Introduzione alla Programmazione
a.a. 2023/24

Eserciziario

20 dicembre 2023

Indice

Come usare l'eserciziario

In questo eserciziario troverete gli esercizi proposti durante le lezioni. Non ci aspettiamo che li facciate tutti durante i laboratori, ma che li usiate anche per il lavoro individuale (ricordatevi che per ogni ora prevista in orario ci si aspetta che ne venga svolta almeno un'altra da soli durante la stessa settimana) e la preparazione finale dell'esame (per la quale sono previste grosso modo le stesse ore che avete in orario durante tutto l'anno).

Ogni volta che verrà affrontato un nuovo argomento caricheremo su Aulaweb una nuova versione aggiornata dell'eserciziaro che comprenderà anche gli esercizi sul nuovo argomento.

Organizzazione dell'eserciziario

La raccolta di esercizi è organizzata in parti, ciascuna delle quali è focalizzata su un nuovo concetto o tipo di costrutto e propone esercizi in cui lo si usa, naturalmente assieme a tutto quello visto fino a quel momento (per cui non è possibile ignorare una parte e dedicarsi alla successiva).

Ciascuna parte è introdotta da un *cheatsheet* riassuntivo delle regole e degli argomenti trattati nel capitolo e poi è strutturato in tre sezioni:

- **Esercizi di riscaldamento**: sono esercizi molto semplici pensati per chi è digiuno di programmazione. In essi dopo il testo, cioè dopo la descrizione del problema da risolvere con il vostro programma, è data la traccia del programma da scrivere riga per riga. Questa traccia è presentata nella forma di commenti C++ in modo che possiate direttamente inserirla nel vostro programma e gradualmente sostituire ogni riga con il codice da voi scritto.

  In questo modo vengono separate le due difficoltà della programmazione: individuare l'algoritmo che porta alla soluzione, e tradurlo in C++. In questa sezione introduttiva, potete concentrarvi solo sulla seconda, perché la prima (trovare l'algoritmo) è già risolta. Ad esempio, supponete di avere la seguente traccia di programma:

```
Scrivere un programma che ripete tre volte sul terminale la parola inserita da un  utente.

// scrivere sull'output un messaggio che chiede di inserire una parola
// dichiarare una variabile msg di tipo string
// leggere dallo standard input msg
// scrivere sull'output una andata a capo seguita da msg ripetuto 3 volte
```

  Ci aspettiamo che voi produciate un programma simile a

```
#include <iostream>
#include <string>
using namespace std;
int main() {
//Scrivere un programma che ripete tre volte sul terminale la parola inserita da un  utente.

// scrivere sull'output un messaggio che chiede di inserire una parola
cout << "Inserisci una singola parola (senza spazi): ";
// dichiarare una variabile msg di tipo string
string msg;
// leggere dallo standard input msg
cin >> msg;
// scrivere sull'output una andata a capo seguita da msg ripetuto 3 volte
cout << endl << msg << msg << msg;
}
```

  Per usare al meglio questo tipo di esercizi, prima provate a leggere solo il testo e a svolgerli in autonomia; se avete seguito bene le lezioni dovreste farcela senza troppo sforzo. Se ci riuscite, confrontate la vostra soluzione con quella proposta per vedere se sono analoghe e se non è così ragionate sui vantaggi/svantaggi delle due alternative. Se non riuscite a trovare autonomamente un algoritmo risolutivo, studiate e capite bene quello che vi proponiamo noi, copiatelo nel vostro codice sorgente e poi espandete ogni riga di commento nell'istruzione (o nelle istruzioni) corrispondenti.

- **Esercizi di base**: non potete passare al prossimo argomento (o sperare di passare l'esame) se non li sapete fare a occhi chiusi. Per ciascun esercizio, scrivete l'algoritmo che intendete implementare sotto forma di commenti (come abbiamo fatto noi per voi nella prima sezione) e solo dopo iniziate a scrivere il codice corrispondente, espandendo ciascuna riga. Per molti esercizi trovate un paio di cloni, ovvero esercizi molto simili, che hanno la stessa soluzione a meno di dettagli minimi. Così chi ha avuto difficoltà a trovare la soluzione al primo, può verificare svolgendo il secondo di aver assimilato bene la soluzione e saperla replicare.
- **Esercizi più avanzati**: per chi ambisce a imparare bene (e quindi prendere un voto alto). Se siete a corto di tempo potete svolgerli nell'ultima fase di preparazione per l'esame finale

Note utili

- Nella parte iniziale dell'eserciziario, verrete guidati nella risoluzione di un problema/esercizio, ossia vi saranno chiariti tutti gli elementi necessari a trovare una soluzione. Mentre il corso procede i problemi prenderanno una forma sempre più astratta: dovrete imparare a passare dall' inquadramento generale ai dettagli, formulando le domande giuste per ridurre le ambiguitá intrinseche nella formulazione di un problema
- Nel seguito useremo sempre
  - `stampare` come sinonimo di `stampare su standard output` (comando C++ `cout`)
  - `leggere` come sinonimo di `leggere da standard input` (comando C++ `cin`)
  - `a capo` come sinonimo di `carattere newline` (in C++: `'\n'` oppure il comando `std::endl`).
- Per compilare un file esempio.cpp (che contiene una funzione main!):
  `g++ -Wall -std=c++14 esempio.cpp -o esempio`
- **Attenzione:** Se è tutto corretto compila senza errori (nessun messaggio), ma se compila senza errori non è detto che sia tutto corretto! Eseguite il programma e vedete se fa quello che deve.
- Per eseguire il programma appena compilato
  `./esempio`
- Nel caso in cui il vostro programma comprenda più di un file sorgente .cpp, **tutti** i file sorgenti devono essere riportati sulla riga di compilazione (invece **non** vanno compilati gli header file, ossia i file .h). Ad esempio, se dovete compilare il programma esempio.cpp che oltre a se stesso include anche i sorgenti sorgente1.cpp e sorgente2.cpp, dovete compilare con
  `g++ -Wall -std=c++14 esempio.cpp sorgente1.cpp sorgente2.cpp -o esempio`
- Vedere i cheatsheet nell'Appendice per altri casi d'uso dei comandi indicati.

Parte 1

Espressioni, lettura e assegnazione

Impariamo ad acquisire familiarità con alcuni elementi base della programmazione: dichiarazioni, assegnazioni, e input-ouput.

Cheatsheet

`//commento`	riga di commento
`/*commento */`	commento
`10`	costante intera (letterale)
`int a;`	dichiara una variabile di tipo `int`
`int a = 10;`	dichiara una variabile di tipo `int` con valore iniziale
`float x = 3.14159;`	dichiara una variabile di tipo `float` con valore iniziale
`char c;`	dichiara una variabile di tipo carattere
`a = 10;`	assegna un valore alla variabile `a`
`a = b;`	assegna alla variabile `a` il valore della variabile `b`
`b+1;`	espressione, ha un valore
`a = b+1;`	assegna valore dell'espressione `b+1` alla variabile `a`
`"ciao"`	stringa costante (letterale)
`'c'`	carattere costante (letterale)
	Errore comune: usare singoli apici per una stringa di più caratteri
`true  false`	valori logici costanti (letterali)
`cin >> a`	legge variabile `a` da standard input (tastiera)
`cin >> a >> b`	legge variabili `a` e `b` da standard input
`cout << a`	scrive variabile `a` su standard output (schermo)
`cout << a << " " << b`	scrive variabili `a` e `b` su standard output con spazio in mezzo
Operatori aritmetici	*Operandi: tipi numerici, risultato espressione: un tipo numerico*
`+ - * /`	operazioni aritmetiche elementari (potenza non esiste)
`%`	operatore *modulo*, calcola il resto della divisione intera
Operatori di confronto	*Operandi: tipi numerici o altri, risultato espressione: `bool`*
`==`	operatore di confronto di uguaglianza
`!=`	operatore di confronto di disuguaglianza
`< <= > >=`	operatori di confronto d'ordine
Operatori logici	*Operandi: `bool`, risultato espressione: `bool`*
`!`	negazione (unario, vuole un solo operando)
`&&`	connettivo logico "AND", `true` se entrambi operandi sono `true`
`||`	connettivo logico "OR", `true` se almeno un operando è `true`

1.1 Esercizi di riscaldamento

1. Scrivere un programma che legge due interi e ne stampa la somma.

   (a) Seguire l'algoritmo proposto (che fissa una serie di dettagli ulteriori). Il programma deve essere scritto in un file chiamato `sum.cpp`.

```
// dichiarare due variabili a e b di tipo int
// leggere  a e b
// stampare la stringa "La somma vale "
// stampare il valore della somma
// stampare un a capo
```

(b) Seguire l'algoritmo proposto (che presenta piccole varianti rispetto al precedente). Il programma deve essere scritto in un file chiamato `sumalt.cpp`.

```
// dichiarare due variabili a e b di tipo int
// leggere a e b
// dichiarare una variabile sum di tipo int inizializzata con il valore della somma di a e b
// stampare la stringa "La somma vale " seguito da sum e un a capo
```

Confrontando i due algoritmi.

2. Come esercizio 1, ma prima di ogni input viene dato un messaggio di richiesta, del tipo "inserisci il valore di ...". Il programma deve essere scritto in un file chiamato `asksum.cpp`.

3. Scrivere un programma che scambia (in inglese swap) tra loro i valori di due variabili intere, lette da input, e stampa i valori prima e dopo lo scambio. Il programma deve essere scritto in un file chiamato `swapint.cpp`.

   Situazione iniziale:
   `a` contiene il valore x `b` contiene il valore y

   Risultato finale:
   `a` contiene il valore y `b` contiene il valore x

```
// chiedere di inserire il valore per la variabile a
// dichiarare una variabile a di tipo int
// leggere a
// chiedere di inserire il valore per la variabile b
// dichiarare una variabile b di tipo int
// leggere b
// stampare un a capo seguito dalla stringa "a vale " e  dal valore di a
// stampare un a capo seguito dalla stringa "b vale " e  dal valore di b
// scambiare fra loro i valori di a e b: per farlo serve una variabile di appoggio c
// dichiarare una variabile c di tipo int inizializzata con il valore di a
// assegnare ad a il valore di b
// assegnare a b il valore di c, ovvero il valore iniziale di a
// stampare un a capo seguito dalla stringa "a vale " e  dal valore di a
// stampare un a capo seguito dalla stringa "b vale " e dal valore di b
```

4. Scrivere un programma che legge due interi e ne stampa la differenza, seguendo l'algoritmo proposto (che fissa una serie di dettagli ulteriori). Il programma deve essere scritto in un file chiamato `dif.cpp`.

```
// chiedere di inserire due valori interi
// dichiarare due variabili a e b di tipo int
// leggere  a e b
// stampare la stringa "La differenza vale "
// stampare il valore di a - b
// stampare una andata a capo
```

1.2 Esercizi di base

5. Scrivere un programma che scambia i valori di due variabili di tipo `char`, lette da input, e stampa i valori prima e dopo lo scambio. Il programma deve essere scritto in un file chiamato `swapchar.cpp`.

6. Scrivere un programma che scambia in maniera circolare "verso sinistra" i valori di tre variabili di tipo float, lette da input, e stampa i valori prima e dopo lo scambio. Il programma deve essere scritto in un file chiamato `rotleft.cpp`.

   Situazione iniziale:
   `a` contiene il valore x `b` contiene il valore y `c` contiene il valore z

   Risultato finale:
   `a` contiene il valore y `b` contiene il valore z `c` contiene il valore x

   Ad esempio se vengono inseriti i valori 3.5, 4.7 e −8.978 li assegna a variabili `a`, `b` e `c` (nell'ordine) e stampa 3.5, 4.7 e −8.978. Poi assegna 4.7 ad `a`, −8.978 a `b` e 3.5 a `c` e stampa 4.7, −8.978 e 3.5.

7. Scrivere un programma che calcola perimetro e area di un rettangolo, dopo aver chiesto e letto i dati necessari. Il programma deve essere scritto in un file chiamato `rectangle.cpp`.

8. Scrivere un programma che chiede all'utente in che anno è nato e stampa quanti anni ha (supponiamo sia nato il 1 Gennaio). Il programma deve essere scritto in un file chiamato `agecalc.cpp`.

9. Scrivere un programma che calcola l'area di un triangolo, dopo aver chiesto e letto i dati necessari. Il programma deve essere scritto in un file chiamato `triangle.cpp`.

10. Scrivere un programma che prende in input il numero di ore (compreso fra 0 e 23) e di minuti (compreso fra 0 e 59) e stampa in output il numero di minuti totali. Il programma deve essere scritto in un file chiamato `time2min.cpp`.

11. Scrivere un programma che calcola circonferenza e area di un cerchio, dopo aver chiesto e letto i dati necessari. Supponiamo $\pi = 3.14$. Il programma deve essere scritto in un file chiamato `circle.cpp`.

12. Scrivere un programma che legge due interi e ne stampa la media (come numero reale). Ad esempio sull'input 1 e 2 stampa 1.5. Il programma deve essere scritto in un file chiamato `mean.cpp`.

13. Scrivere un programma che, per ciascuna di queste frasi, stampa la frase seguita dal simbolo `=` e da un'espressione booleana che calcola il suo valore di verità. Il programma deve essere scritto in un file chiamato `trueorfalse.cpp`.
    [SUGGERIMENTO: per stampare i booleani come `true` e `false` invece che come 1 e 0 si deve impostare a `true` il flag `boolalpha` di `cout`. Per fare questo si usa la stessa sintassi della stampa, ovvero si deve "stampare" un comando, come segue: `std::cout << std::boolalpha`]

    - tre è maggiore di uno
    - quattro diviso due è minore di zero
    - il carattere "zero" è uguale al valore zero
    - dieci mezzi è compreso fra zero escluso e dieci incluso (ossia: dieci mezzi è maggiore di zero E dieci mezzi è minore o uguale a dieci)
    - non è vero che tre è maggiore di due e minore di uno
    - tre minore di meno cinque implica sette maggiore di zero
      [SUGGERIMENTO: *A* implica *B* è vera se *B* è vera, oppure se *A* è falsa]

1.3 Esercizi più avanzati

14. Scrivere un programma che legge due numeri e li stampa in ordine crescente *senza confrontarli*. Il programma deve essere scritto in un file chiamato `sorttwo.cpp`.
    [SUGGERIMENTO: se alla media sottraggo la semidistanza, che valore ottengo?]

15. Scrivere un programma che scambia fra loro i valori di due variabili `int` senza usare variabili di appoggio. Il programma deve essere scritto in un file chiamato `magicswap.cpp`.
    [SUGGERIMENTO: l'or esclusivo, o XOR (in C++ l'operatore `^`), gode di varie proprietà, tra cui la proprietà di simmetria – cioè A^B == B^A – e la proprietà associativa – cioè (A^B)^C == A^(B^C). Inoltre, A^A ==0 e A^0==A per qualsiasi A, B e C.]

