviamente il computer non può leggere int a = 5;, poiché questo per lui non ha senso. È proprio questo il ruolo del compilatore, dare un senso a ciò che scriviamo in C. Dopo tutto bisogna anche ricordare che il C è universale, mentre ogni tipo di processore esegue istruzioni scritte in modo leggermente diverso!
Il compilatore è un programma ".exe", ovvero un eseguibile: "gcc.exe".
Possiamo eseguirlo chiamandolo dal prompt dei comanti (cmd su Start) scrivendo gcc. Se durante l''installazione del compilatore avete inserito il suo percorso nelle variabili d'ambiente del sistema, il prompt dei compandi riconoscerà il programma e lo lancerà (e fallirà miseramente perché non ha nessun file da compilare).
Per far compilare un file al compilatore possiamo passarlo come argomento a gcc scrivendo gcc nomefile, se il file si trova nella stessa cartella o gcc C:\Users\User\Desktop\nomefile se il file si trova sul desktop, quindi in poche parole dobbiamo inserire il percorso del file che troviamo nelle sue proprietà o nella barra in alto di Esplora Risorse.
Nel caso per comodità volessimo cambiare cartella nel cmd (che viene visualizzata a sinistra ad ogni nuovo comando), possiamo utilizzare il comando cd(change directory), e inserire il path(percorso) della cartella, quindi cd C:\Users\User\Desktop, oppure cd Desktop se la cartella Desktop si trova nella cartella in cui ci troviamo (cosa che capita spesso dato che il path di default è C:\Users\User).

Una volta eseguito gcc sul nostro file ".c" succede qualcosa di magico, ovvero gcc apre il nostro programma, scritto in caratteri ASCII, li legge, uno alla volta, cerca di capire il significato id ciò che abbiamo scritto, controlla se ci sono degli errori di sintassi, per esempio un ; mancato alla fine di un'istruzione, delle " che non sono state chiuse, variabili non dichiarate, roba scritta male ecc..
Finita questa fase, vuol dire che il nostro programma è quantomeno comprensibile dal computer e traducibile. A questo punto viene trasformato in ASSEMBLY, che cos'è?
Non ci interessa, sappiate che è un linguaggio molto più brutto e complicato dove int a = 5+3; corrisponde a 5 righe diverse e ogni riga può contenere informazioni sulla posizione in memoria della variabile, istruzioni su come trattare l'espressione, cosa fare prima, cosa fare dopo, i cast, l'addizione, l'allocazione e tutta quella roba.
Questo linguaggio, per la cronaca, è scritto a lettere, quindi è sempre in ASCII, ma non ha senso impararlo se non si studiano i componenti del pc, perché?
Perché ad ogni istruzione di questo linguaggio, corrispondono esattamente 32 o 64 bit (a seconda del vostro computer) e ogni istruzione corrisponde a un ciclo del processore, ovvero?
Avete mai sentito dire "Processore a 3.4 GHz (giga heartz)"? Bene, bene, questo vuol dire

che il vostro processore esegue 3.4 miliardi di quelle istruzioni al secondo (più o meno).

Quindi:
scriviamo
gcc programma.c -o programma
dove -o serve a specificare il nome dell'output che sarà programma.exe, il nostro programma.c;

```
int main()
{
   int a = 5;
}
```

lungo 32 caratteri (inclusi gli spazi) e quindi grande 32byte (ogni carattere ricordiamo è un byte), verrà trasformato in un file eseguibile molto più grande e molto più lungo, che se proviamo ad aprire con il blocco note non riusciremo a leggere (per provare vi basta trascinare il file in una nuova finestra del blocco note) grande anche qualche mega.
Perché non possiamo leggerlo?
Il file, essendo un file eseguibile, è scritto in bit, e ogni 64 bit rappresenta un'istruzione da eseguire, quindi, quando il blocco note cerca di aprirlo, prova a leggere 8 bit alla volta (1 byte) e ci mostra a che carattere corrispondono, quando in realtà essi fanno parte di un'istruzione più grande con tutto un altro significato!
Piccola parentesi, vorrei puntualizzare che i caratteri del linguaggio assembly, non vengono tradotti nel loro corrispettivo binario dell'ascii, ma ad ogni istruzione corrisponde una serie di bit che indica quell'istruzione.
Per eseguire il nostro programma non ci resterà che chiamarlo come abbiamo fatto per gcc, ovvero:
programma.exe
o anche
programma
sul prompt dei comandi.
Se proviamo ad aprirlo con doppio click, il nostro programma si aprirà e si chiuderà e non faremo in tempo neanche a vederlo.

