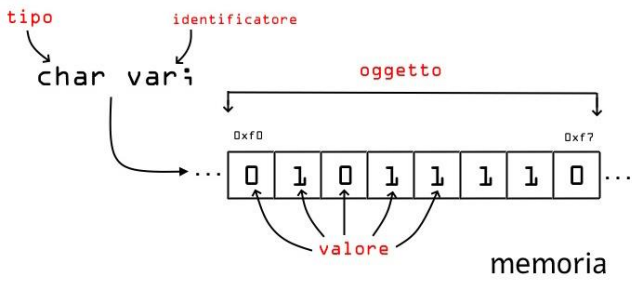
Le Variabili

Possiamo considerare le variabili come dei contenitori di dati, queste scatole ci permettono di leggere, elaborare e salvare i valori che utilizziamo all'interno dei nostri programmi.

In C++ le variabili sono composte da 4 componenti fondamentali:

- Il tipo (`char`) che definisce l'insieme dei valori possibili e delle operazioni effettuabili sulla variabile.
- L' identificatore (`var`) che identifica l'oggetto all'interno del programma.
- L'oggetto che corrisponde alla memoria che contiene il valore della variabile.
- Il valore, cioè l'insieme di bit contenuti nell'oggetto che vengono interpretati dal compilatore a seconda del tipo.



Tipi principali in C++

In C++ esistono cinque famiglie principali di tipi, ognuno di essi ha delle caratteristiche precise e delle operazioni apposite.

1. Boolean
2. Character
3. Integer
4. Float
5. Void

Questi tipi costruiscono i tipi di base su cui si costruiscono tutte le altre strutture ed ora ve li presenterò uno ad uno.

Boolean

```
bool b;
```

Il tipo boolean ha la dimensione di 1 bit e può assumere solamente i valori `true` e `false`. Questo tipo viene utilizzato per esprimere e gestire i risultati delle operazioni logiche che tratteremo successivamente. Un esempio di utilizzo può essere il seguente:

```
int a = 1;

int b = 2;

int c = 1;

bool var = { a == b };              //var = false

bool var = { a == c };              //var = true
```

Per definizione in valore intero `true` corrisponde a `1` mentre `false` corrisponde a `0`. Al contrario, cioè spostandoci da valori interi a valori booleani, tutti i valori interi diversi da `0` corrispondono a `true` mentre `0` corrisponde a `false`.

La dimensione di una variabile bool è esattamente **1 bit**.

Character

La famiglia dei tipi character si occupa di gestire per l'appunto i simboli che fanno parte di uno o più alfabeti. I tipi che compongono questa famiglia sono i seguenti:

char

```
char c;
```

`char` è il tipo carattere di default utilizzato per gestire i testi. La sua dimensione è di almeno **8bit** (1 byte) e quindi di **256 combinazioni** che corrispondono ognuna ai caratteri dell'alfabeto ASCII (esteso), i caratteri

Rispetto a questo alfabeto possiamo essere sicuri delle seguenti assunzioni:

- Contiene i 26 caratteri appartenenti all'alfabeto (comprese le lettere straniere in versione minuscola e maiuscola)
- Contiene la punteggiatura più comune

Non sono sicure invece le seguenti affermazioni:

- Non sono presenti più di 127 caratteri. (Alcuni alfabeti ne prevedono 255 come ad esempio l'extended ASCII)
- Non ci sono più caratteri dell'alfabeto. (alcuni alfabeti prevedono le lettere specifiche come l'alfabeto polacco)
- I caratteri alfabetici sono l'uno successivo all'altro.
- I caratteri utilizzati per scrivere in C++ sono tutti presenti. (in alcuni set mancano {, }, [,], | e \)
- `char` è di un byte da 8-bit. (Alcuni utilizzano 16 bit altri 32bit)

Altri tipi di char

Oltre al tipo `char` esistono anche i seguenti "sottotipi":

- `signed char`: in cui è garantito che possa contenere sia valori positivi che valori negativi [-127, 127].
- `unsigned char`: che garantisce solamente valori positivi [0, 255].
- `wchar_t`: garantisce di poter contenere il più grande simbolo conosciuto nell'alfabeto UNICODE (è grande).
- `char16_t`: garantisce di poter contenere un alfabeto di 16bit.
- `char32_t`: garantisce di poter contenere un alfabeto di 32bit.

Davanti a queste ultime definizioni possiamo dire che il tipo char è di tipo **unsigned char**.

Se siete curiosi di sapere il valore intero dei caratteri presenti nel vostro alfabeto vi lascio il codice per visualizzarli in output:

```
char c;
cout << "Il valore di" << c << " è di " <<  (int)c << '\n';
```

Character literals

Un character literal è un singolo carattere inserito in mezzo a due apici singoli (') ad esempio `'a'` oppure `'0'` ed il tipo di questi simboli è per l'appunto il tipo `char`. Ognuno di questi caratteri può essere convertito nel valore intero in cui è definito sulla macchina in cui si sta eseguendo il codice, per esempio comunemente il valore intero di '0' è 48.

