02 - Recap

Bailetti Tommaso

ITI Don Orione

3 Febbraio 2022

Le variabili

Le variabili sono il nostro modo di inserire all'interno del nostro programma dei "valori". Al momento dell'esecuzione vengono allocati in memoria i `byte` necessari per tutte le nostre variabili.

Tipo di variabili

Il programmatore necessita di sapere cos'è il **tipo** di una variabile. Il tipo non è che la definizione di **cosa** stiamo salvando all'interno della variabile. Ci sono differenti tipi di variabili in C++.

Tipo di variabili

Tipo	Descrizione	Dimensione (byte)	Intervallo numerico
`char`	Singolo carattere numero intero *small*	1	`signed`: da –128 a 127 `unsigned`: da 0 a 255
`short int`	Numero intero *short*	2	`signed`: da –32 768 a 32 767 `unsigned`: da 0 a 65 535
`int`	Numero intero		Dipendente dal compilatore
`long int`	Numero intero *long*	4	`signed`: da –2 147 483 648 a 2 147 483 647 `unsigned`: da 0 a 4 294 967 295
`bool`	Valore booleano	1	`true`/`false`
`float`	Numero *floating-point*	4	da -3.4×10^{-38} a 3.4×10^{38}
`double`	Numero *floating-point* a doppia precisione	8	da -1.7×10^{-38} a 1.7×10^{38}

Ulteriori keyword per le variabili

Per le variabili sono presenti due ulteriori keyword che permettono di modificare le definizioni delle nostre variabili:

`signed/unsigned` Ci permette di definire se la nostra variabile deve mantenere il segno o meno.

`const` Ci permette di definire e inizializzare una variabile il quale valore non è mai modificato durante l'esecuzione.

Quindi in sostanza, per dichiarare una variabile

const *unsigned* <tipo> <nomeVariabile>;

dove le keyword tra gli asterischi sono opzionali, a seguire degli esempi

```
1  const float pi = 3.14;
2  unsigned int numberOfResets = 0;
3  char yesChar = 'Y';
```

Scope and Lifetime

Ora che sappiamo cosa sono le variabili, è necessario approfondire cosa sono lo "scope" e il "lifetime" di una variabile.

Scope

L'**ambito di visibilità** o `scope` è il blocco di istruzioni dove la nostra variabile è dichiarata. Al di fuori di essa la nostra variabile di fatto non esiste più. Le variabili possono quindi essere:

Variabili Globali È valida dall'inizio dell'esecuzione del software fino al termine. Una variabile è di fatto globale solo se è definita al di fuori di un blocco di istruzioni.

Variabili Locali Sono valide solo all'interno del proprio blocco e non è possibile accedervi al di fuori di esso.

Lifetime

Il **tempo di vita** di una variabile o `lifetime` è il tempo per la quale la nostra variabile è presente all'interno del programma. Di fatto dipende da che `scope` diamo alla nostra variabile.

Variabile Permanente *una variabile globale è sempre permanente*, si crea all'inizio dell'esecuzione e rimane per tutta la durata dell'esecuzione.

Varaibile Temporanea *è allocata solamente all'inizio dell'esecuzione del blocco che ne comprende la dichiarazione*. Lo spazio in memoria, una volta eseguito il blocco, viene liberato dalla variabile.

Matematica in C++

Operatore	Descrizione
+	Somma o segno
-	Sottrazione o segno
*	Moltiplicazione
/	Divisione
%	Modulo (resto della divisione)

Un consiglio che vi dò, è quello di mettere sempre tra parentesi i calcoli nell'ordine in cui volete siano eseguiti. C++ usa le priorità "normali" della matematica (prima le moltiplicazioni e divisioni, poi le somme e sottrazioni) tuttavia prendiamo l'abitudine di non fidarci, può succedere che cambiando linguaggio di programmazione non sia più così.