La funzione *printf(...)*

Fino ad ora abbiamo visto come maneggiare la memoria del nostro computer, come fargli fare calcoli attraverso le espressioni e come far interagire le variabili, ma non abbiamo ancora visto come ottenere neanche un output!
A questo serve la funzione *printf(...)*: print formatted
Cos'è?
La funzione printf è... una funzione, ovvero? Lo vedremo più in là, per ora vi basti sapere, che prende in input delle espressioni chiamate parametri e che ha l'effetto di scrivere qualcosa sul nostro prompt dei comandi.
Ecco come si usa:

```
printf("Ciaoooo");
```

Quindi, printf, poi le parentesi e subito dopo, la prima cosa che va messa è ciò che vogliamo scrivere tra virgolette, poi il ; alla fine
Questa riga, quando verrà eseguita, ci stamperà, sotto la riga dove abbiamo chiamato il programma nel prompt dei comandi, "Ciaoooo" (senza ").
E se volessimo stampare il contenuto di una variabile? Vi ho detto che la prima cosa che va messa nella printf sono le virgolette, quindi scrivere printf(variabile) è sbagliato!
Si usano i cosiddetti *placeholder*, cosa sono?
Sono per l'appunto, dei contenitori che servono ad indicare la posizione di una variabile da

stampare nella nostra stringa tra ":
int a = 10, b = 5;
printf("%d %d", a, b);
Ecco come si usano, si scrivono all'interno delle " insieme ad altro testo, anche attaccato (ciao%dciao) e indicano la posizione dove ogni variabile, in ordine di scrittura dopo la prima virgola, viene stampata.
Ora, come ho già detto tante volte, il tipo di variabile è molto importante, poiché da esso dipende la quantità di memoria che occupa e il significato dell'informazione che rappresenta.
Per questo, esiste un placeholder diverso, quasi per ogni tipo di variabile:

Placeholder	*Descrizione e utilizzo*
%d	d sta per decimal, si utilizza per le variabili di tipo int
%i	i sta per intero, sempre per gli int, ma la base numerica potrebbe cambiare. qui trovate più informazioni https://www.geeksforgeeks.org/difference-d-format-specifier-c-language/ In ogni caso usate d e state apposto
%f	f si usa per i float e per i double (attenzione nella scanf che vedremo più in là è necessario usare lf per i double)
%ld e %h	si usano rispettivamente per i long int e gli short int perché? perché occupano quantità di memoria differenti dagli int
%c	c sta per char e si usa per le variabili di tipo char

Ci sono sono poi altri modificatori che non ci interessano per esempio è possibile scegliere il numero di cifre minimo da stampare, se vi va di vedere come funzioano andate sul primo tutorial che trovate su YouTube sui placeholder in C :)

Bene, abbiamo finalmente imparato come darte un minimo di vita al nostro programma e vedere come funziona un po' più da vicino.
Ne approfitto per dirvi che quando un programma non vi funziona, scrivere molte printf sulle variabili usate è utile per vedere cosa succede al suo interno e localizzare l'errore. Questo modo di fare viene chiamato debug.
Se qualcuno fosse sorta la domanda: "E se volessi mettere delle " nella mia stringa da stampare senza chiuderla?": lo vediamo la prossima volta!