Parte 2

Scelte condizionali

Impariamo ad utilizzare i costrutti di scelta condizionale `if-else` e di scelta multipla `switch`.

Cheatsheet

code-block = istruzione; *oppure* { sequenza-di-istruzioni }

```
if ( espressione-booleana )
  code-block
```

```
if ( espressione-booleana )
  code-block-1
else
  code-block-2
```

N.B.: `else` non vuole una espressione booleana!

```
switch ( espressione ) {
   case valore-1:
    sequenza-di-istruzioni
    break;
   case valore-2:
    sequenza-di-istruzioni
    break;
   default:
    sequenza-di-istruzioni
}
```

Tipi primitivi:

CATEGORIA	NOME	LUNGHEZZA
Tipi interi	`int`	non specificato, tipic. 4 byte
	`long int` (o solo `long`)	>= `int`, tipic. 8 byte
	`short int` (`short`)	<= `int`, tipic. 2 byte
	`char`	1 byte
	Tutti possono essere anche `unsigned`	
Tipi floating point (reali)	`float`	32 bit (4 byte)
	`double`	64 bit (8 byte)
	`long double`	80 bit (10 byte)

Altri tipi comuni:

Tipo `std::string`	(stringhe “stile C++”) NON È UN TIPO PRIMITIVO: infatti occorre aggiungere in testa al file `#include <string>`
Costanti di tipo “stringa”	(stringhe “stile C”) SOLO PER COSTANTI LETTERALI, NON VARIABILI Esempio: `"Hello, world!"`

2.1 Esercizi di riscaldamento

1. Scrivere un programma che legge due caratteri e stampa la stringa “Uguali” se sono lo stesso carattere e la stringa “Diversi” se sono due caratteri differenti. [File `equalchars.cpp`]

```
// dichiarare due variabili a e b di tipo char
// leggere a e b
// se a e b sono uguali
//      stampare la stringa "Uguali"
// altrimenti
//      stampare la stringa "Diversi"
```

2. Scrivere un programma che legge tre interi e li stampa in ordine crescente, seguendo l’algoritmo proposto (che fissa una serie di dettagli ulteriori). [File `sort3int.cpp`]

```
// chiedere di inserire tre numeri interi
// dichiarare tre variabili a0, a1 e a2 di tipo int
// leggere a0, a1 e a2
// ordinare a0, a1 e a2, ovvero:
// dichiarare una variabile intera aux inizializzata con a1
// se a0 maggiore di a1 scambiare fra loro a0 e a1, cioe'
//      - assegnare ad a1 il valore di a0, ad a0 il valore di aux e...
//      ... ad aux il valore di a1 (a questo punto a0 <= a1==aux)
// se a0 maggiore di a2
//      - assegnare ad a1 il valore di a0, ad a0 il valore di a2 e...
//      ... ad a2 il valore di aux
// altrimenti
//      - se a1 maggiore di a2 scambiare fra loro a1 e a2, cioe'
//              -- assegnare ad a1 il valore di a2, ad a2 il valore...
//              ... di aux (a questo punto a0<=a1<=a2)
// stampare  la stringa  "I numeri inseriti, in ordine crescente, sono: "
// stampare il valore di a0, seguito dal carattere <
// stampare il valore di a1, seguito dal carattere <
// stampare il valore di a2
// stampare un  a capo
```

2.2 Esercizi di base

3. Scrivere un programma che legge un numero intero e ne stampa il valore assoluto (ovvero il numero senza segno, ad esempio se leggo −7 devo stampare 7, (non usare funzioni matematiche di libreria `cmath`). [File `absval.cpp`]

4. Scrivere un programma che legge tre numeri reali e li stampa in ordine decrescente. [File `sortdown3reals.cpp`]

5. Scrivere un programma che legge un numero intero da input e stampa se è o no divisibile per 13. [File `multipleof13.cpp`]

6. Scrivere un programma che verifica se tre numeri reali dati in input possono essere i lati di un triangolo, cioè se nessuno di essi è maggiore della somma degli altri due. (https://it.wikipedia.org/wiki/Disuguaglianza_triangolare) [File `triangleinequality.cpp`]

7. Scrivere un programma che chiede all’utente un numero reale e lo legge salvandolo in una variabile di tipo `float`. Quindi, ne fa una copia (in un’altra variabile di tipo `float`) e in cascata (ovvero usando il risultato di ciascuna operazione come argomento per la successiva), lo divide per 4.9, per 3.53 e per 6.9998. Poi, sempre in cascata, moltiplica per 4.9, per 3.53 e per 6.9998. Infine confronta il risultato ottenuto con il numero iniziale e se rappresentano due numeri reali diversi stampa `"moltiplicare NON e' l'inverso di dividere"`. [File `checkprecision.cpp`]

8. Scrivere un programma che verifica se un numero intero dato in input rappresenta o meno un anno bisestile (per semplicità intendiamo solo come divisibile per 4 e non gestiamo i casi "particolari", piu’ avanti, negli esercizi avanzati c’è l’esercizio completo su anno bisestile). [File `simpleleapyear.cpp`]

9. Scrivere un programma che implementa un turno di gioco di forbice-carta-sasso, ovvero che legge in input la mossa dei due giocatori (ogni mossa è un carattere 'f', 'c', 's', quindi due caratteri) e restituisce in output chi ha vinto seguendo le regole del gioco ("Giocatore 1 hai vinto!" oppure "Giocatore 2 hai vinto!"). Il programma deve controllare che i giocatori abbiano inserito uno dei tre caratteri validi.
https://it.wikipedia.org/wiki/Morra_cinese. [File `turnomorra.cpp`]

10. Scrivere un programma che legge da input un numero intero `Temp` e stampa:

    - "Freddo incredibile" se `Temp` è compreso fra −20 e 0
    - "Freddo" se `Temp` è compreso fra 1 e 15
    - "Normale" se `Temp` è compreso fra 16 e 23
    - "Caldo" se `Temp` è compreso fra 24 e 30
    - "Caldo da morire" se `Temp` è compreso fra 31 e 40
    - "Non ci credo, il termometro deve essere rotto" se `Temp` è superiore a 40 o inferiore a −20

    [File `temperature.cpp`]

11. Scrivere un programma che legge un numero intero compreso fra 1 e 12 e stampa il nome del mese corrispondente (1==gennaio...). Implementare usando lo `switch`. Se il numero non è compreso fra 1 e 12 stampa un messaggio di errore ed esce. [File `monthname.cpp`]

2.3 Per chi non si accontenta

12. Scrivere un programma che verifica se un numero intero dato in input rappresenta o meno un anno bisestile. Usare la regola del calendario gregoriano che si trova su Wikipedia alla voce "Anno bisestile". [File `leaptyearcomplete.cpp`]

13. Scrivere un programma che scrive in lettere i nomi italiani delle ore, approssimati ai 15 minuti. Il programma deve prendere in input due valori interi, uno tra 0 e 23 (ore) e l'altro tra 0 e 59 (minuti) e se i valori dati in input non rispettano il vincolo stampa un messaggio di errore ed esce ritornando -1 come codice di errore. Se l'input è corretto, scrive "Sono le ore " seguito dal valore delle ore (p.es. se è 11 scrive "undici", ma se è 1 scrive "l'una" e se è 0 scrive "mezzanotte") e dal valore dei minuti, approssimato per difetto al quarto d'ora (ad esempio se i minuti sono 15-29 scrive " e un quarto", se 30-44 scrive " e mezza", se 45-59 scrive " e tre quarti"; se 0-14 invece non scrive niente). Infine, se i minuti non sono divisibili esattamente per 15, scrive " passate". Esempio: 12:34 "Sono le ore dodici e mezza passate", 00:56 "Sono le ore mezzanotte e tre quarti passate". [File `saytime.cpp`]

14. Scrivere un programma che prende in input tre numeri reali, a, b e c e stampa le radici dell'equazione di secondo grado $ax^2 + bx + c$. Attenzione alle radici immaginarie. [File `roots.cpp`]

    [SUGGERIMENTO: Radice di x: `sqrt(x);` aggiungere in testa al file: `#include <cmath>`]

Parte 3

Cicli

Impariamo ad utilizzare i cicli `for`, `while`, e `do while`, per richiedere in modo efficiente l'esecuzione di un gruppo di istruzioni per più di una volta.

Cheatsheet

`++i`	`--i`	pre-incremento/pre-decremento (prima si incrementa/decrementa, poi si calcola il valore risultante)
`i++`	`i--`	post-incremento/post-decremento (prima si calcola il valore, poi si incrementa/decrementa)

Ciclo `for` "preconfezionato" per ripetere `N` volte un blocco di codice:

```
for ( i = 0; i < N; ++i )
   code-block
```

Ciclo `for` in generale:

```
for ( istruzione-1 ; espressione-booleana; istruzione-2 )
   code-block
```

Chi fa cosa:
`istruzione-1`: inizializzazione
`espressione-booleana`: test di terminazione
`istruzione-2`: avanzamento

Ordine esecuzione:
`istruzione-1` → `espressione-booleana` → `code-block` → `istruzione-2` ⟶ ↑

Ciclo while

```
while ( espressione-booleana )
   code-block
```

Ciclo do. . . while

```
do
   code-block
while ( espressione-booleana );
```

N.B.: entrambi terminano quando `espressione-booleana` vale `false`

Quale costrutto devo usare? Regola di prima approssimazione:

Conosco il numero di iterazioni da effettuare	⟶	`for`
Posso verificare una condizione di arresto prima di iniziare il ciclo	⟶	`while`
Posso verificare una condizione di arresto ma solo alla fine del ciclo	⟶	`do ... while`

Tecnica del look-ahead

```
leggi-input
while(input ok) {
  fai-qualcosa-su-input
  leggi-input // (successivo)
}
```

3.1 Esercizi di riscaldamento

1. Scrivere un programma che legge un certo numero di valori reali e ne stampa la media (notare che lo schema seguente fissa una serie di dettagli ulteriori non specificati nel "testo" dell'esercizio).

```
// stampare la stringa "Di quanti numeri vuoi fare la media?"
// dichiarare una variabile how_many di tipo int
// leggere how_many
// se how_many non è strettamente positivo
//      - stampare "Errore: il numero doveva essere positivo"
//      - uscire dal main ritornando il codice di errore 42
// dichiarare una variabile sum di tipo float inizializzata a 0
/* iterare how_many volte le seguenti istruzioni
    - stampare un a capo seguito dalla stringa "Inserisci un numero "
    - dichiarare una variabile x di tipo float
    - leggere  x
    - assegnare a sum la somma di sum e x
*/
// stampare  un a capo seguito dalla stringa "La media è "
// stampare la divisione di sum per how_many
```

   Implementate questo esercizio in tre varianti: usando un ciclo for [File `mediafor.cpp`] oppure un ciclo while [File `mediawhile.cpp`] oppure un ciclo do while [File `mediadowhile.cpp`].

2. Scrivere un programma che legge lettere maiuscole finché l'utente non inserisce un carattere che non è una lettera maiuscola e stampa la prima in ordine alfabetico. [File `alphafirst.cpp`]
   [SUGGERIMENTO: Le lettere maiuscole vengono rappresentate con numeri consecutivi, quindi per sapere se un carattere rappresenta una lettera maiuscola basta verificare che sia maggiore o uguale del carattere 'A' e minore o uguale al carattere 'Z'.]

```
// stampare la stringa "Inserisci una lettera maiuscola"
// dichiarare una variabile first di tipo char
/* ripetere
        - leggere first
   finché first minore di 'A' o maggiore di 'Z'
   // Hint: ovvero finché l'utente non inserisce una lettera maiuscola
*/
// Hint: a questo punto sappiamo che first è una lettera maiuscola!
// dichiarare una variabile c di tipo char inizializzata con 'Z'
// Hint: a questo punto sappiamo che first <= c!
/* ripetere
        - se c è minore di first
                -- assegnare a first il valore di c
        - stampa della stringa "Inserisci una lettera maiuscola (o altro carattere per terminare)"
        - lettura di c
   finché c è maggiore o uguale ad 'A' e minore o uguale a 'Z'
   // Hint: ovvero finché l'utente inserisce lettere maiuscole
*/
// stampare la stringa "La lettera più piccola inserita è " seguita da first
```

3. Scrivere un programma che scrive il fattoriale di un numero chiesto all'utente. Il fattoriale di un numero è definito per induzione come $0! = 1$ e $(n+1)! = (n+1) * n!$. Quindi, ad esempio $3! = (2+1)! = 3 * 2! = 3 * (1+1)! = 3 * 2 * 1! = 3 * 2 * (0+1)! = 3 * 2 * 1 * 0! = 3 * 2 * 1 * 1$. In generale $n! = n * (n-1) * (n-2) * ... * 1$. [File `fatt.cpp`]

```
// stampare la stringa "Inserire un numero positivo: "
// dichiarare una variabile intera n
// leggere n
// se n è minore di zero
//      - stampare "Avevo detto positivo!"
//      - uscire dal programma con l'istruzione   return 7;
// dichiarare una variabile intera F inizializzata a n
/* iterare su una variabile intera i inizializzata a n-1 e decrescente di 1 finché i è maggiore di 1
        - assegnare a F il prodotto di F e i
*/
// se F è zero
//      - stampare "Il fattoriale di 0 è 1"
// altrimenti
//      - stampare "Il fattoriale di " seguito da n, seguito da " è " seguito da F
```

4. Scrivere un programma che legge un numero intero `n` strettamente positivo ed un carattere, e stampa il carattere `n` volte. [File `stampanvolte.cpp`]

```
// stampare la stringa "Inserisci un numero maggiore di 0: "
// dichiarare una variabile len di tipo int
// leggere len
// se len non e` maggiore di zero
//      - stampare "Avevo detto positivo!"
//      - uscire dal programma con l'istruzione   return 1;
// stampare la stringa "Inserisci il carattere da replicare: "
// dichiarare una variabile c di tipo char
// leggere c
/* iterare su i a partire da 1 e fino a len
      - stampare c
*/
```

5. Scrivere un programma che legge un numero intero strettamente positivo e stampa il triangolo rettangolo fatto di `'*'` con lato lungo quanto il numero letto. [File `stampatriang.cpp`]

   Ad esempio su 5 stamperà:

```
*
**
***
****
*****
```

```
// stampare la stringa "Inserisci un numero maggiore di 0: "
// dichiarare una variabile length di tipo int
// leggere length
/* iterare su i a partire da 1 e fino a length
        - iterare su j a partire da 1 e fino a i
                -- stampare '*'
*/
```

