

Esercizi su

---

Istruzioni di scelta multipla

Overflow

---

# Scelta multipla

# Esercizio 1/2

---

- Scrivere un programma che chieda all'utente di scegliere tra varie opzioni e stampi il nome dell'opzione scelta
- Esempio 1:
  - 1 *Opzione A*
  - 2 *Opzione B*
  - 3 *Opzione C*
  - 4 *Opzione D*

*Scegli un'opzione: 2*  
*Hai scelto l'opzione B*

- Esempio 2:

*1 Opzione A*

*2 Opzione B*

*3 Opzione C*

*4 Opzione D*

*Scegli un opzione: -1*

*Scelta non valida*

- Soluzione in: *primo\_menu.cc*

# Esercizio 1/3

---

- Scrivere un programma che chieda all'utente di scegliere tra le seguenti opzioni (di cui una ripetuta due volte) e stampi il nome dell'opzione scelta

*1 Opzione A*

*3 Opzione B*

*4 Opzione C*

*5 Opzione C*

*6 Opzione D*

*Scegli un'opzione: 4*

*Hai scelto l'opzione C*

- Esempio 2:

*1 Opzione A*

*3 Opzione B*

*4 Opzione C*

*5 Opzione C*

*6 Opzione D*

*Scegli un'opzione: 2*

*Scelta non valida*

# Esercizio 3/3

---

- **Non replicate il codice** per la stampa dei messaggi nei casi 4 e 5
- Soluzione in: *menu\_multiplo.cc*

# Esercizio per casa: calcolatrice

---

- Testo e soluzione in *calcolatrice.cc*



---

# Fasi di sviluppo

# Azioni ed attori

---

- Per descrivere il comportamento di un programma o quello che accade nelle fasi del suo sviluppo si fa riferimento spesso alle azioni che possono essere compiute da uno dei seguenti attori: il **programmatore**, il **compilatore**, il **programma** stesso o l'utilizzatore del programma (**utente**)

# *I tempi di un programma 1/3*

---

- Inoltre tali azioni possono essere compiute in tre diversi momenti dello sviluppo del programma:

**1) A tempo di scrittura del programma, da parte del programmatore**

Es.: il valore iniziale di una variabile o di una costante viene definito a tempo di scrittura del programma da parte del programmatore mediante una **inizializzazione**

# *I tempi di un programma 2/3*

---

**2)A tempo di compilazione del programma, da parte del compilatore**

**Es.: un errore di sintassi viene comunicato dal compilatore appunto durante la compilazione del programma**

# *I tempi di un programma 3/3*

---

**3)A tempo di esecuzione del programma, da parte dell'utente o del programma stesso**

Es.: (su richiesta del programma) l'utente fornisce il valore di una variabile mentre il programma stesso è in esecuzione

---

# Dimensioni dei tipi di dato ed Overflow

# Operatore *sizeof*

---

- Restituisce la *dimensione* di un'espressione o di un tipo
- *sizeof (espressione)*  
Numero di byte (*char*) necessari per memorizzare i possibili valori dell'espressione
- *sizeof (nome\_tipo)*  
Numero di byte (*char*) necessari per memorizzare un oggetto del tipo passato come parametro

# Esercizio: dimensione int 1/2

---

- Scrivere un programma che stampi sullo schermo la dimensione in *byte* di un oggetto di tipo *int* sulla macchina su cui è eseguito



# Esercizio: dimensione int 2/2

---

```
int main()
```

```
{
```

```
    cout<<sizeof(int)<<endl ;
```

```
    return 0 ;
```

```
}
```

# Valori possibili per il tipo *int*

---

- Tipicamente un oggetto di tipo **int** memorizzato su  $n$  byte, ossia  $8*n$  bit, può contenere valori interi nell'intervallo

$$[-2^{(8*n - 1)}, 2^{(8*n - 1)} - 1]$$

- Quindi, per esempio su 4 byte si ha

$$[-2^{31}, 2^{31} - 1] =$$
$$[-2147483648, 2147483647]$$

- Lo standard prevede la disponibilità di costanti o funzioni per conoscere i limiti per ogni tipo di dato (che vedremo)

# Dimensioni tipiche

---

- In generale, le dimensioni dei tipi di dato dipendono dal *data model* utilizzato dal compilatore. I tipici *data model* prevedono:
  - Su macchine a 32 bit
    - **int** su 4 byte
  - Su macchine a 64 bit
    - **int** quasi sempre su 4 byte
    - sono, per esempio, su 4 byte nel caso del gcc sotto Linux