Il costrutto *if*

Ed eccoci qui, al nostro primo costrutto di controllo!
Cos'è un costrutto di controllo?
È un... qualcosa... che ci permette di controllare il flusso del nostro programma, come i rombi che avevamo visto nei diagrammi di flusso!
L'if ci permette di fare esattamente questo: verificare una condizione e decidere in base a questa se eseguire un blocco di codice o un altro.
Esso si costruisce in questo modo:
if(espressione)
{

```
   codice
   a=5;
   b=11+a;
   printf("la condizione e' vera, e comunque b e' %d", b);
}
```
Ora, è importante ricordare, che in C, l'if non valuta il tipo dell'espressione, ma solo il suo valore e, se il valore è 0 (quindi 0, 0.0, o il carattere che rappresentato dal primo numero della tabella ascii (che non è '0')), il blocco di codice nell'if NON viene eseguito, se il valore è DIVERSO da 0 (quindi 1, -12, 24.124, 'n'), allora il blocco di codice viene eseguito.
Eseguito l'if, il programma prosegue dopo le parentesi graffe indipendentemente dalla condizione. (le graffe si fanno con SHIFT+ALT Gr+è, +)
Fin qui, credo sia chiaro.
All'if, possiamo aggiungere una bella parolina chiamata else, cos'è?
Un altro blocco di codice che viene eseguito solo se la condizione dell'if è falsa, ovvero se è l'epressione è 0:
```
if(0)
{
   printf("Questo non verrà stampato");
}
else
{
   printf("La condizione è falsa");
}
```
Ora, è ovvio che scrivere numeri arbitrari all'interno della condizione non ha senso, a meno che non vogliate forzare la condizione.
All'interno della condizione va una qualsiasi espressione, quindi, in teoria, potremmo scriverci una variabile per controllare che non sia 0, o una somma di variabili.
Piuttosto scomodo vero, non possiamo verificare nulla così.
Per questo ci vengono in aiuto le ESPRESSIONI LOGICHE, con i loro OPERATORI LOGICI.

Le espressioni logiche

Ora, non sto qui a spiegarvi la logica, la faremo la prossima volta, MA, vedremo almeno gli operatori logici aritmetici, ovvero? > < >= <= == !=
Questi sono operatori BINARI, che prendono due valori e ne tirano fuori un altro, come funzionano?
Vi basterà vedere questo esempio:
```
a = 5 > 2;
/* a sarà uguale a 1 */
a = 5 < 5;
/* a sarà uguale a 0 poiché 5 non è minore di 5*/
a = 5 >= 5;
/* a sarà uguale a 1 poiché 5 è maggiore O UGUALE a 5 */
a = 5 == 5;
/* a sarà uguale a 1 poiché 5 è uguale a 5*/
/*attenzione, scrivere a = b = 5 non verifica che b sia 5, ma assegna 5 a b e poi assegna b ad a, per verificare l'uguaglianza dobbiamo sempre usare il ==*/
a = 5 != 5;
/* a sarà 0 perché 5 non è DIVERSO da 5, mentre a = 5 != 4; avrà 1 assegnato dopo la valutazione dell'espressione*/
```
Questo modo di fare può essere usato all'interno degli if:
```
int a = b = 5;
if(a != b)
{
   printf("a e' diverso da b, e comunque metto l'apice perche' la e accentata non sta nella
```

```
tabella ascii e quindi a volte si stampa male");
} else
{
   printf("a e' uguale a b, e comunque sono le 5:00:31");
}
printf("Questa frase viene stampata in ogni caso, a meno che non facciate cose sbagliate tipo provare ad accedere a memoria che non vi appartiene o mandare il programma in loop");
printf("Suggerimento, se un programma vi si blocca, premete CTRL+C durante la sua esecuzione per arrestarlo senza chiudere il cmd, funziona anche con gli altri!");
```

Ultima cosa, IMPORTANTISSIMA
Il C, non tiente conto degli spazi che usate, ovvero, voi potreste letteralmente scrivere tutto il programma su una sola riga, finché mettete in ; alla fine di ogni istruzione (rettangolo) non ci sono problemi, ma, ovviamente, scrivere il programma con la giusta INDENTAZIONE che è cosa buona e giusta di chi sa programmare, aiuta a renderlo molto più leggibile e comprensibile.
Per questo vi consiglio VIVAMENTE di aggiungere un TAB ogni volta che aprite una nuova parentesi graffa, di mettere gli spazi dopo le , , di mettere gli spazi quando usate operatori logici e se volete anche quelli aritmetici.
Inoltre, avrete già visto la presenza di /*asdasfas*/ questi sono commenti, non vengono trattati in fase di compilazione, vengono completamente ignorati e non influenzano in nessun modo il programma, è molto importante usarli (cosa che non faccio perché sono pigro) perché un programma che fate oggi e riaprite tra una settimana non sapete come funziona, né tantomeno lo sa qualcuno a cui lo mandate, perciò, fate dei commenti, spiegate ciò che volete fare e come lo fate!

Tipo mo devo riaprire un programma assurdo che ho fatto in aprile, ce li ho anche messi i commenti, ma secondo voi so dove mettere mano? No. Non so neanche come faccia a funzionare...
Vabbè notte.