6. Scrivere un programma che propone all'utente un menu con quattro alternative, ne legge la scelta e seleziona l'alternativa corrispondente. [File `menu.cpp`]

```
/*
   dichiarare una variabile intera answer
   ripetere
        - stampare la stringa "1  Prima scelta"
        - stampare la stringa "2  Seconda scelta" su una nuova riga
        - stampare la stringa "3  Terza scelta" su una nuova riga
        - stampare la stringa "0  Uscita dal programma" su una nuova riga
        - stampare la stringa "Fai una scelta: " su una nuova riga
        - leggere answer
        - Se il valore di answer è 1
                -- scrivere il messaggio: "Hai fatto la prima scelta"
        - Se il valore di answer è 2
                -- scrivere il messaggio: "Hai fatto la seconda scelta"
        - Se il valore di answer è 3
                -- scrivere il messaggio: "Hai fatto la terza scelta"
        - Se il valore di answer è 0
                -- scrivere il messaggio: "Hai scelto di uscire dal programma."
                -- terminare l'esecuzione.
        - In tutti gli altri casi
                -- scrivere il messaggio: "Scelta non valida"
   finché answer è diverso da zero
*/
```

7. Scrivere un programma che ripete le seguenti operazioni finché l'utente non decide di terminare:

   - chiede all'utente un numero intero positivo
   - stampa su una nuova riga tante `'|'` quanto è grande il numero
   - chiede all'utente se vuole terminare

   [File `barplot.cpp`]

```
// dichiarare una variabile answer di tipo carattere
/* ripetere
        - stampare la stringa "inserisci un numero intero positivo"
        - dichiarare una variabile n di tipo intero
        - leggere n
        - iterare su una variabile intera i a partire da 1 fino a n
                -- stampare '|'
        - stampare un'andata a capo
        - stampare su una nuova riga la stringa
        "inserisci s o S per terminare, qualsiasi altro carattere per proseguire"
        - leggere answer
finché answer è diverso sia da 's' che da 'S'
*/
// stampare la stringa "ho terminato perchè hai inserito " seguita da answer
// che cosa succede se inserisci un numero negativo e perchè?
```

8. Scrivere un programma che chiede all'utente un numero intero positivo e stampa il numero ottenuto leggendo il numero dato da destra verso sinistra. Ad esempio su 17 stampa 71, su 27458 stampa 85472 e così via. [File `reversedigits.cpp`]

```
// stampare la stringa "Inserire un numero positivo: "
// dichiarare una variabile intera k
// leggere k
// se k è minore di zero
//      - stampare "Valore non valido"
//      - uscire dal programma ritornando il codice di errore 66
// stampare su una nuova riga la stringa "Rovesciando " seguita da k
// dichiarare una variabile intera inv inizializzata a zero
/* finché k è maggiore di zero
        - dichiarare una variabile intera mod inizializzata con il resto della divisione intera di k per 10
        - assegnare a k il quoziente di k per 10
        - assegnare a inv la moltiplicazione di inv per 10
        - assegnare a inv la somma di inv e mod
*/
// stampare la stringa " si ottiene " seguita da inv
```

Varianti (in alternativa): modificare il programma in modo che se l'utente inserisce un numero negativo invece di uscire dal programma

(a) si ripeta la richiesta di inserimento (e la lettura) finchè non viene dato un numero positivo.

(b) si stampi il numero ottenuto leggendo il numero dato da destra verso sinistra. Ad esempio su -56 si stampi -65. [File `reversedigitsneg.cpp`]

3.2 Esercizi di base

9. Partendo dall'esercizio 2.9 (vedi precedente sezione) scrivere un programma che gioca più mani di morra cinese però contro il computer. Come primo passo consigliamo di modificare l'esercizio 2.9 in modo da inserire per il turno del computer (che sostituisce Giocatore 2) la generazione di una mossa random (suggerimento: generare un numero intero tra 0 e 2 e associarlo alle mosse 0=>'f', 1=>'c', 2=>'s'). Esempio minimale di generazione di un numero random compreso tra 0 e 9 (per l'esercizio servirà tra 0 e 2):

```
#include <cstdlib>
#include <ctime>
#include <iostream>

using namespace std;

int main() {
  srand(time(NULL)); // questa linea va eseguita una volta sola nel programma
  int randomNumber = (rand()%10); // questa invece va eseguita ogni volta che vogliamo generare un nuovo numero
  cout << randomNumber << endl;
}
```

Poi, alla fine di ogni mano il programma deve dire chi ha vinto (cioè il giocatore umano o il computer) ha vinto e chiedere all'utente se vuole continuare; se l'utente risponde 'S' o 's' prosegue con un'altra mano, altrimenti termina. [File `partitamorra.cpp`]

10. Scrivere un programma che chiede all'utente un numero intero K strettamente positivo e ne stampa il numero di cifre (in base 10). Ad esempio su 27458 stampa 5.

La richiesta deve essere esattamente questa `"Inserisci un numero intero strettamente positivo: "` (notare lo spazio in fondo);

L'output è:

— se $K <= 0$ allora `"Il numero inserito NON e' valido"`

— se $1 <= K <= 9$ allora `"Il numero inserito e' composto da 1 cifra"` (notare il singolare)

— se $K >= 10$ allora `"Il numero inserito e' composto da N cifre"` dove `N` è il numero di cifre di K (notare il plurale)

[File `numdigits.cpp`]

11. Scrivere un programma che chiede all'utente di inserire numeri interi. Dopo ogni numero inserito chiede all'utente se terminare con l'inserimento (carattere 'n') o continuare (qualsiasi altro carattere). Finito il ciclo di lettura dei numeri stampa la media dei numeri letti.
[File `mediainterattiva.cpp`]

12. Scrivere un programma che legge due numeri interi strettamente positivi (le 2 basi di un trapezio rettangolo: b e B con b<=B) e stampa il trapezio rettangolo fatto di `'x'` con le basi lunghe quanto i numeri letti, e l'altezza pari alla differenza fra le basi più uno.
[File `stampatrapezio.cpp`]

    Ad esempio avendo in input 5 e 9 stamperà:

    ```
    xxxxx
    xxxxxx
    xxxxxxx
    xxxxxxxx
    xxxxxxxxx
    ```

    (che è alto $5 = 9 - 5 + 1$). Si noti che data la scelta dell'altezza a ogni riga bisogna stampare un carattere in più.

13. Scrivere un programma che chiede all'utente un numero intero positivo n e stampa un rombo di asterischi che sulle due diagonali ha $2 * n + 1$ caratteri. Ad esempio con 8 in input stampa

    ```
            *
           ***
          *****
         *******
        *********
       ***********
      *************
     ***************
    *****************
     ***************
      *************
       ***********
        *********
         *******
          *****
           ***
            *
    ```

    che sulle due diagonali ha 17 caratteri. [File `stamparombo.cpp`]
    [SUGGERIMENTO: È più facile stampare il rombo con due cicli, il primo per le righe in cui il numero di asterischi cresce e il secondo per le righe in cui il numero di asterischi diminuisce. In questo modo si può suddividere l'esercizio in due parti: (a) stampare il triangolo superiore (la parte superiore del rombo), dopodichè (b) adattare il codice per stampare il triangolo inferiore (la parte inferiore del rombo).]

14. Scrivere un programma che chiede all'utente di scegliere il suo colore preferito presentandogli un menu con almeno 5 scelte di colori i cui nomi iniziano con una lettera diversa (ad esempio rosso, verde, blu, giallo e arancio). Per selezionare la scelta l'utente potrà usare l'iniziale del colore scelto, indifferentemente in maiuscolo o minuscolo. Se il carattere inserito non corrisponde a nessun colore proposto il programma ricomincerà da capo presentando il menu con le scelte. [File `menucolors.cpp`] [SUGGERIMENTO: Come base di partenza può essere usato l'esercizio 3.6]

15. Scrivere un programma che chiede all'utente un numero intero e verifica se è *palindromo*, ovvero se le sue cifre (in base 10) lette da destra a sinistra corrispondono alle cifre lette da sinistra a destra (ad esempio 165373561 è palindromo). Dopo la verifica stampa il risultato all'utente. Per risolvere questo esercizio si può partire dalla soluzione dell'esercizio 3.8. [File `ispalindrome.cpp`]

16. Scrivere un programma che chiede all'utente un numero intero e lo legge. Poi chiede all'utente di provare a indovinarne il quadrato e se l'utente inserisce il valore giusto gli dice `"Bravo!"` ed esce, altrimenti gli propone di riprovare finché l'utente non riesce ad indovinare.
[File `guesspowertwo.cpp`]

3.3 Esercizi più avanzati

17. Scrivere un programma che verifica se un numero intero positivo dato in input è un *numero di Armstrong*. Un intero positivo a n cifre si dice *numero di Armstrong* se è uguale alla somma delle potenze n-esime delle cifre che lo compongono. Ad esempio $153 = 1^3+5^3+3^3 = 1*1*1+5*5*5+3*3*3$ è un numero di Armstrong, come pure $1634 = 1^4+6^4+3^4+4^4 = 1+1296+81+256$. [File `armstrongnum.cpp`]

18. Scrivere un programma che legge un intero positivo e stampa il numero di zeri alla fine del suo fattoriale (in base 10) **senza calcolarne il fattoriale**. Ad esempio su 5 stampa 1 perché $5! = 12\mathbf{0}$, mentre su 11 stampa 2 perché $11! = 39,916,8\mathbf{00}$. Si noti che, siccome il fattoriale diventa rapidamente più grande dei numeri rappresentabili sul calcolatore, è molto importante riuscire a calcolare quanto richiesto senza dover calcolare il fattoriale. [File `factzeroes.cpp`]

19. Scrivere un programma che legge un intero positivo compreso fra 1 e 3000 e lo stampa in notazione romana. [File `romannum.cpp`]

    Si ricorda che

    - i numeri romani sono scritti usando le lettere I (per 1), V (per 5), X (per 10), L (per 50), C (per 100), D (per 500) e M (per 1000) e rappresentano un numero in maniera additiva (non posizionale come i numeri arabi), partendo dai simboli che rappresentano i numeri più grandi a sinistra e man mano scendendo con simboli che rappresentano numeri sempre più piccoli; ad esempio MMXVII rappresenta 2017 come $1000 + 1000 + 10 + 5 + 1 + 1$.
    - si possono incontrare dei simboli in ordine inverso, ma in questo caso i valori invece di andare sommati vanno sottratti; questo meccanismo può essere usato solo per i numeri 4, rappresentato da IV, 9, rappresentato da IX, 40, rappresentato da XL, 90, rappresentato da XC, 400, rappresentato da CD, e 900, rappresentato da CM. Quindi ad esempio 1984 si rappresenta con MCMLXXXIV e 999 si rappresenta con CMXCIX (e non con IM).

    [SUGGERIMENTO: La logica di rappresentazione dei numeri romani è di sommare da sinistra a destra le rappresentazioni dei numeri 1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4 e 1.]

Parte 4

Array

Impariamo ad utilizzare gli array.

Cheatsheet

`float x[N];`	Dichiara un array di `float` di lunghezza `N`
`int x[4]={1,2,3,4};`	Dichiara e inizializza un array di 4 interi
	N.B. Anche `x[]={1,2,3,4};` `x[8]={1,2,3,4};` (inizializza i primi 4 e azzera il resto)
	N.B. `x[2]={1,2,3,4};` ⟶ errore "too many initializers"
`x[i]`	Accede all'i-esimo elemento dell'array `x`
	Se `i<0` o `i>=N` dà segmentation fault!
	errore comunissimo, verificate sempre la validità dei valori degli indici!!
`x[i] = 1;`	Imposta un valore per l'elemento i-esimo ("scrittura")
`a = x[i];`	Usa valore dell'elemento i-esimo ("lettura")

Non si possono applicare operatori aggregati, ad es. no `array1 = array2`, no `cout << array`, no `array = 0`.

4.1 Esercizi di riscaldamento

1. **Stampa un array di `int`.** Scrivere un programma che, dati un array `a` di `int` e la sua lunghezza `N`, stampa tutto l'array. [File `stampaArrayInt.cpp`]

```
// Creare e popolare un array a di lunghezza 7.
int a[7]={2, 4, 34, 78, 4, 3, 876};
// Iterare sulla variabile intera i a partire da 0 e fino a 7 escluso:
// - Stampare il valore i-esimo dell'array in posizione i e a capo
```

   Output atteso:

```
Valore di a[0] = 2
Valore di a[1] = 4
Valore di a[2] = 34
Valore di a[3] = 78
Valore di a[4] = 4
Valore di a[5] = 3
Valore di a[6] = 876
```

2. **Stampa inverso di un array di `float`.** Scrivere un programma che lavora su un array di `float` e li stampa in ordine inverso [File `stampaArrayFloat.cpp`]

```
// Creare e popolare un array a di lunghezza 5.
float a[5]={2.4, 5.67, 34, 28.456, 846.42};
// Iterare sulla variabile intera i a partire da N-1 (4) e fino a 0 incluso:
```

```
    // - Stampare il valore i-esimo dell'array in posizione i e a capo
```

Output atteso:

```
Valore di a[4] = 846.42
Valore di a[3] = 28.456
Valore di a[2] = 34
Valore di a[1] = 5.67
Valore di a[0] = 2.4
```

3. **Crea un array con N elementi di tipo `int`.** Scrivere un programma che dichiara un array `a` di `N` interi e lo "popola" (assegna valori ai suoi elementi). [File `creaArrayInt.cpp`]

```
    // Dichiarare una costante N con valore 10
    // Dichiarare un array a di N interi
    // Iterare sulla variabile intera i a partire da 0 e fino a N escluso:
    //      Assegnare all'elemento i-esimo di a il valore N-i
    // Stampare l'array in ordine crescente da a[0] a a[N-1] come visto sopra
```

Output atteso:

```
Valore di a[0] = 10
Valore di a[1] = 9
Valore di a[2] = 8
Valore di a[3] = 7
Valore di a[4] = 6
Valore di a[5] = 5
Valore di a[6] = 4
Valore di a[7] = 3
Valore di a[8] = 2
Valore di a[9] = 1
```

4. **Crea un array con N elementi di tipo `float` con valori dipendenti da un input.** Scrivere un programma simile al precedente, ma che lavora su un array di `float` e richiede un valore iniziale `v` di tipo `float` usato per inizializzare i suoi elementi. [File `creaArrayFloat.cpp`]

```
    // Dichiarare una costante N con valore 10
    // Dichiarare un array a di N float
    // Chiedere in input un valore float v;

    // Iterare sulla variabile intera i a partire da 0 e fino a N escluso:
    // - Assegnare all'elemento i-esimo di a il valore i*v;

    // Stampare l'array in ordine crescente da a[0] a a[N-1] come visto sopra
```