Operatori rapidi

Va menzionata anche la presenza di operatori matematici "compatti", i quali sono:

Versione Compatta	Versione Estesa
`a += b`	`a = a + b`
`a -= b`	`a = a - b`
`a *= b`	`a = a * b`
`a /= b`	`a = a / b`

Incrementi

C++ mette a disposizione la possibilità di incrementare e decrementare le nostre variabili matematiche con la seguente sintassi:

- `<nomeVariabile>++` o `++<nomeVariabile>`
- `<nomeVariabile>--` o `--<nomeVariabile>`

di fatto questa cosa si traduce in:

- `<nomeVariabile> = <nomeVariabile> + 1`
- `<nomeVariabile> = <nomeVariabile> - 1`

Attenzione

con la sintassi `++<nomeVariabile>` e `--<nomeVariabile>` **l'operazione matematica avviene prima della interpretazione**.

Logica in C++

A questo punto dobbiamo imparare come far svolgere condizionalmente delle operazioni al nostro programma. Si possono effettuare comparazioni tra variabili e costanti con la seguente sintassi:

```
a <operatore> b
```

Operatore	Descrizione
==	Uguale
<	Minore
<=	Minore o uguale
>	Maggiore
>=	Maggiore o uguale
!=	Diverso

Nota bene

Tutte le comparazioni ritornano `true` o `false`.

Espressioni Logiche

Inoltre le condizioni logiche possono essere combinate in un espressione logica, la quale pone il risultato della nostra condizione attraverso una porta logica.

Operatore	Descrizione
!	NOT (negazione)
&&	AND (congiunzione logica)
||	OR (disgiunzione logica)

Casting

Il `cast` è una conversione fra tipi di variabili. Esistono due tipi di `casting` e sono condizionati dalla perdita o no dei dati durante il `cast`:

casting implicito Il casting implicito avviene quando passo da un tipo di dato "con meno informazioni" a uno "con più informazioni". Avviene automaticamente e non dobbiamo fare nulla.

casting esplicito Il casting esplicito invece avviene quando, durante un `cast`, avviene perdita di informazioni. Il `casting` esplicito **deve essere specificato quando scriviamo il programma**.

Casting esplicito

Il casting esplicito avviene scrivendo la seguente forma:

```
1  float pi = 3.14;
2  int i = (int)pi;
3  int i = int(pi);
```

Output e Input

La libreria `stdinput`, che usiamo in ogni programma, ci permette di scrivere a console e ricevere input da tastiera. Infatti a ogni programma che noi scriviamo, troviamo in alto l'import della libreria con `#include "<iostream>"` Questo ci permette di utilizzare `cout` e `cin` che comportano rispettivamente l'output e l'input del nostro programma.

Output

Per effettuare output nella nostra console si utilizza il comando `cout` come segue:

```
1  cout << "Questo è un output!";
2  cout << "Questo è un " << "output " << "composito!";
3  int i = 3;
4  cout << "Questo è l' " << "output " << " n°" << --i;
```

Caratteri speciali in output

Ci sono dei caratteri speciali in output che modificano il comportamento della nostra console, ritornando a capo, aggiungendo un tab, ecc...

Simbolo	Descrizione
`/r`	Singolarmente o in coppia definiscono il ritorno a capo e/o l'invio
`/n`	
`/t`	Tabulazione
`//`	Carattere *backslash*
`/"`	Carattere «"» all'interno di una sequenza di caratteri delimitata da simboli «"»

È conveniente per il programmatore scrivere il ritorno a capo con `endl`.

Input

L'input in un programma è svolto tramite il comando `cin` e la sintassi è:

```
1  int i = 0;
2  cout << "Inserire il numero di patate comprate: ";
3  cin >> i;
```

Attenzione

Non è possibile concatenare una richiesta input come si fa con l'output.

Concetti del flusso di esecuzione

Al momento noi abbiamo già visto diversi modi di eseguire il nostro codice, il primo tra tutti è quello in **sequenza**. Di fatto noi inseriamo i comandi fino a quando questi non finiscono.