Esistono alcuni caratteri speciali, riconoscibili perché preceduti da `'\'` (backslash), che hanno dei comportamenti speciali

Nome	Name	Literal in C++	ASCII	Console
Nuova linea	New line	`'\n'`	NL(LF)	Va a capo
Tab orizzontale	Horizontal tab	`'\t'`	HT	Immette tre spazi
Tab verticale	Vertical tab	`'\v'`	VT	Si sposta sulla riga sottostante alla stessa distanza
Spazio precedente	Backspace	`'\b'`	BS	Elimina l'ultimo carattere
Carriage return	Carriage return	`'\r'`	CR	La controparte di /n per Mac
Form feed*	Form feed	`'\f'*`	FF	Va all'inizio della pagina successiva
Alert*	Alert	`'\a'*`	BEL	Serve ad emettere un segnale (beep)
Backslash	Backslash	`'\\'`	\	Stampa \
Punto di domanda	Question mark	`'\?'`	?	Stampa ?
Apice singolo	Single quote	`'\''`	'	Stampa '
Doppio apice	Double quote	`'\"'`	"	Stampa "

*: literal praticamente inutilizzati nel mondo moderno.

La particolarità di questi caratteri è che sono stati pensati le stampanti degli anni 70: ai tempi non c'erano le stampe automatiche come oggi ed era necessario inviare manualmente i comandi alla macchina per eseguire le operazioni di scrittura.

Inoltre è bene sapere che si possono rappresentare i caratteri in ottale, decimale ed esadecimale ottenendo sempre lo stesso risultato in ASCII (utilizzando i cout ad esempio).

Ottale	Decimale	Esadecimale	ASCII
`'\60'`	48	`'\x30'`	`'0'`
`'\137'`	95	`'\x05f'`	`'_'`

Integer

Questa famiglia di tipi permette di salvare ed operare con i numeri interi cioè numeri positivi e negativi senza cifre decimali.

Int

```
int i;
```

`Int` è il tipo intero principale **solitamente** è grande **32 bit** (4 byte) e quindi può contenere 2^{32} combinazioni i numeri normalmente nel range $[-2^{16},\ 2^{16}]$ e questo lo rende `signed` di default.

Perché deve poter contenere sia numeri positivi che negativi.

Altri tipi di int

Oltre al tipo `int` esistono anche i seguenti tipi:

- `signed int` : garantisce un range di valori compreso in $[-2^{31}+1,\ 2^{31}-1]$
- `unsigned int` : garantisce un range di valori compreso in $[0,\ 2^{32}-1]$
- `short int` : solitamente utilizzato per numeri piccoli normalmente gestisce **16bit** di memoria e può contenere numeri negativi (si può utilizzare unsigned come prefisso).
- `long int` : solitamente utilizzato per numeri piccoli normalmente gestisce **64bit** di memoria e può contenere numeri negativi.
- `long long int` : serve per gestire numeri **più grandi di 64bit**;

Il tipo di int corrisponde al tipo `signed int`, perché può contenere valori positivi e negativi.

Integer literals

Gli integer literals danno la possibilità al programmatore di rappresentare i numeri interi in forma ottale, decimale e esadecimale.

Ottale	Decimale	Esadecimale
`0`	-	`0x0`
`02`	`2`	`0x2`
`083`	`123`	`0x53`

A quanto pare ragazzi il numero intero 0 che scriviamo in C++ è in notazione ottale non decimale.

Floating-Point

Questa famiglia di tipi serve a rappresentare i numeri reali, cioè che hanno parte decimale.

Float

```
float f;
```

`float` è il tipo reale di riferimento, solitamente ha una grandezza di **32 bit** (4 byte) e può contenere sia valori positivi che negativi.

Altri tipi di float

- `double`: definito "float a doppia precisione" a 64 bit.
- `long double`: definito "float a precisione estesa" a 128bit.

Attenzione: questa famiglia di tipi è differente da quella di tipo integer perché per codificare i numeri decimali utilizza la codifica in virgola mobile: calcola i valori utilizzando la mantissa, l'esponente e il segno prendendo spunto dalla notazione scientifica. La differenza principale è che **non associa ad ogni combinazione un numero** ma li calcola a seconda dei bit utilizzati.

Per approfondire l'argomento:

https://www.youtube.com/watch?v=qi2F9JsWrGo

Void

Void significa "vuoto", proprio per questo motivo il tipo void non ha alcun oggetto corrispondente in memoria. Mediamente viene utilizzato per non ritornare alcun risultato nelle strutture chiamate **funzioni**.