Esempio di Output atteso

```
Inserisci valore di v = 1.56
Valore di a[0] = 0
Valore di a[1] = 1.56
Valore di a[2] = 3.12
Valore di a[3] = 4.68
Valore di a[4] = 6.24
Valore di a[5] = 7.8
Valore di a[6] = 9.36
Valore di a[7] = 10.92
Valore di a[8] = 12.48
Valore di a[9] = 14.04
```

5. **Leggi un array di `int` da tastiera.** Scrivere un programma che dichiara un array `a` di `N` interi e lo "popola" leggendo valori da input. [File `leggiArrayInt.cpp`]

```
    // Dichiarare una costante N con valore 10
    // Dichiarare un array a di N interi

    // Iterare sulla variabile intera i a partire da 0 e fino a N escluso:
    // - Dichiarare una variabile intera val
    // - Stampare il messaggio composto dalla stringa "Inserisci un valore intero per a[" i "] = "
    // - Leggere da input un valore val
    // - Assegnare all'elemento i-esimo di a il valore di val

    // Stampare l'array in ordine crescente da a[0] a a[N-1] come visto sopra
```

Esempio di Output atteso

```
Inserisci un valore intero per a[0] = 4
Inserisci un valore intero per a[1] = 5
Inserisci un valore intero per a[2] = 2
Inserisci un valore intero per a[3] = 44
Inserisci un valore intero per a[4] = 788
Inserisci un valore intero per a[5] = 3354
Inserisci un valore intero per a[6] = 56343
Inserisci un valore intero per a[7] = 35643
Inserisci un valore intero per a[8] = 353
Inserisci un valore intero per a[9] = 3
Valore di a[0] = 4
Valore di a[1] = 5
Valore di a[2] = 2
Valore di a[3] = 44
Valore di a[4] = 788
Valore di a[5] = 3354
Valore di a[6] = 56343
Valore di a[7] = 35643
Valore di a[8] = 353
Valore di a[9] = 3
```

6. **Leggi un array di `float` da tastiera e stampalo su singola riga.** Scrivere un programma simile al precedente, ma che lavora su array di `float` e stampa il contenuto dell'array su una sola riga (gestire la virgola nel caso finale). Usare `N = 5`. [File `leggiArrayFloat.cpp`]

   Esempio di Output atteso

```
Inserisci un valore float per a[0] = 5.67
Inserisci un valore float per a[1] = 34.2
Inserisci un valore float per a[2] = 73.56
Inserisci un valore float per a[3] = 345.23
Inserisci un valore float per a[4] = 498.45
I valori contenuti nell'array a sono: { 5.67, 34.2, 73.56, 345.23, 498.45 }
```

7. **Media**: scrivere un programma che legge `N` valori reali, li memorizza in un array di lunghezza `N`, e ne stampa la media. [File `average.cpp`]

```
// copiare qui il codice del programma "leggiArrayFloat.cpp"
// dichiarare una variabile sum di tipo float e inizializzarla a zero
// iterare su i a partire da 0 e fino a N-1
    //- sommare il contenuto dell'i-esimo elemento di a a sum
// stampare la divisione di sum per N
```

   Esempio di Output atteso

```
Inserisci un valore float per a[0] = 67.7
```

```
Inserisci un valore float per a[1] = 56.7
Inserisci un valore float per a[2] = 73.1
Inserisci un valore float per a[3] = 89.5
Inserisci un valore float per a[4] = 56.0
I valori contenuti nell'array a sono: { 67.7, 56.7, 73.1, 89.5, 56 }
La media dei valori contenuti nell'array a è:  68.6
```

8. **Parabola**: scrivere un programma che dati un array `arr` di `float`, la sua lunghezza `N`, e tre valori float `a`, `b`, `c` memorizza in ogni elemento `x`-esimo dell'array il valore ax^2+bx+c. Poi stampa l'array. [File `parabola.cpp`]

```
// Dichiarare una costante N con valore 10
// Dichiarare un array arr di N float

// leggere da input un valore di a per il calcolo di ax^2+bx+c
// leggere da input un valore di b per il calcolo di ax^2+bx+c
// leggere da input un valore di c per il calcolo di ax^2+bx+c

// stampare "Valori nell'intervallo [0,9] della parabola  " seguito dall'equazione della parabola

// Iterare sulla variabile intera x a partire da 0 e fino a N escluso:
// - Assegnare all'x-esimo elemento dell'array il valore ax^2+bx+c

// Stampare l'array in ordine crescente da arr[0] a arr[N-1] come visto sopra
```

Esempio di Output atteso

```
Fornire il valore di a = 2.34
Fornire il valore di b = 5.45
Fornire il valore di c = -8.76
Valori nell'intervallo [0,9] della parabola  2.34x^2 + 5.45x + -8.76
Valore di arr[0] = -8.76
Valore di arr[1] = -0.97
Valore di arr[2] = 11.5
Valore di arr[3] = 28.65
Valore di arr[4] = 50.48
Valore di arr[5] = 76.99
Valore di arr[6] = 108.18
Valore di arr[7] = 144.05
Valore di arr[8] = 184.6
Valore di arr[9] = 229.83
```

4.2 Esercizi di base

9. Scrivere un programma che legge N=5 interi e li salva in un array di `int`. Quindi stampa il valore massimo contenuto nell'array e il numero di volte in cui questo appare. Il programma appena avviato non stampa nessun messaggio e resta in attesa dei 5 valori interi separati da spazio. Premere invio dopo l'ultimo intero inserito. [File `maxInt.cpp`]

   Suggerimenti:

   per trovare il massimo dell'array e il numero di volte in cui questo appare ci sono (almeno) due soluzioni:

   - una che percorre l'array due volte, una volta per cercare il numero massimo e una altra per contare quante volte è presente
   - un'altra che percorre l'array una sola volta ed ogni volta che trova un massimo (relativo) conta poi quante volte e presente sino a che non trova un altro massimo (se esiste). L'ultimo massimo trovato è quello definitivo.

   Per fornire in input N interi è possibile inserirli separandoli da spazio e premere una solo volta invio alla fine, non è necessario premere invio dopo ogni intero.

   L'output, su due righe, è:

```
"maxVal = " seguito dal valore massimo
"maxCount = " seguito dal numero di volte in cui questo appare
```

Esempio di esecuzione:

```
9 11 11 3 -23
maxVal = 11
maxCount = 2
```

10. Scrivere un programma che legge `N` interi in un array `a` di `int` (vedi `leggiArrayInt`). Quindi con un opportuno messaggio di output stampa il numero `P` dei numeri pari contenuti nell'array ed il numero `D` di quelli dispari (`P` e `D` sono quindi entrambi valori interi). [File `nPari.cpp`]

11. Scrivere un programma `reverse` che legge `N` interi in un array `source` (vedi `leggiArrayInt`), e poi copia in un array `dest` gli elementi di `source` in ordine inverso.

    Quindi stampa `source` e `dest` (lasciando una riga vuota in mezzo per chiarezza). [File `reverse.cpp`]

12. Scrivere un programma che, usando l'algoritmo "crivello di Eratostene", trova i numeri primi minori di 1000. Verificare i risultati ottenuti con la vostra implementazione confrontandoli con la lista di primi riportati qua: https://it.wikipedia.org/wiki/Lista_di_numeri_primi

    [File `crivello.cpp`]

```
CRIVELLO DI ERATOSTENE (n)

1) Creare un array di bool chiamato isprime di lunghezza n, inizializzandolo a tutti valori true.

   Al termine dell'algoritmo, l'elemento i-esimo di isprime varrà true se i e' primo, false altrimenti

2) Inizialmente, sia p pari a 2, il numero primo più piccolo (quindi 0 e 1 non sono primi).

3) Partendo da p escluso, marcare come NON PRIMI tutti i numeri multipli di p (2p, 3p, 4p...).
   Ovvero impostare a false ogni elemento isprime[2*p], isprime[3*p], ...

4) Partire da p=p+1 e scorrere in avanti l'array isprime finché non si trova il primo numero
   NON marcato (isprime[p] è true), oppure finché non è finita la lista

5) Se la lista è  finita, stop. Altrimenti p diventa pari al numero trovato e si ricomincia dal punto 3)
   (la prima volta sarà per p=3)

All'uscita dell'algoritmo, tutti i numeri non marcati (tali che il corrispondente elemento di isprime
vale ancora true) sono tutti i numeri primi < n

Stampare tutti i numeri tali che il corrispondente elemento di isprime è true.
```

Per capire meglio il funzionamento dell'algoritmo è possibile osservare l'animazione presente nella pagina Wikipedia che mostra graficamente l'esecuzione di una variante molto simile sui numeri compresi tra 0 e 120: https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

[SUGGERIMENTO: per implementare una versione base (senza ottimizzazioni) del crivello si può procedere in questo modo:

(a) Ciclare su tutti gli N elementi di `isprime` per impostarli a `true`

(b) Impostare gli elementi di `isprime` con indice 0 e 1 a `false`

(c) Ciclare sugli elementi di `isprime` con `p` che va da 2 a N-1.

   i. Se in `isprime` si trova un elemento `true` in posizione `p`

   ii. Ciclare su tutti i multipli di `p` per marcarli `false`

(d) Ciclare su tutti gli N elementi di `isprime` è stampare l'indice di quelli pari a `true`

Ricordate il criterio per scegliere tra `for`, `while` e `do ... while` (ripassare parte sui cicli).]

13. Scrivere un programma che definisce due valori costanti, `M` pari a 5 e `N` pari a 8. Dichiara poi un array bidimensionale `a` di dimensioni `M×N` con `int a[M][N];` (`M`=righe e `N`=colonne), e riempe ogni riga con `N` valori pari all'indice della riga corrente. Terminata la fase di inizializzazione dell'array, il programma stampa l'array. [File `bidimensional.cpp`]

    Output atteso con `M`=5 e `N`=8:

    ```
    0 0 0 0 0 0 0 0
    1 1 1 1 1 1 1 1
    2 2 2 2 2 2 2 2
    3 3 3 3 3 3 3 3
    4 4 4 4 4 4 4 4
    ```

14. Scrivere un programma che definisce un valore costante, `N` pari a 10.

    Dichiara poi un array bidimensionale `tavolaPitagorica` di dimensioni `N×N`, e lo riempie dei valori della tavola pitagorica, dove l'elemento (`i`, `j`) contiene il prodotto tra `i+1` e `j+1` (perché?).

    Infine chiede all'utente una coppia di valori tra 1 e 10 (verificare che siano entrambi nell'intervallo altrimenti richiederli entrambi), e restituisce il loro prodotto – ottenuto **consultando la tavola pitagorica** come una *look-up table*, e non eseguendo la moltiplicazione. [File `tabelline.cpp`]

4.3 Esercizi più avanzati

15. Scrivere un programma che legge un array `a` e calcola un valore di tipo `bool` che vale `true` se l'array è palindromo. Poi stampa un messaggio che comunica il risultato all'utente. [File `palyndromeArray.cpp`]

    [SUGGERIMENTO: usare il programma `reverse`.]

16. Scrivere un programma che legge un array di `int` e stampa la frequenza di ogni valore contenuto (il numero di volte che compare). [File `frequenze.cpp`]

    [SUGGERIMENTO: conviene avere un array di contatori (`int`) lungo tanto quanto l'array di ingresso.]

17. Scrivere un programma che legge un array di `int` e stampa il secondo valore più elevato. [File `secondo.cpp`]

18. Scrivere un programma che legge un array di `int source` e scrive in un altro array `dest` il contenuto dell'array `source` ordinato in modo crescente. Poi stampa `dest`. [File `sortInt.cpp`]

19. Scrivere un programma che legge un array di `int`, riordina i suoi elementi in modo crescente, e poi lo stampa. [File `sortInPlace.cpp`]

20. Scrivere un programma che esegue lo stesso compito di `reverse`, ovvero legge un array di `float` e inverte l'ordine dei valori contenuti, ma questa volta **senza usare un altro array** come spazio di lavoro. [File `reverseInPlace.cpp`]

    [SUGGERIMENTO: Basta fare swap fra le celle poste alla stessa distanza dagli estremi dell'array.]

21. Scrivere un programma che legge un array di interi positivi, lo scorre dall'inizio alla fine, e di tutti gli elementi che sono ripetuti in sequenza contigua cancella tutte le occorrenze tranne una, trasformando le ripetizioni in elementi unici (vedi esempi qui sotto).

    Al termine del procedimento, i valori che non sono stati eliminati devono essere contenuti in elementi consecutivi dello stesso array, e gli elementi rimanenti devono essere azzerati. Il programma infine stampa tutti gli elementi non zero. [File `unique.cpp`]

    Esempi:

    Array iniziale:

1	1	1	2	3	3	4

    Risultato finale:

1	2	3	4	0	0	0

    Array iniziale:

2	2	1	2	3	3	4

    Risultato finale:

2	1	2	3	4	0	0

    Array iniziale:

2	2

    Risultato finale:

2	0

    Array iniziale:

5

    Risultato finale:

5

    [SUGGERIMENTO: Il programma è semplice se copiate il primo elemento di ogni sequenza ripetuta in un array ausiliario, e alla fine ne ricopiate il contenuto nell'array originale.]

22. Scrivere un programma come il precedente, ma realizzato senza usare array ausiliari (dovete usare un algoritmo *in place*). [File `uniqueInPlace.cpp`]

23. La tavola pitagorica è simmetrica, quasi metà di essa contiene informazione ripetuta. Precisamente, N(N−1)/2 elementi sopra la diagonale sono uguali a N(N−1)/2 elementi sotto la diagonale.

	1	**2**	**3**	**4**
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12
4	4	8	12	16

$N = 4,\ N(N-1)/2 = 6$

Scrivere un programma che usa un array monodimensionale per rappresentare la tavola pitagorica usando solo gli N(N+1)/2 elementi necessari. Dal punto di vista dell'utente il programma deve comportarsi in modo identico a quello dell'esercizio 14.
[File `tabellineTriang.cpp`]

Parte 5

Array avanzati

Impariamo a realizzare operazioni su array e matrici più sofisticate di quelle viste fino ad ora: inserimento di valori, algoritmi di ordinamento e di ricerca degli elementi.

5.1 Esercizi di riscaldamento

1. Scrivere un programma che effettui la ricerca dell'elemento `item` (un intero) nell'array `v` (array di 15 interi letti da input).