La selezione

La selezione è l'esecuzione del codice quando viene soddisfatta una determinata condizione. In poche parole, utilizzato gli `if`, siamo in grado di condizionare l'esecuzione del nostro codice.

```
1  // Piccolo programma che se feature è 0, sappiamo che
   ↪  qualcosa è disattivato
2  const int feature = 0;
3  if (feature == 0) {
4      cout << "Funzionalità disattivata";
5  }
```

La selezione, `else` e `else if`

Possiamo anche definire cosa succede nel caso il nostro `if` non sia soddisfatto tramite l'`else`:

```
1  const int feature = 0;
2  if (feature == 0) {
3      cout << "Funzionalità disattivata";
4  } else {
5      cout << "Funzionalità attiva";
6  }
```

La selezione, `else` e `else if`

Possiamo pure mettere in combinazione `if` e `else` per le casistiche più particolari:

```
1  const int feature = 0;
2  if (feature == 0) {
3      cout << "Funzionalità disattivata";
4  } else if (feature == 1) {
5      cout << "Funzionalità attiva";
6  } else {
7      cout << "Codice funzionalità errato";
8  }
```

Casistiche multiple

Ci sono casi dove dobbiamo passare le nostre variabili attraverso una lunga lista di `if`, possiamo rendere il nostro codice più leggibile con lo `switch`.

Casistiche multiple

```
1  char selezione;
2  cin >> selezione;
3  switch (selezione)
4  {
5      case 'A':
6          cout << "Eseguo A...";
7          break;
8      case 'B':
9          cout << "Eseguo B...";
10         break;
11     default:
12         cout << "Selezione invalida!";
13         break;
14 }
```

Operatore ternario

Va citato per referenza anche l'operatore ternario, il quale ci aiuta a rendere più snello il nostro codice. È uno strumento principalmente usato per assegnare delle variabili in maniera condizionale. Senza definire in maniera rigida la sintassi, un esempio è molto più semplice.

Esempio di operatore ternario

```
1  int i = 0;
2  if(<condizione>) {
3      i = <valore>;
4  } else {
5      i = <altroValore>;
6  }
```

```
1  int i = <condizione> ? <valore> : <altroValore>;
```

Le ripetizioni

Sul fronte delle ripetizioni, ci sono tre strutture che ci permettono effettuare cicli del nostro programma.

- `while`
- `do while`
- `for`

while

Il `while` è la struttura ripetitiva più semplice. Basta inserire all'interno una condizione, il ciclo si ripeterà fino a quando la condizione rimane vera durante l'esecuzione. Per entrare dentro il ciclo, la condizione deve essere **inizialmente vera**.

```
1  while (<condizione>) {
2      // Codice da ripetere
3  }
```

Attenzione

Se la condizione rimane sempre uguale all'interno del `while`, **il ciclo eseguirà all'infinito**.

do while

Il `do while` è una struttura che permette di eseguire il primo ciclo del while a prescindere dalla condizione.

```
1 do {
2     // Codice da ripetere
3 } while (<condizione>);
```

Info

Solitamente questo tipo di ciclo si utilizza quando si deve eseguire l'operatione all'interno del ciclo almeno una volta.

for

Il ciclo `for` è il ciclo più usato solitamente, riprende la logica del `while` ma impone all'utente di definire la variabile che viene utilizzata per il ciclo e l'incremento per ogni iterazione.

```
1  for (int i = 0; i < <valore>; i++) {
2      // Codice da ripetere
3  }
```

Interruzione o continuazione di un ciclo

Esistono due keyword che permettono di modificare il comportamento di un ciclo: `break` e `continue`.

`break` all'interno di un ciclo

La keyword `break` è utilizzata per interrompere il nostro ciclo e uscire dallo stesso.

```
1  while (true) {
2      char character;
3      cin >> character;
4      if (character == 'q') {
5          break;
6      }
7  }
```

continue

Continue invece, è utilizzata per passare alla prossima iterazione del nostro ciclo.

```
1  for (int i = 0; i <= 100; i++) {
2      if (i % 2 == 1) {
3          continue;
4      }
5      cout << i << endl;
6  }
```