Non possiamo definire delle variabili di tipo void perché in questo caso il compilatore ci segnalerà un errore.

Dichiarazione di una variabile

In C++ possiamo *dichiarare* una variabile nel nostro programma nel seguente modo:

```
int var;
```

Dove in corrisponde al `tipo` della variabile e `var` corrisponde al suo identificatore

La dichiarazione delle variabili serve ad indicare al calcolatore che i dati che verranno utilizzati, permettendo al compilatore di indicare gli errori logici che compromettono l'integrità del programma. Inoltre, a seconda del tipo utilizzato, il computer esegue una serie di ottimizzazioni nell'utilizzo della memoria riducendo l'utilizzo di risorse e garantendo al programma le prestazioni adeguate.

Inoltre, è possibile dichiarare più variabili sulla stessa linea come negli esempi seguenti:

```
int i, i1, i2, i3;
```

```
float f1, f2, f3;
```

Variabili e identificatori

Dare un nome alle variabili, in realtà agli oggetti in generale, è una vera e propria arte soprattutto quando programmiamo delle grandi applicazioni.

In generale vi consiglio di tenere a mente queste semplici regole:

1. Fate sempre in modo che il nome della variabile sia vicino alla sua funzione all'interno del programma.
2. Cercate di non utilizzare nomi troppo lunghi, la sintesi è sempre la strada migliore.
3. Definite i nomi delle variabili in modo che la prima lettera sia sempre minuscola, il linguaggio non vieta di utilizzare le maiuscole ma in generale è meglio evitare.
4. Per i nomi composti è meglio `variabile_nome_composto` piuttosto che `variabileNomeComposto`
5. Non iniziate gli identificatori con '_' (per esempio: `_var`) mediamente questi nomi sono riservati a degli oggetti utilizzati dal linguaggio sottoscocca.

Inizializzazione di una variabile

Per poter utilizzare una variabile, dopo averla adeguatamente dichiarata, dovremo eseguire la sua *inizializzazione*, cioè dovremo assegnarle un valore così che il calcolatore possa effettivamente ricavarle uno spazio all'interno della memoria dinamica (RAM), in questo modo il compilatore assegnerà un valore da noi scelto all'oggetto in memoria corrispondente alla variabile.

La sintassi è la seguente:

```
var = 0;
```

La cosa importante da ricordare in questa fase è di non utilizzare la variabile senza che sia inizializzata, in questo caso il linguaggio C++ non segnalerà un errore ma ci permetterà di lavorare con delle variabili di cui non sappiamo il valore effettivo dando così spazio a dei comportamenti imprevedibili del programma.

In classe utilizzare una variabile senza che sia inizializzata verrà considerato errore.

Assegnazione di una variabile

L'operazione che ci permette di inizializzare una variabile si chiama *assegnazione* e può essere eseguita in qualsiasi punto del programma (ovviamente dopo la dichiarazione), tramite questa operazione (attuata tramite l'operatore =) possiamo scrivere o sovrascrivere una variabile con un nuovo valore.

L'inizializzazione è a tutti gli effetti un'assegnazione ma dividiamo i due concetti per marcarne la funzione.

Possiamo assegnare ad una variabile sia un valore statico, `var = 0`, che un valore dinamico, `var = var1` dove `var1` è un'altra variabile.

Costanti

Le costanti sono dei particolari tipi di variabile che principalmente vengono impiegate quando si vuole essere sicuri che il loro valore rimanga invariato durante tutto il ciclo di vita del programma.

Oltre alla loro affidabilità bisogna tenere a mente che le costanti sono un anche un vantaggio per le prestazioni del programma.

Per utilizzare le costanti si deve impiegare il prefisso `const` nella dichiarazione della variabile come nell'esempio seguente:

```
const float PI = 3.14159;
const char LETTERA_A = 'A';
```

Per convenzione è meglio scrivere l'identificatore delle costanti in maiuscolo in modo da poterle riconoscerle a colpo d'occhio all'interno del programma.

Esercizi di comprensione

1. Verificare la grandezza in byte dei tipi fondamentali int, float, char, boolean

   Per questo esercizio vi consiglio l'utilizzo dell'operatore sizeof

   https://en.cppreference.com/w/cpp/language/sizeof

2. Verificare i valori massimi e minimi inseribili nei seguenti tipi di variabili in bit
   a. Int
   b. Unsigned int
   c. Short int
   d. Unsigned short int
   e. Long long int
   f. Unsigned long int
   g. Char
   h. Signed char
   i. Unsigned char

   Per verificare queste dimensioni dovete utilizzare la libreria esterna limits.h utilizzando la dicitura #include <climits> all'inizio del codice.

   Vi lascio la documentazione che elenca le costanti dichiarate da questa libreria:

   https://en.cppreference.com/w/c/types/limits