   [File `sequentialSearch.cpp`]

```
// dichiarare una variabile const int N inizializzata a 15
// dichiarare una variabile int item
// leggere item da input
// dichiarare un array v di N interi
// leggere v da input (vedi esercizi parte precedente)
// dichiarare una variabile int loc e inizializzarla a -1
// dichiarare una variabile bool found e inizializzarla a false
/* iterare sugli elementi di v fino a che found diventa true o  si è iterato su tutto l'array
   - se il valore alla pos corrente (i) e' uguale a item
   -- assegnare true a found e il valore di i a loc
*/
// se trovato, scrivere su output: item " è stato trovato in posizione " loc
// altrimenti scrivere: item " non è stato trovato"
```

2. Scrivere un programma che effettui la ricerca dell'elemento `item` (un intero) nell'array `v` (array di 15 interi ORDINATI letti da input).

   [File `binarySearch.cpp`]

   NOTA. Per l'algoritmo di binarySearch vi rimandiamo alle slides presentate a lezione.

   [SUGGERIMENTO: Ricordate che il metodo di Ricerca Binaria opera su sequenze ordinate!]

3. Scrivere un programma che effettui l'ordinamento di un array `v` dato secondo l'algoritmo *Selection Sort*

   [File `selectionSort.cpp`]

```
// dichiarare una variabile int greatestIndex
/* iterare sull'array dalla prima all'ultima posizione
        - memorizzare in greatestIndex la posizione corrente (sia i)
        - *//* iterare sull'array dalla posizione successiva alla corrente (i+1) fino all'ultimo elemento
                -- se il valore alla pos corrente (j) e' < del valore alla pos greatestIndex
                          --- memorizzare j in greatestIndex
          *//*
        - scambiare il valore alla posizione i con quello alla pos greatestIndex
*/
```

4. Scrivere un programma che prende in input 20 numeri interi e li stampa in ordine decrescente.

   [File `sortDown.cpp`]

```
// scegliere 20 come valore per la costante N
// dichiarare un array v di interi
// leggere v da input (leggiArrayInt)
// visualizzare su output i valori inseriti (stampaArrayInt)
// usare SelectionSort per ordinare v in ordine decrescente
// usare stampaArrayInt per stampare i valori
```

5. Scrivere un programma che permetta di riempire l'array bidimensionale (matrice) di interi `A`, di dimensioni `M` × `N`, con valori letti da input.

   [File `readMatrix.cpp`]

```
// Dichiarare le variabili necessarie, dando a M e N i valori 3 e 4 rispettivamente
/* iterare sulle righe della matrice (indice i) fino a M
   *//* iterare sulle colonne della matrice (indice j) fino a N
          - leggere un valore e memorizzarlo in A[i][j]
   *//*
*/
```

6. Scrivere un programma che stampi su output l'array bidimensionale di interi `A`, di dimensioni `M` × `N`.

   [File `printMatrix.cpp`]

```
      // Dichiarare le variabili necessarie, dando a M e N i valori 3 e 4 rispettivamente
      /* iterare sulle righe della matrice (indice i) fino a M
         *//* iterare sulle colonne della matrice (indice j) fino a N
                - scrivere A[i][j]
         *//*
      */
```

5.2 Esercizi di base

7. Confrontare l'efficienza della sequentialSearch vs binarySearch quando si cerca se un certo valore (item) è presente nell'array. Modificare i precedenti esercizi 1 e 2 nel seguente modo:

   (a) Inizializzare con valori a scelta un array di 30 elementi direttamente nel codice. I valori scelti devono essere inseriti in ordine crescente. (in questo modo si evita di dover digitare in input ogni volta i valori ad ogni esecuzione)

   (b) Dichiarino una variabile intera `count` inizializzata a 0

   (c) Ogni volta che nel programma si controlla se l'elemento che si sta cercando (item) è l'elemento corrente dell'array, incrementare count

   (d) Alla fine, stampare il valore di count (numero di accessi effettuati all'array)

   Provare quindi a cercare con entrambi gli algoritmi (e su array contenenti gli stessi 30 valori) degli elementi che sono memorizzati all'inizio, a metà, e alla fine dell'array. Quanti accessi sono necessari con i due algoritmi? Ragionare su quale sia meno costoso (e quindi il più veloce).

   [File `sequentialSearchCount.cpp`] [File `binarySearchCount.cpp`]

8. Scrivere un programma che, letta una matrice di float A, quadrata di dimensione 2x2, calcoli il determinante di tale matrice e lo stampi in uscita. `https://it.wikipedia.org/wiki/Determinante_(algebra)#Metodi_di_calcolo`
   [File `det2.cpp`]

9. Scrivere un programma che verifica se una matrice è simmetrica. Stampa poi su output l'esito della verifica.

   [File `verifySymmetry.cpp`]

   [SUGGERIMENTO: Una matrice simmetrica è una matrice QUADRATA i cui elementi sono simmetrici rispetto alla diagonale principale. Es:

   $$\begin{matrix} 1 & 2 & 3 \\ 2 & 0 & 5 \\ 3 & 5 & 4 \end{matrix}$$

   ]

10. Scrivere un programma che dichiara tre array di interi `s1`, `s2` e `d`, il terzo dei quali di dimensioni pari alla somma delle dimensioni degli altri due (ad esempio se `s1` ha dimensione `N=9`, e `s2` ha dimensione `M=10`, `d` dovrà avere dimensione `N+M` cioè `19`).

    Il programma deve popolare sia `s1` che `s2` in modo random con interi nell'intervallo [0 , 255] (vedi Parte 3, Es. 9 per la generazione di numeri random). A questo punto, stampare il contenuto di `s1` e `s2` (vedi Parte 4, Es. 6 per la stampa su singola riga).

    Poi, il programma deve ordinare `s1` e `s2` in maniera crescente ([HINT: utilizzate il codice sviluppato negli esercizi precedenti in questa sezione! esempio selectionSort.cpp]).

    Infine riempire `d` in modo che contenga tutti gli elementi dei due array `s1` e `s2` ordinati tra loro in modo crescente.

    Alla fine il programma deve stampare `s1` e `s2` ordinati e l'array `d`.

    [File `mergeArrays.cpp`]

    Attenzione: nel costruire il contenuto di `d` tenete conto del fatto che i due array di partenza sono stati precedentemente ordinati. Se il merge dei due array `s1` e `s2` viene effettuato in modo sensato NON è necessario eseguire un ordinamento di `d`.

    Esempio di output:

```
Array s1 (generato) = { 103, 136, 48, 74, 180, 37, 142, 0, 2 }
Array s2 (generato) = { 167, 196, 204, 93, 66, 95, 49, 139, 222, 49 }
Array s1 (ordinato) = { 0, 2, 37, 48, 74, 103, 136, 142, 180 }
Array s2 (ordinato) = { 49, 49, 66, 93, 95, 139, 167, 196, 204, 222 }
Array d = { 0, 2, 37, 48, 49, 49, 66, 74, 93, 95, 103, 136, 139, 142, 167, 180, 196, 204, 222 }
```

11. Fissata una costante intera positiva $N = 10$, scrivere un programma che, preso in ingresso un numero intero positivo minore di 2^N (quindi nel caso specifico < 1024 e $>= 0$), memorizza la sua rappresentazione binaria su un array di lunghezza N. Quindi stampare il contenuto dell'array. Nel caso il numero non sia valido (quindi negativo o $>= 2^N$) stampare `"Numero NON valido"`.

    [File `dec2bin.cpp`]

    L'algoritmo per il calcolo della rappresentazione binaria è il seguente: si divide il numero in ingresso per 2 fino a che il risultato non è 0. La rappresentazione binaria è data dai resti delle divisioni nell'ordine inverso in cui sono stati calcolati. Quindi basta eseguire le divisioni finché non si arriva a 0 e memorizzare a ogni iterazione i resti della divisione intera in un array partendo dall'ultima posizione.

    Esempi di esecuzione (coppie input e output):

```
0 0000000000
1 0000000001
5 0000000101
16 0000010000
127 0001111111
128 0010000000
765 1011111101
1000 1111101000
1023 1111111111
1024 Numero NON valido
-1000 Numero NON valido
3456 Numero NON valido
-1 Numero NON valido
```

12. Scrivere il programma che esegua la traslazione verso sinistra degli elementi di un vettore letto in ingresso (di dimensione `D` fissata nel codice del programma), ovvero ogni elemento deve essere copiato in quello di indice immediatamente minore. Il ***valore del primo elemento deve essere perso***, quello dell'ultimo deve essere rimpiazzato da 0. Quindi stampa il risultato.

    Ad esempio: [1 10 15 18] ⟶ [10 15 18 0]

    [File `shiftLeft.cpp`]

13. Scrivere un programma che esegua la rotazione verso destra degli elementi di un array letto in ingresso (di dimensione `D` fissata nel codice del programma), ovvero ogni elemento deve essere copiato in quello di indice immediatamente maggiore, e il ***valore del primo elemento deve rimpiazzato*** da quello dell'ultimo. Quindi stampa il risultato.

    Ad esempio: [1 10 15 18] ⟶ [18 1 10 15]

    [File `rotateRight`]

14. Modificare `shiftLeft.cpp` e `rotateRight.cpp` in modo da ottenere programmi che leggano un numero intero N, positivo o negativo, ed eseguano rispettivamente la traslazione e la rotazione verso sinistra (se N negativo) o verso destra (se N positivo) di |N| posizioni.

    [File `shiftN.cpp`] [File `rotateN.cpp`]

    NOTA. Se |N| ≥ lunghezza array, shift creerà un vettore vuoto. Se |N| > lunghezza array, rotate si comporterà come se |N| fosse sostituito da |N % lunghezza array|. In particolare, se |N| == k×lunghezza array (k intero), rotate creerà un vettore uguale a quello dato, ovvero nessuna modifica.

15. Scrivere un programma che implementi il gioco del tris. La matrice di gioco dovrà essere rappresentata da una matrice 3x3. Implementare il gioco in modo tale che giochino 2 umani uno contro l'altro, alternativamente.

    [File `tris.cpp`]

    Potete trovare alcune informazioni sulle regole del gioco su `https://it.wikipedia.org/wiki/Tris_(gioco)`

5.3 Esercizi più avanzati

Questi esercizi sono abbastanza complessi e il testo breve vi lascia molta libertà su come implementarne la soluzione (e quindi vi fornisce anche poco aiuto). Potete provare a svolgerli solo se siete riusciti a completare e a comprendere a pieno i precedenti.

16. Scrivere un programma che implementi il gioco "Forza 4". [File `forza4.cpp`]

    Potete trovare alcune informazioni sulle regole del gioco su `https://it.wikipedia.org/wiki/Forza_quattro`

17. Modificare il programma del gioco del tris prevedendo che uno dei giocatori sia il computer (assumiamo che il PC giochi selezionando in modo casuale una delle caselle libere). [File `trisPC.cpp`]

18. Scrivere un programma che implementi il gioco "Mastermind". [File `mastermind.cpp`]

    Potete trovare alcune informazioni sulle regole del gioco su `https://it.wikipedia.org/wiki/Mastermind`

Parte 6

Struct

Impariamo ad utilizzare il tipo di dato struct, che ci permette di aggregare dati sia omogenei (stesso tipo) che non omogenei.

Cheatsheet

Creazione del *tipo* `tipo-struttura`:

```
struct tipo-struttura {
   dichiarazione-membro-1;
   dichiarazione-membro-2;
   dichiarazione-membro-3;
};
```

Le dichiarazioni sono normali dichiarazioni di variabili. Le variabili-membro si possono usare individualmente.
N.B.: Qui le dichiarazioni non possono contenere inizializzazioni!
N.B.: Un membro può a sua volta essere di un tipo struttura!

Dichiarazione di una *variabile* di tipo `tipo-struttura`:

```
struct tipo-struttura nome-variabile;
```

oppure

```
tipo-struttura nome-variabile;
```

N.B.: Qui `struct` è opzionale.

Uso dei membri:

```
data.giorno = 25;
data.mese = 12;
data.anno = 800;
if(data.giorno == 1) std::cout<<"Oggi inizia un nuovo mese\n";
```

Nota: No operazioni aggregate:
no `cout << data`, no `data++`

Però **OK assegnazione**: `data1 = data2`

Usare funzioni matematiche: in testa al file aggiungere `#include <cmath>`

`fabs(x)`	Valore assoluto (float abs)
`sqrt(x)`	Radice quadrata
`exp(x)`	Esponenziale in base e
`pow(x,y)`	Potenza (power), x^y
`log(x)` `log2(x)` `log10(x)`	Logaritmo in base e, 2, 10
`sin(x)` `cos(x)` `tan(x)`	Funzioni goniometriche
`ceil(x)` `floor(x)`	Arrotonda a intero per eccesso/per difetto

Operano tutte su dati di tipi `double` (anche `float` che viene convertito automaticamente in `double`); restituiscono `double`.

6.1 Esercizi di riscaldamento

1. Definire un tipo struct `Person` per rappresentare i dati relativi a una persona:

```
struct Person {
    std::string name;
    std::string surname;
    int birthYear;
};
```

   Dichiarare due variabili di tipo `Person`:

```
Person me, you;
```

Assegnare valori ai membri di `me` e di `you`:

```
me.name = "Bruce";
me.surname = "Wayne";
me.birthYear = 1939;

you.name = "Clark";
you.surname = "Kent";
you.birthYear = 1933;
```

Accedere (in lettura) ai valori dei membri, qui per stamparli:

```
std::cout << "My name is " << me.name << " " << me.surname << std::endl;
std::cout << "I was born in " << me.birthYear << std::endl;
```

Assegnare il valore di un'intera variabile struct a un'altra:

```
me = you;
```

Accedere (in lettura) ai valori dei membri, qui per stamparli:

```
std::cout << "My name is " << me.name << " " << me.surname << std::endl;
std::cout << "I was born in " << me.birthYear << std::endl;
```

[File `person.cpp`]

2. Definire un tipo struct `Point` per rappresentare punti su un piano cartesiano. La struct deve mantenere le coordinate di un punto:

```
struct Point {
        double x;
        double y;
};
```

   - Scrivere un programma che legge le informazioni relative a due `Point` P1 e P2 e, dopo aver verificato che non siano lo stesso punto, esprime la posizione di P1 rispetto a P2. [File `relativePos.cpp`]

```
// Stampare "Inserire le coordinate del punto P1: "
// Dichiarare una variabile P1 di tipo Point
// Leggere da input le coordinate e memorizzarle in P1.x e P1.y
// Stampare "Inserire le coordinate del punto P2: "
// Dichiarare una variabile P2 di tipo Point
// Leggere da input le coordinate e memorizzarle in P2.x e P2.y
// Se P1 e P2 sono lo stesso punto (ossia se hanno le stesse coordinate)
//      - Stampare "I punti sono uguali" seguito da un a capo
// Altrimenti
//      - Stampare "Il secondo punto è "
//      - Se   P2.y > P1.y
//              -- Stampare "in alto "
//      - Altrimenti
//              -- Stampare "in basso "
//      - Se   P2.x > P1.x
//              -- Stampare "a destra "
//      - Altrimenti
//              -- Stampare "a sinistra "
//      - Stampare " rispetto al primo" seguito da un a capo
```

[SUGGERIMENTO: Per verificare l'uguaglianza tra punti si può controllare il valore delle differenze tra le coordinate.]
NOTA. È opportuno considerare una tolleranza. Espresso in notazione matematica, questo significa verificare non se $x = y$, ma se $|x - y| < t$ con t positivo piccolo, una "tolleranza" entro cui decidiamo di considerare un valore praticamente pari zero.

