Riprendiamo l'esercizio calcolatrice

- Scrivere un programma ***calcolatrice*** che legga da tastiera due variabili intere *a* e *b*, poi
- Stampi un menu con un valore numerico associato a quattro operazioni possibili:
  1. *Addizione*
  2. *Sottrazione*
  3. *Moltiplicazione*
  4. *Divisione*
- legga in ingresso la variabile intera *controllo* che rappresenta la scelta di operazione
- Utilizzi un costrutto **`switch`** per effettuare l'operazione scelta usando *a* e *b* come operandi
- Stampi il risultato dell'operazione su schermo
  - Se l'operazione è una divisione bisognerà stampare parte intera e resto

Riprendiamo l'esercizio calcolatrice

- Eseguire il programma calcolatrice ed effettuare una somma usando come primo operando il valore 2147483647
- Cosa succede?
- Es.
  - 2147483647 + 78 = -2147483571

Overflow

- quando il valore di una espressione è troppo grande (in modulo) per essere contenuto nel tipo di dato del risultato (o nell'oggetto a cui si vuole assegnare tale valore) si ha un **overflow**
- 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

- Tipicamente un dato 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 il tipo **`int`** (4 byte) può rappresentare i numeri nell'intervallo

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

- per gli interi **con segno**
  - 1 bit è usato per il segno e 31 bit per rappresentare il numero

Overflow

- È però possibile specificare in C/C++ che si intende usare interi **senza segno**
- In questo caso, tutti e 32 i bit a disposizione si usano per rappresentare il numero

```
unsigned int a;
```

- il tipo **`unsigned int`** (4 byte) può rappresentare i numeri nell'intervallo

$[0, 2^{32} - 1] = [0, 4294967295]$

Overflow

- NOTA: sia col tipo int che col tipo **`unsigned int`** i numeri rappresentabili con 32 bit sono sempre

$2^{32} = 4294967296$

Overflow

- Lo standard prevede la disponibilità di costanti o funzioni per conoscere i limiti per ogni tipo di dato
- La funzione **`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

Overflow

- Il massimo numero di valori (numeri) rappresentabili con un certo tipo di dato si può dunque calcolare in un programma in questo modo

- ```
  // calcolo del numero di bytes
  int bytes = sizeof (<tipo-di-dato>);
  // calcolo del numero di bit
  // ciascun byte contiene 8 bit
  int bits = bytes * 8;
  // calcolo del ‘‘numero di numeri’’
  // rappresentabili con questo tipo di dato
  int values = exp2 (bits);
  ```

La funzione exp2 calcola 2^{bits}. Per usarla serve aggiungere `#include <math.h>` al programma
  ```

Overflow

- Quindi, il massimo numero rappresentabile con un **`unsigned int`** sarà calcolabile come

```
int max = values - 1;
```

- Mentre il minimo e massimo numero rappresentabile con un **`int`** sarà calcolabile come

```
int min = -(values / 2);
int max = (values / 2) - 1;
```

Esercizio

- Modificare il programma calcolatrice perché verifichi che non ci sia overflow sulle operazioni di somma

SUGGERIMENTI:

- memorizzare in una costante MAXINT l'intero più grande rappresentabile (come descritto nella slide precedente)

- Riflettere sulle seguenti domande:
  1. è vero che la somma di due numeri positivi A e B di tipo **`int`** non supera MAXINT se e solo se **`B <= (MAXINT - A)`** ?
  2. è vero che MAXINT – A non genera mai overflow se A è un numero positivo di tipo **`int`** ?

Riprendiamo l’esercizio calcolatrice

- Eseguire il programma calcolatrice ed effettuare una divisione usando come secondo operando il valore 0
- Cosa succede?
- Es.
  - 300 / 0

Terminazione forzata

- Abbiamo provato a far eseguire al processore una operazione illegale: una divisione per zero
- In questi casi il processore genera una ***eccezione hardware***
- Nella fase di avvio iniziale, il sistema operativo associa ad ogni possibile eccezione hardware del codice di gestione dell'eccezione stessa
- Tra le altre cose, il codice eseguito a seguito della nostra eccezione di divisione per zero, termina forzatamente il processo che ha generato l'eccezione

Terminazione forzata

- La terminazione forzata è una delle modalità (tra le più evidenti) di fallimento di un programma
- Un'altra è la restituzione di risultati errati
- Alcune cause comuni di fallimenti sono:
  - Le variabili non sono inizializzate
  - I valori passati alle funzioni o immessi dall'esterno non sono quelli attesi e non si effettuano controlli
  - C'è stato un overflow

Esercizio

- Modificare il programma calcolatrice perché verifichi che non ci sia divisione per zero