---

# Overflow

# Esercizio 1/3

---

- Scrivere un programma che definisce una variabile *i* di tipo *int*, la inizializza ad un valore qualsiasi e
- stampa il valore di *i*
- incrementa *i* di una unita'
- stampa il nuovo valore di *i*

# Esercizio 2/3

---

```
main()
{
    int i = 23 ;
    cout<<i<<endl ;
    i++ ;
    cout<<i<<endl ;
}
```

# Esercizio 3/3

---

- Modificare il programma affinché inizializzi  $i$  al seguente valore:

**2147483647**

- Quale sarà l'output del programma?
- Eseguirlo per controllare

# Overflow 1/3

---

- Si ha quando il valore di una espressione è **troppo grande** (in modulo) per essere contenuto
  - nel tipo di dato del risultato, oppure
  - nell'oggetto a cui si vuole assegnare tale valore
- In tal caso, il risultato o il nuovo valore dell'oggetto sarà in generale logicamente **non correlato** con l'operazione effettuata
  - E potrà variare da sistema a sistema



# Overflow 2/3

---

- E' stato segnalato automaticamente qualche errore durante l'esecuzione del precedente programma?
- Lo standard non prescrive segnalazioni d'errore di *overflow* a tempo di esecuzione
- Quello che succede è che le operazioni sono effettuate **senza controllare** se il risultato sarà corretto

# Overflow 3/3

---

- Una conoscenza approfondita della rappresentazione interna del tipo di dato permette di prevedere il valore risultante anche in caso di *overflow*
- Ma, come si è detto, tale valore può variare da sistema a sistema
- Non vedremo la rappresentazione dei numeri in memoria e, per semplicità, in tutti i programmi che faremo in questo corso, considereremo come **casuale** il risultato di una operazione in caso di *overflow*

# Esercizio 1/4

---

- Scrivere un programma che, letti due numeri interi in ingresso, stampi la loro somma sia se c'è stato *overflow* sia se non c'è stato, ma, nel caso ci sia stato, avvisi che c'è stato appunto un *overflow*
- Esempio:

*Inserisci i due numeri da sommare:  
2147483647 78*

*2147483647 + 78 = -2147483571*

*Attenzione: il risultato non è attendibile perché c'e' stato overflow*

# Esercizio 2/4

---

- Attenzione al fatto che l'utente può inserire anche numeri negativi !!!
  - Magari svolgere l'esercizio prima solo per numeri positivi
- Dall'analisi del problema ci accorgiamo che
  - in caso di overflow, una volta effettuata la somma il risultato è casuale, per cui tale valore non si può utilizzare per nessun tipo di deduzione

# Esercizio 3/4

---

- Altra idea *difettosa*
  - Confrontare il risultato della somma con il valore massimo o il valore minimo memorizzabile in un **int**
  - Purtroppo, anche se c'è stato overflow, tali confronti danno comunque esito positivo
    - Perché il risultato stesso è di tipo **int**, anche se c'è overflow
    - Ed in un oggetto di tipo **int** non si può ovviamente memorizzare un numero
      - più grande del massimo numero consentito per il tipo **int**, oppure
      - più piccolo del minimo numero memorizzabile in un numero di tipo **int**

# Provando a spiegare ...

---

- ... con una immagine



# Esercizio 4/4

---

- Soluzione in *somma\_overflow.cc*
- Guardare l'idea suggerita nella prossima slide solo dopo aver provato da soli

# Suggerimenti

---

- Suggerimento generale: per leggibilità memorizzare in una costante MAXINT l'intero più grande rappresentabile (scoperto per esempio con l'esercizio precedente)
- **Idea** Riflettere sulle seguenti domande
  - 1) e vero che la somma di due numeri positivi A e B di tipo **int** non supera MAXINT se e solo se  $B \leq \text{MAXINT} - A$  ?
  - 2) e vero che  $\text{MAXINT} - A$  non genera mai overflow se A è un numero positivo di tipo **int** ?



# Esercizio per casa

---

- Scrivere un programma che, letti due numeri interi in ingresso, stampi il risultato del prodotto e dica se tale prodotto ha generato *overflow* oppure no
- Soluzione non fornita

# Mini-prova scritta d'esame

---

- *mini-scritto\_24Ott16.pdf*
- Soluzione in *sol-mini-scritto\_24Ott16.pdf*
- Faremo questa prova di autovalutazione il 24 ottobre 2016