- Scrivere un programma che legge le informazioni relative a due `Point` P1 e P2 e ne stampa la distanza.
  [File `dist.cpp`]

```
// Stampare "Inserire le coordinate del punto P1: "
// Dichiarare una variabile P1 di tipo Point
// Leggere da input le coordinate e memorizzarle in P1.x e P1.y
// Stampare "Inserire le coordinate del punto P2: "
// Dichiarare una variabile P2 di tipo Point
// Leggere da input le coordinate e memorizzarle in P2.x e P2.y
// Calcolare e stampare la distanza tra i due punti
```

[SUGGERIMENTO: dati due punti $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$, la loro distanza si calcola come $D(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.]

3. Definire una struct `StraightLine` per rappresentare l'equazione di una retta, completamente caratterizzata da coefficiente angolare e quota all'origine:

```
struct StraightLine {
        double m; // coefficiente angolare
        double q; // quota
};
```

Scrivere un programma che legge i parametri di una retta e li memorizza in una variabile di tipo `StraightLine`, poi legge le coordinate di un punto (memorizzato nella struct definita al punto 1.) e verifica se la retta passi o no per il punto.

```
// Stampare "Inserire i parametri  della retta R: "
// Dichiarare una variabile R di tipo StraightLine
// Leggere da input i parametri in R.m e R.q
// Stampare "Inserire le coordinate del punto P: "
// Dichiarare una variabile P di tipo Point
// Leggere da input le coordinate e memorizzarle in P.x e P.y
// Stampare il messaggio "La retta R di equazione y=mx+q "...
// ... (dove m e q saranno opportunamente sostituiti con R.m e R.q)
// Se la retta passa per il punto, ossia se il valore assoluto di...
// ... P.y - R.m*P.x - R.q è minore della tolleranza
//      - Stampare il messaggio " passa "
// Altrimenti
//      - Stampare il messaggio " non passa "
// Stampare il messaggio "per il punto di coordinate " ...
// ...seguito da P.x e P.y e da un a capo
```

[SUGGERIMENTO: per calcolare il valore assoluto di un float si usa la funzione fabs (per gli interi è abs).]

[File `retta.cpp`]

6.2 Esercizi di base

4. Definire una struct `Rect` per rappresentare un rettangolo mediante i vertici (`Point`) in alto a sinistra e in basso a destra.

   Scrivere un programma che legge in input due rettangoli, chiedendo le coordinate dei punti top_left e bottom_right di ciascuno; verifica se uno dei due rettangoli sia contenuto nell'altro; e stampa un messaggio di output opportuno.

   [File `rectanglesIn.cpp`]

5. Definire una struct `Date` per rappresentare date, ossia informazioni relative a giorno, mese ed anno (tutti memorizzabili con degli interi senza segno)

```
struct Date {
    unsigned int day;
    unsigned int month;
    unsigned int year;
};
```

   Scrivere un programma che legge la data corrente D una data qualsiasi D1. Dopo avere verificato la correttezza di D1 (per D assumiamo che sia inserita correttamente), controlla se D1 sia una data passata o futura e stampa un messaggio di output opportuno.

   [File `whenDate.cpp`]

   [SUGGERIMENTO: Per agevolare il controllo della correttezza della data conviene chiedere l'anno come primo dato in input, per verificare se sia o no bisestile. Tale controllo fornisce indicazioni sul controllo successivo, quello del mese e del giorno, anche se in casi limitati.]
   [SUGGERIMENTO: per verificare se la data sia passata o futura si può procedere per passi, controllando prima l'anno: se è minore dell'anno in corso allora la data è sicuramente passata, se è maggiore dell'anno in corso allora la data è sicuramente futura. Se è esattamente l'anno in corso allora occorre controllare il mese e, solo nel caso anch'esso non sia informativo (ossia uguale al mese corrente) passare al controllo del giorno.]

6. Definire una struct `Triangle` per rappresentare triangoli sul piano cartesiano con coordinate intere. `Triangle` includerà dunque tre campi che rappresentano altrettanti punti. `Triangle` avrà inoltre due ulteriori campi per memorizzare area e perimetro del triangolo.
   [SUGGERIMENTO: Per memorizzare i vertici del triangolo, potete usare una variante della struct `Point` del punto 1.]

   - Scrivere un programma che legge le coordinate dei 3 vertici di un triangolo e le memorizza in una struct di tipo `Triangle`. Verifica quindi che i tre vertici siano distinti. Poi calcola il perimetro e l'area del triangolo (vedi `https://it.wikipedia.org/wiki/Formula_di_Erone`) e memorizza le informazioni nei corrispondenti campi della struct. Infine stampa area e perimetro leggendoli dalla struct. [File `triangles.cpp`]
   - Scrivere un programma che legge e calcola le informazioni relative a 3 triangoli, memorizzate in altrettante variabili di tipo `Triangle`. Verificare poi quale dei tre triangoli abbia area maggiore e stampare un opportuno messaggio di output.
     [File `largestTriangle.cpp`]

7. Scrivere un programma che, dato un valore di N preimpostato ma modificabile, legge le coordinate di N punti in un array di Point (che rappresenta una spezzata o "polilinea" costituita da N-1 segmenti), e:

   - calcola e stampa la lunghezza totale della spezzata (vedi Esercizio 6.2, dist.cpp)
   - verifica se i lati hanno tutti la stessa lunghezza
   - verifica se la spezzata è chiusa (cioè se il primo e ultimo punto coincidono), e se lo è scrive un messaggio: "La linea è chiusa e quindi definisce un poligono" `https://it.wikipedia.org/wiki/Poligono`. Se i lati hanno tutti la stessa lunghezza, aggiungere: "regolare" `https://en.wikipedia.org/wiki/Regular_polygon`.

   [File `poly1.cpp`]

8. Partendo da esercizio precedente, implementare poi la seguente variante: se per una certo valore di N-1 il poligono ha un nome (N-1=3 "triangolo", N-1=4 "rettangolo", N-1=5 "pentagono"...) sostituire alla parola "poligono" il nome appropriato.
   [File `poly2.cpp`]

   [SUGGERIMENTO: Usare switch..case con caso default]

9. Definire una struct `Time` per mantenere informazioni orarie come terne ora, minuti, secondi (memorizzabili con degli interi senza segno).

   Scrivere un programma che legge le informazioni relative a due variabili T1, T2 di tipo `Time`. Poi verifica (1) la correttezza dei dati inseriti (cioè che i valori di ore, minuti e secondi siano ammissibili) e (2) che T1 rappresenti un'ora precedente (o uguale) a T2. In caso affermativo calcola il tempo trascorso tra i due orari, assumendo che si riferiscano allo stesso giorno (e.g., sono trascorse 2 ore, 5 minuti e 3 secondi). [File `timeDiff.cpp`]

6.3 Esercizi più avanzati

10. Definire un tipo struct `Student` per mantenere informazioni riguardanti uno studente, ed in particolare matricola, nome, cognome, data di nascita, voto medio.

    [SUGGERIMENTO: Usate delle stringhe (tipo `std::string`) per rappresentare nome e cognome.]

    Scrivere un programma che legga le informazioni relative ad almeno N studenti con $N > 2$, e le stampi in ordine decrescente di età.

    [SUGGERIMENTO: Usate array di struct.]

    [File `students.cpp`]

11. Definire un tipo struct `Complex` che rappresenta un numero complesso in due forme:

    (a) come parte reale e parte immaginaria (forma cartesiana);

    (b) come modulo e fase (forma esponenziale),

    tutte variabili-membro di tipo `double`.

    Scrivere un programma che legge due numeri complessi e ne calcola:

    - Somma
    - Differenza
    - Prodotto
    - Rapporto

    Ogni operazione deve mantenere *consistenti*, allineate fra loro, le due rappresentazioni, ovvero le coppie (re, im) e (modulo, fase) di un dato numero complesso devono sempre rappresentare lo *stesso* numero.

    Stampare i risultati delle operazioni nelle due forme, cartesiana ed esponenziale.

    [File `complexCalc.cpp`]

Parte 7

Funzioni

Impariamo a scrivere funzioni, l'elemento-base per costruire un programma complesso partendo da operazioni più semplici.

Per ciascuna funzione sarà necessario anche scrivere un `main` per poterla sottoporre a dei test ("testare"), ovvero chiamare con argomenti noti per verificare che il risultato restituito sia quello atteso.

NOTA. Ritroverete in questa parte molti esercizi che avete già svolto senza usare funzioni. D'ora in avanti, tenete presente che alcune funzioni sviluppate in un esercizio possano essere utili allo svolgimento di un esercizio successivo. Cercate di riutilizzare (ossia di richiamare) le funzioni già scritte, quando è appropriato.

Cheatsheet

Una funzione riceve in ingresso n **argomenti** o **parametri** (con n che può anche essere 0, nessun argomento) e **restituisce** in uscita un valore, oppure niente. Il tipo del valore restituito va dichiarato; nel caso "niente", il tipo è `void`.

Dichiarazione:

Esempio

```
int funz(float);
```

- Una funzione viene dichiarata scrivendo il suo **prototipo**
- Un prototipo può non contenere i nomi degli argomenti, ma deve contenerne il tipo. L'ordine conta.
- Un prototipo si può ripetere più volte nello stesso file (purché sempre identico, altrimenti rappresentano funzioni diverse)
- Solo un file in cui compare un prototipo può chiamare la corrispondente funzione

Definizione:

Esempio

```
int funz(float x) // intestazione
{
  // corpo = un code-block
}
```

- Una funzione viene definita scrivendo il suo codice
- Il codice di una funzione contiene una intestazione (simile a un prototipo) e un corpo
- Nell'intestazione, i nomi e i tipi degli argomenti **sono necessari**. L'ordine conta.
- Una funzione si deve definire una volta sola, pena errore
- All'interno di un sorgente, dopo che una funzione è stata definita si può usare anche senza prototipo. Se voglio usarla prima, devo aggiungere un prototipo prima.

Invocazione o chiamata:

Esempi:

```
a = funz(y);
a = funz2(x,y)+z;
std::cout << funz3() << std::endl;
```

MA ATTENZIONE: `funz4(x) = a; //ERRORE!`

- Un'invocazione di funzione è una espressione
- Una funzione viene chiamata con il nome seguito, tra parentesi, da tutti gli argomenti nell'ordine definito
- Più argomenti separati da virgole: es. `funz(x,y)`
- Se zero argomenti: parentesi vuote, ma comunque necessarie: es. `funz()`
- Una chiamata di funzione è ammessa dove è ammessa *in lettura* una variabile del tipo restituito dalla funzione
- Una chiamata di funzione **non** si può usare *in scrittura*, ovvero non le si può assegnare un valore. Non è un *lvalue*

Errori comuni:

- `return x;` in funzione che restituisce `void`
- `return;` in funzione che restituisce un tipo non-`void`
- omettere `return x;` in funzione che restituisce un tipo non-`void`
- dichiarare un argomento nella intestazione della funzione, e poi dichiarare una variabile con lo stesso tipo e nome *anche all'interno* del corpo della funzione
- leggere da `cin` il valore di un argomento di input
- modificare argomento dichiarato `const`
- passare argomenti in ordine sbagliato
- sperare che le modifiche fatte a un argomento (non passato come reference) siano conservate nel programma chiamante (approfondimento prossima parte eserciziario)

7.1 Esercizi di riscaldamento

1. **Alcune possibili funzioni** con varie combinazioni di tipi e argomenti.

   Copiare le seguenti funzioni in un file sorgente (nel caso stiate seguendo l'eserciziario in pdf si **sconsiglia** il copia-e-incolla, leggete e trascrivete)

   (a) Una funzione che non riceve alcun argomento e non restituisce alcun valore (tipo del valore restituito: `void`)

   ```
   void hello()
   {
     cout << "Hello, world\n";
   }
   ```

   (b) Una funzione che riceve un argomento di tipo `int` e non restituisce alcun valore (tipo del valore restituito: `void`)

   ```
   void hellomany(int n)
   {
     cout << "Hello, we are " << n << endl;
   }
   ```

   (c) Una funzione che non riceve alcun argomento e restituisce un valore di tipo `int`

   ```
   int givemefive()
   {
     return 5;
   }
   ```

   (d) Una funzione che riceve un argomento di tipo `int` e restituisce un valore di tipo `int`

   ```
   int prossimo(int n)
   {
     return n + 1;
   }
   ```

   (e) Una funzione che riceve due argomenti di tipo `int` e restituisce un valore di tipo `int`

   ```
   int somma(int a, int b)
   {
     return a + b;
   }
   ```

   Scrivere in testa al file sorgente le solite istruzioni (vedi parte introduttiva) e in coda un programma principale (funzione `int main()`) che fa il test delle cinque funzioni come segue:

   ```
   // chiamare la funzione hello
   // chiamare la funzione hellomany passandole come argomento il valore 5
   // chiamare la funzione givemefive stampando il valore restituito
   // chiamare la funzione prossimo passandole come argomento il valore 4 e stampando il valore restituito
   // chiamare la funzione somma passandole come argomenti i valori 2 e 3 e stampando il valore restituito
   ```

   [File `testf_basic.cpp`]

   [SUGGERIMENTO: Per stampare direttamente il valore restituito da una funzione si veda il cheatsheet, terzo esempio di invocazione]

2. Riscrivere il programma dell'esercizio precedente spostando la funzione `main` dopo i comandi iniziali, ma prima di tutte le altre funzioni. Verificare che la compilazione non ha successo. (Perché?) Modificare poi tale programma scrivendo i prototipi delle cinque funzioni subito prima della funzione `main`. I prototipi devono specificare quanto indicato qui di seguito:

   ```
   // funzione hello restituisce void e non richiede argomenti
   // funzione hellomany restituisce void e richiede un argomento di tipo int
   // funzione givemefive restituisce un valore di tipo int e non richiede argomenti
   // funzione prossimo restituisce un valore di tipo int e richiede un argomento di tipo int
   // funzione somma restituisce un valore di tipo int e richiede due argomenti di tipo int
   ```

   Riprovare a compilare e a eseguire.

   [File `testf_basic2.cpp`]

3. Scrivere una funzione che riceve un argomento intero `hm`, legge `hm` numeri reali e ne restituisce la media.

```
float average(int hm){
// se hm non è positivo
//      - stampare un messaggio di errore
//      - terminare il programma con exit(0)
// dichiarare una variabile sum di tipo float inizializzata a 0
/* iterare hm volte le seguenti istruzioni
        - stampare un a capo seguito dalla stringa "Inserisci un numero "
        - dichiarare una variabile x di tipo float
        - leggere  x
        - assegnare a sum la somma di sum e x
*/
// restituire il risultato della divisione di sum per hm
}
```

Scrivere un programma per testare la funzione secondo il seguente algoritmo:

```
// stampare la stringa "Di quanti numeri vuoi fare la media?"
// dichiarare una variabile how_many di tipo int
// leggere how_many
// dichiara una variabile avg (float) e inizializzarla con il risultato della chiamata di average su how_many
// stampare  un'andata a capo seguita dalla stringa "La media è "
// stampare avg
```

[File `testf_average.cpp`]

NOTA. "Chiamare una funzione f su x" è un modo colloquiale per intendere “chiamare una funzione f usando x come argomento”, ovvero fare la chiamata f(x).

4. Scrivere una funzione che dati due float `base` e `altezza`, restituisce l’area del rettangolo di base `base` e altezza `altezza`. La funzione deve verificare che base e altezza siano valori positivi ed in caso contrario ritorna rispettivamente -1 e -2 (ritornare -3 se entrambi invalidi). Testare i vari casi con input opportuni.

```
float area(float base, float altezza) {
// se base AND altezza non sono positivi
//      - ritornare -3
// se base non è positivo
//      - ritornare -1
// se altezza non è positivo
//      - ritornare -2
// restituire base x altezza
}
```

Scrivere un programma per testare la funzione `area`:

```
// dichiarare due variabili b e h di tipo float
// leggere b e h
// dichiarare la variabile float a
// chiamare la funzione area assegnando ad a il valore che restituisce
// se a è negativa gestire i vari casi con opportuni messaggi di errore
// altrimenti stampare l'area
```

[File `testf_area.cpp`]

5. Scrivere una funzione senza argomenti che legge lettere minuscole finchè l'utente non inserisce un carattere che non è una lettera minuscola, e restituisce l'ultima in ordine alfabetico (ovvero quella che numericamente è la massima).

```
char last_letter(){
// stampare la stringa "Inserisci una lettera minuscola "
// dichiarare una variabile last di tipo char
/* ripetere
        - leggere last
   fintanto che last minore di 'a' o maggiore di 'z'
*/
// dichiarare una variabile c di tipo char inizializzata con 'a'
/* ripetere
        - se c è maggiore di last
                -- assegnare il valore di c a last
        - stampare la stringa: "Inserisci una lettera minuscola (o altro carattere per terminare)"
        - leggere  c
   finché c è maggiore o uguale ad 'a' e minore o uguale a 'z'
*/
// restituire il carattere last
}
```

Scrivere un programma per testare la funzione `last_letter` secondo il seguente algoritmo

```
// dichiarare la variabile char last
// chiamare la funzione last_letter() assegnando a last il valore che restituisce
// stampare la stringa "La lettera più grande inserita è "
// stampare il contenuto di last
```

[File `testf_last_letter.cpp`]

6. Scrivere una funzione che dato come argomento un intero non negativo n restituisce come risultato il suo fattoriale.

   Il fattoriale di un numero è definito per induzione come $0! = 1$ e $(n+1)! = (n+1) * n!$. Quindi, ad esempio $3! = (2+1)! = 3*2! = 3*(1+1)! = 3*2*1! = 3*2*(0+1)! = 3*2*1*0! = 3*2*1*1$. In generale $n! = n*(n-1)*(n-2)*...*1$.

```
int factorial(int n){
// se n è minore di zero
//   - stampare un messaggio di errore pertinente
//   - ritornare -1
// se n è zero
//   - restituire 1
/* iterare su una variabile intera i inizializzata a n-1 e decrescente di 1 finché i è maggiore di 1
     - assegnare a n il prodotto di n e i
*/
// restituire n
}
```

Scrivere un programma per testare la funzione `factorial`:

```
// stampare la stringa "Inserire un numero positivo: "
// dichiarare una variabile intera num
// leggere num
// richiamare la funzione factorial su num e salvare il risultato
// se risultato non è negativo
//  - stampare il risultato seguito da " è il fattoriale di " seguito da num
// altrimenti stampare messaggio di errore
```

[File `testf_factorial.cpp`]

7. Scrivere una funzione chiamata `replicate`, che restituisce `void`. La funzione riceve come argomenti un numero intero `N`> 0 e un carattere `c`, e stampa `N` volte il carattere `c`. Se il numero non è strettamente positivo, la funzione non stampa niente.

```
void replicate (int N, char c){
   // iterare su i a partire da 1 e fino a N:
   //     - stampare c
}
```

Scrivere un programma che fa il test di questa funzione con `N` pari a 10, 1, 0, -10 e `c` un carattere a scelta.

```
int main (){
  // chiamare replicate con argomenti 10 e 'x'
  // stampare un a capo
  // chiamare replicate con argomenti 1 e 'x'
  // stampare un a capo
  // chiamare replicate con argomenti 0 e 'x'
  // stampare un a capo
  // chiamare replicate con argomenti -10 e 'x'
  // stampare un a capo
}
```

NOTA. Anche se questo è un caso banale, notate che abbiamo cercato di fare il test in tutte le condizioni possibili: condizione generica (10), condizione-limite ammessa (1), condizione-limite non ammessa (0), condizione generica non ammessa (-10).

[File `testf_replicate.cpp`]

8. Scrivere una funzione che preso come argomento numero intero strettamente positivo stampa un triangolo rettangolo fatto di `'*'` con lato lungo quanto il numero letto. Ad esempio, ricevuto come argomento il valore 5, stamperà:

```
*
**
***
****
*****
```

```
void triangle(int length){
// iterare su i a partire da 1 e fino a length:
//     - chiamare replicate su i e '*'
//     - stampare un a capo
}
```

Scrivere un programma per testare la funzione:

```
// stampare la stringa "Inserisci un numero maggiore di 0: "
// dichiarare una variabile len di tipo int
// leggere len
// se len e' positivo chiamare triangle su len
// altrimenti stampare un messaggio di errore
```

[File `testf_triangle.cpp`]

7.2 Esercizi di base

Per ciascun esercizio in questa sezione considerate quali valori sono accettabili come parametri, verificatene la correttezza nella funzione prima di usarli e segnalate eventuali errori.

9. Scrivere una funzione `isPrime` con un argomento *n* di tipo intero che restituisce `true` se *n* è *positivo e primo* oppure `false` negli altri casi (*negativo o non primo*). Dal main richiamare la funzione su un insieme di valori utili a verificarne il funzionamento (memorizzati in un array).

   [File `testf_prime.cpp`]

10. Scrivere una funzione `getLength` con un argomento num di tipo intero che restituisce il numero di cifre (in base 10). Ad esempio su 27458 restituisce 5. Dal main richiamare la funzione su un insieme di valori utili a verificarne il funzionamento (memorizzati in un array).

    [File `testf_ndigits.cpp`]

11. Scrivere una funzione `onlineAverage` senza argomenti che chiede all'utente di inserire e legge numeri reali sino a che l'utente vuole concludere l'inserimento. Quindi ad ogni ciclo oltre al numero reale chiede all'utente se vuole continuare e legge la risposta (`y` continua e `n` si ferma). Finito il ciclo di lettura restituisce la media dei numeri letti (di tipo `double`). Stampare quindi il valore della media nel `main`. NB: non è richiesto salvare i valori inseriti in un array, quindi NON usare alcun array.

    [File `testf_online_average.cpp`]

12. Scrivere una funzione `dist` che riceve come argomenti in ingresso due oggetti `p1` e `p2` di tipo struct `Punto`, definito come:

    ```
    struct Punto {
       double x, y;
    };
    ```

    La funzione calcola e restituisce in uscita la distanza euclidea tra i due punti, un valore di tipo double definito come da teorema di Pitagora (somma tra la differenza tra le coordinate x, al quadrato, e la differenza tra le coordinate y, al quadrato, il tutto sotto radice quadrata).

    Scrivere un programma che, fissato un valore ≥ 2 per una costante `N`, legge da input (`cin`) gli elementi di un array di `Punto` lungo `N`, che rappresenta una linea spezzata composta da `N`−1 segmenti, ne calcola la lunghezza totale `tot` usando la funzione `dist`, e stampa un messaggio del tipo "La lunghezza della spezzata è `tot`".

    [File `testf_poly1.cpp`]

13. Un evaporatore è una macchina in cui viene inserita una certa quantità iniziale di acqua e che ogni giorno ne disperde una percentuale prefissata nell'ambiente. Quando l'acqua contenuta scende sotto la soglia minima di funzionamento la macchina si spegne per evitare danni.

    Scrivere una funzione che presi come argomenti un `float` che rappresenta i litri di acqua inizialmente introdotti nella macchina, un `int` che rappresenta la percentuale di evaporazione giornaliera e un `float` che indica la soglia minima al di sotto della quale la macchina si spegne, restituisce il numero di giorni in cui la macchina può continuare ad operare senza essere riempita. Si assuma che tutti gli argomenti siano sempre non negativi.

    [File `testf_evaporator.cpp`]

7.3 Esercizi più avanzati

14. Partendo da quanto fatto nell'Esercizio 17 della Parte 3, scrivere una funzione con un argomento intero `n` che verifica se un numero intero positivo dato in input è un *numero di Armstrong* e se sì restituisce `true`, altrimenti restituisce `false`.
    Un intero positivo che si può rappresentare con *n* cifre (come minimo) si dice *numero di Armstrong* se è uguale alla somma delle potenze *n*-esime delle cifre che lo compongono. Ad esempio $153 = 1^3 + 5^3 + 3^3 = 1*1*1 + 5*5*5 + 3*3*3$ è un numero di Armstrong, come pure $1634 = 1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256$.

    [File `testf_armstrong.cpp`]

15. Partendo da quanto fatto nell'Esercizio 18 della Parte 3, scrivere una funzione con un argomento intero `n` che restituisce il numero di zeri alla fine del fattoriale (in base 10) del suo argomento **senza calcolarne il fattoriale**. Ad esempio su 5 stampa 1 perché $5 = 12\mathbf{0}$, mentre su 11 stampa 2 perché $11! = 39\,916\,8\mathbf{00}$.

    [File `testf_zeroes_factorial.cpp`]

16. Partendo da quanto fatto nell'Esercizio 19 della Parte 3, scrivere una funzione con un argomento intero `n` compreso fra 1 e 3000 e lo stampa in notazione romana.

    [File `testf_roman.cpp`]

Parte 8

Funzioni avanzate ed Eccezioni

Cheatsheet

Passaggio parametri

- *parametro passato per **valore***: un parametro formale che riceve una copia del contenuto del corrispondente parametro attuale
  `int incrementa(int num)`
- *parametro passato per **riferimento***: un parametro formale che riceve l'indirizzo di memoria (la locazione) del parametro attuale corrispondente
  `void incrementa(int& num)`
- *parametro passato come **costante***: in aggiunta alle due opzioni sopra è possibile indicare come costanti i parametri formali tramite la parola chiave `const` prima del tipo. In questo modo all'interno del corpo della funzione i parametri passati per costante non possono essere modificati
  `void stampaDatiPersona(const Persona& p)`

Trattamento errori

- `throw` interrompe esecuzione e segnala condizione eccezionale (es. errore)
- `try` indica parte del programma dove possono essere segnalate eccezioni
- `catch` indica parte del programma che fa qualcosa in risposta a una eccezione ricevuta

Uso `throw`:
`throw E` dove `E` valore, variabile od oggetto di tipo `T`
Uso `try`- `catch`:

```
try {
  // parte dove puo' esserci errore
}
catch (T1& a) {
  // parte dove si tratta il caso di T = T1
}
catch (T2& b) {
  // parte dove si tratta il caso di T = T2
}
catch (...) {
// parte dove si trattano tutti i casi non previsti sopra
}
```

Regola mnemonica: `catch(T& a)` è formalmente simile a una funzione: posso metterne diversi con lo stesso "nome" `catch`, purché il tipo dell'argomento sia diverso.

8.1 Esercizi di riscaldamento

1. Scrivere una funzione di nome `swapDouble` che prende due argomenti `a` e `b` di tipo `double`, e quando viene chiamata ne scambia i valori, e restituisce `void`.

   Passare i parametri per riferimento (quindi con `swapDouble(double& x, double& y)`). Provare a passare i parametri per valore, che cosa succede?

```
int main (){
   // dichiarare a e b di tipo double
   // stampare a e b
   // chiamare swapDouble(...) passando come parametri a e b
   // stampare a e b (per osservare risultato)
}
```

[File `testf_swapDouble.cpp`]

2. Scrivere una funzione di nome `swapPoint` che prende due argomenti `p1` e `p2` di tipo `Point` (una struct), e quando viene chiamata ne scambia i valori, e restituisce `void`.

```
struct Point {
    double x;
    double y;
};
```

Passare i parametri per riferimento `swapPoint(Point& x, Point& y)`. Provare a passare i parametri per valore, che cosa succede?

```
void swapPoint(Point& P1, Point& P2){
   // scambiare i valori x e y tra P1 e P2
}

void printPoint(const Point& P){
   // stampare le coordinate x e y di P
}
```

```
int main (){
   // dichiarare due punti P1 e P2
   // assegnare valori a scelta a P1 e P2
   // stampare P1 e P2 usando printPoint(...)
   // scambiare valori P1 e P2 usando swapPoint(...)
   // stampare P1 e P2 usando printPoint(...)  (per osservare risultato)
}
```

[File `testf_swapPoint.cpp`]

3. Scrivere una funzione che riceve un argomento intero `hm`, legge `hm` numeri reali e ne restituisce la media. Usare le eccezioni per gestire i casi di errore.

```
float average(int hm){
// se hm non è positivo
//      - dichiarare una variabile err di tipo int
//      - sollevare una eccezione con argomento err (throw err)
// dichiarare una variabile sum di tipo float inizializzata a 0
/* iterare hm volte le seguenti istruzioni
        - stampare un a capo seguito dalla stringa "Inserisci un numero "
        - dichiarare una variabile x di tipo float
        - leggere  x
        - assegnare a sum la somma di sum e x
*/
// restituire il risultato della divisione di sum per hm
}
```

Scrivere un programma per testare la funzione secondo il seguente algoritmo:

```
try {
    // stampare la stringa "Di quanti numeri vuoi fare la media?"
    // dichiarare una variabile how_many di tipo int
    // leggere how_many
    // stampare  un'andata a capo seguita dalla stringa "La media è "
    // stampare il risultato della chiamata di average su how_many
}
catch(int& err) {
    // stampare un messaggio d'errore
}
```

[File `testf_average_exception.cpp`]

4. Scrivere una funzione che dati due float `base` e `altezza`, restituisce l'area del rettangolo di base `base` e altezza `altezza`. La funzione deve verificare che base e altezza siano valori positivi ed in caso contrario sollevare una eccezione di tipo int.

```
float area(float base, float altezza) {
// se base AND altezza non è positivi
//      - dichiarare una variabile err di tipo int inizializzata a 3
//      - sollevare una eccezione con argomento err (throw err)
// se base non è positivo
//      - dichiarare una variabile err di tipo int inizializzata a 1
//      - sollevare una eccezione con argomento err (throw err)
// se altezza non è positivo
//      - dichiarare una variabile err di tipo int inizializzata a 2
//      - sollevare una eccezione con argomento err (throw err)
// restituire base x altezza
}
```

Scrivere un programma per testare la funzione `area`:

```
// dichiarare due variabili b e h di tipo float
// leggere b e h
try {
        // dichiarare la variabile float a
        // chiamare la funzione area assegnando ad a il valore che restituisce
        // stampare l'area
}
catch (int& err) {
        // stampare un messaggio che indica un errore sul valore della...
        // ... base (se err==1) o dell'altezza (se err==2) o di entrambi  (se err==3)
}
```

[File `testf_area_exception.cpp`]

5. Scrivere una funzione che dato come argomento un intero non negativo n restituisce come risultato il suo fattoriale.

   Il fattoriale di un numero è definito per induzione come $0! = 1$ e $(n+1)! = (n+1) * n!$. Quindi, ad esempio $3! = (2+1)! = 3*2! = 3*(1+1)! = 3*2*1! = 3*2*(0+1)! = 3*2*1*0! = 3*2*1*1$. In generale $n! = n*(n-1)*(n-2)*...*1$.

   Usare le eccezioni per gestire i casi di errore.

```
int factorial(int n){
// se n è minore di zero
//   - dichiarare una variabile err di tipo string ed inizializzarla con un messaggio di errore pertinente
//   - sollevare una eccezione con argomento err (throw err)
// se n è zero
//   - restituire uno
/* iterare su una variabile intera i inizializzata a n-1 e decrescente di 1 finché i è maggiore di 1
     - assegnare a n il prodotto di n e i
*/
```

```
// restituire n
}
```

Scrivere un programma per testare la funzione `factorial`:

```
try {
//   stampare la stringa "Inserire un numero positivo: "
//   dichiarare una variabile intera num
//   leggere num
//   stampare il risultato della chiamata di factorial su num seguito da " è il fattoriale di " seguito da num
}
catch(string& err) {
 //   stampare err, ossia il messaggio di errore
}
```

[File `testf_factorial_exception.cpp`]

6. Scrivere una funzione chiamata `replicate`, che restituisce `void`. La funzione riceve come argomenti un numero intero `N`> 0 e un carattere `c`, e stampa `N` volte il carattere `c`. Usare le eccezioni per gestire i casi di errore. In particolare la funzione `replicate` come segue. Dichiarare una struct `OutOfRangeError` con un campo stringa chiamato `functionName`, un campo stringa chiamato `paramName` e uno di tipo intero chiamato `paramValue`.

```
void replicate(int length, char c){
// se length non è maggiore di zero
//   - dichiarare una variabile err di tipo OutOfRangeError
//   - inizializzare il campo functionName di err con la stringa replicate
//   - inizializzare il campo paramName di err con la stringa length
//   - inizializzare il campo paramValue con il valore del parametro length
//   - sollevare una eccezione con argomento err (throw err)
/* iterare su i a partire da 1 e fino a length
    - stampare c
*/
}
```

Scrivere un programma per testare la funzione `replicate`:

```
try {
   // stampare la stringa "Inserisci un numero maggiore di 0: "
   // dichiarare una variabile len di tipo int
   // leggere len
   // stampare la stringa "Inserisci il carattere da replicare: "
   // dichiarare una variabile c di tipo char
   // leggere c
   // chiamare replicate su len e c
}
catch(OutOfRangeError& err) {
   // stampare un messaggio d'errore usando i campi di err
}
```

[File `testf_replicate_exception.cpp`]

7. Scrivere una funzione che preso come argomento numero intero strettamente positivo stampa un triangolo rettangolo fatto di `'*'` con lato lungo quanto il numero letto. Ad esempio, ricevuto come argomento il valore 5, stamperà:

```
*
**
***
****
*****
```

Come per l'esercizio precedente dichiarare una struct `OutOfRangeError` con un campo stringa chiamato `functionName`, un campo stringa chiamato `paramName` e uno di tipo intero chiamato `paramValue`. Questa struct sarà usata sia per segnalare errori in `replicate(...)` che in `triangle(...)`

```
void triangle(int length){
// se len non è maggiore di zero
//   - dichiarare una variabile err di tipo OutOfRangeError
//   - inizializzare il campo functionName di err con la stringa triangle
//   - inizializzare il campo paramName di err con la stringa length
//   - inizializzare il campo paramValue con il valore del parametro length
//   - sollevare una eccezione con argomento err (throw err)
/* iterare su i a partire da 1 e fino a length
    - chiamare replicate(...) su i e '*'
    - stampare un a capo
*/
}
```

Scrivere un programma per testare la funzione `triangle`:

```
try {
   // stampare la stringa "Inserisci un numero maggiore di 0: "
   // dichiarare una variabile len di tipo int
   // leggere len
   // chiamare triangle su len
}
catch(OutOfRangeError& err) {
   // stampare un messaggio di errore usando i campi di err
}
```

[File `testf_triangle_exception.cpp`]

8. Questo esercizio e il successivo illustrano l'uso di funzioni che chiamano altre funzioni. Alcune svolgono compiti elementari; questi compiti elementari vengono usati da altre funzioni per svolgere compiti più complessi, chiamando opportunamente le prime.

   Scrivere un insieme di funzioni che, opportunamente composte, permettono di ottenere la gestione di un menu. Verificare il funzionamento di ogni funzione con opportuni programmi.

   - Scrivere una funzione che presi come argomenti quattro stringhe, le stampa nell'ordine ricevuto, ciascuna su una nuova riga e preceduta da un numero progressivo.

     ```
     void print_menu(string choice1, string choice2, string choice3, string choice4){
        // stampare '1' seguito da un carattere tab seguito da choice1
        // stampare su una nuova riga '2' seguito da un tab seguito da choice2
        // stampare su una nuova riga '3' seguito da un tab seguito da choice3
        // stampare su una nuova riga '4' seguito da un tab seguito da choice4
     }
     ```

     Scrivere un programma per testare la funzione:

     ```
     // dichiarare una costante s1 di tipo string inizializzata con "Prima scelta"
     // dichiarare una costante s2 di tipo string inizializzata con "Seconda scelta"
     // dichiarare una costante s3 di tipo string inizializzata con "Terza scelta"
     // dichiarare una costante s4 di tipo string inizializzata con "Quarta scelta"
     // chiamare print_menu su s1, s2, s3, s4
     ```

     [File `testf_print_menu.cpp`]

   - Scrivere una funzione che prende come argomenti un intero `n`, compreso fra uno e quattro, e quattro stringhe e che stampa su una nuova riga il parametro stringa `n`-esimo preceduto dalla stringa `"Scelta effettuata: "`.

```
void print_choice(int n, string ch1, string ch2, string ch3, string ch4){
// Stampare un  a capo seguito da  "Scelta effettuata: "
// A seconda del valore di n
//   Nel caso 1:
//        - stampare ch1
//   Nel caso 2:
//        - stampare ch2
//   Nel caso 3:
//        - stampare ch3
//   Nel caso 4:
//        - stampare ch4
}
```

Scrivere un programma per testare la funzione:

```
// dichiarare una costante s1 di tipo string inizializzata con "Prima scelta"
// dichiarare una costante s2 di tipo string inizializzata con "Seconda scelta"
// dichiarare una costante s3 di tipo string inizializzata con "Terza scelta"
// dichiarare una costante s4 di tipo string inizializzata con "Quarta scelta"
// chiamare print_choice su 1, s1, s2, s3, s4
// chiamare print_choice su 2, s1, s2, s3, s4
// chiamare print_choice su 3, s1, s2, s3, s4
// chiamare print_choice su 4, s1, s2, s3, s4
```

[File `testf_print_choice.cpp`]

- Scrivere una funzione con un argomento intero `max` che chiede all'utente di inserire una scelta compresa fra uno e max finché l'utente non ne inserisce una accettabile e la restituisce.

```
int get_choice(int max){
// Dichiarare una variabile scelta di tipo int
/* Ripetere
      - Stampare "Inserisci una scelta fra 1 e  " seguito da max
      - Stampare un  a capo
      - Leggere scelta
    finché scelta minore di uno o maggiore di max */
// Restituire scelta
}
```

Scrivere un programma per testare la funzione:

```
// stampare il risultato della chiamata di get_choice su 7
```

[File `testf_get_choice.cpp`]

In tre esecuzioni successive provare a inserire `1`, `3`, una sequenza di più `8` conclusa da un `4`.

- Scrivere una funzione che, prese come argomenti quattro stringhe, le stampa nell'ordine ricevuto, ciascuna su una nuova riga e preceduta da un numero progressivo, chiede all'utente un intero `n` compreso fra uno e quattro e stampa su una nuova riga il parametro stringa `n`-esimo preceduto dalla stringa `"Scelta effettuata: "`:

```
int use_menu(string choice1, string choice2, string choice3, string choice4){
// Chiamare print_menu su choice1, choice2, choice3, choice4
// Dichiarare una variabile n di tipo int inizializzata con il risultato della chiamata di get_choice su 4
// Chiamare print_choice su n, choice1, choice2, choice3, choice4
// Restituire n
}
```

Scrivere un programma per testare la funzione secondo il seguente algoritmo

```
// dichiarare una costante s1 di tipo string inizializzata con "Prima scelta"
// dichiarare una costante s2 di tipo string inizializzata con "Seconda scelta"
// dichiarare una costante s3 di tipo string inizializzata con "Terza scelta"
// dichiarare una costante s4 di tipo string inizializzata con "Quarta scelta"
// chiamare use_menu su s1, s2, s3, s4
```

[File `testf_use_menu.cpp`]

9. Scrivere un programma, per testare le funzioni implementate nell'esercizio precedente, che propone all'utente un menu con quattro alternative, ne legge la scelta e seleziona l'alternativa corrispondente finché non viene selezionata l'alternativa quattro. Il programma deve comportarsi come descritto nel seguente algoritmo.

```
// dichiarare una costante s1 di tipo string inizializzata con "Prima scelta"
// dichiarare una costante s2 di tipo string inizializzata con "Seconda scelta"
// dichiarare una costante s3 di tipo string inizializzata con "Terza scelta"
// dichiarare una costante s4 di tipo string inizializzata con "Basta!"
/* ripetere
        - dichiarare una variabile intera answer inizializzata con use_menu su s1, s2, s3, s4
   finché answer è diverso da quattro
*/
```

[File `testf_menu.cpp`]

10. Scrivere una funzione con un parametro intero `k` che restituisce il numero ottenuto leggendo `k` da destra verso sinistra. Ad esempio su 17 restituisce 71, su 27 458 restituisce 85472 e così via.

```
int reverse(int k){
// dichiarare una variabile intera sign inizializzata con 1
// se k minore di zero
//    - assegnare -1 a sign
//    - assegnare -k a k
// dichiarare una variabile intera inv inizializzata a zero
/* finché k è maggiore di zero
    - dichiarare una variabile intera mod inizializzata con
      il resto della divisione intera di k per 10
    - assegnare a k il quoziente di k per 10
    - assegnare a inv la moltiplicazione di inv per 10
    - assegnare a inv la somma di inv e mod
*/
// restituire inv moltiplicato per sign
}
```

Scrivere un programma per testare la funzione:

```
// stampare la stringa "Inserire un numero intero: "
// dichiarare una variabile intera z
// leggere z
// stampare su una nuova riga la stringa "Rovesciando " seguita da z
// stampare la stringa " si ottiene " seguita dal risultato della chiamata di...
// ... reverse su z
```

[File `testf_reverse.cpp`]

8.2 Esercizi di base

Per ciascun esercizio in questa sezione considerate quali valori sono accettabili come parametri, verificatene la correttezza nel corpo della funzione prima di usarli e segnalate eventuali errori usando il meccanismo delle eccezioni, preferibilmente dichiarando delle opportune struct per rappresentarli e catturare le informazioni utili per segnalare l'errore.

11. Scrivere una funzione `void square(int n)` con un parametro di tipo intero che stampa un quadrato vuoto con i lati composti di `'x'` di dimensioni pari all'argomento. Se l'argomento è negativo o 0 non stampa nulla ma solleva un'eccezione di tipo string con un messaggio `"Errore valore < 1"`. Ad esempio con `n=7` stamperà:

```
x x x x x x x
x           x
x           x
x           x
x           x
x           x
x x x x x x x
```

[SUGGERIMENTO: IMPORTANTE: per rendere il risultato più graficamente simile a un quadrato ogni `x` (minuscola) andrà stampata usando la stringa `"x "` e ogni cella vuota come una stringa composta da due spazi `"  "`. Quindi ogni elemento stampato per comporre il quadrato è una stringa di due caratteri.]

[File `testf_square.cpp`]

12. Scrivere una funzione con due parametri di tipo intero che stampa il trapezio rettangolo fatto di `'x'` con le basi lunghe quanto gli argomenti, e l'altezza pari alla differenza fra le basi più uno. Ad esempio su 5 e 9 stamperà:

```
xxxxx
xxxxxx
xxxxxxx
xxxxxxxx
xxxxxxxxx
```

(che è alto $5 = 9 - 5 + 1$). Si noti che data la scelta dell'altezza a ogni riga bisogna stampare un carattere in più rispetto alla precedente.

[File `testf_trapezium.cpp`]

[SUGGERIMENTO: usare la funzione `replicate`.]

13. Scrivere una funzione con tre parametri di tipo float che li moltiplica fra loro, divide il risultato ottenuto per ciascuno degli argomenti in successione e restituisce un booleano che vale vero se il risultato dell'operazione è 1.

[File `testf_is_one.cpp`]

14. Scrivere una funzione `replicate2_line` con parametri `f`, `f_c`, `s`, `s_c`, dove `f` e `s` sono di tipo intero e `f_c` e `s_c` di tipo carattere. La funzione stampa su una riga a sé stante `f` volte `f_c`, seguito da `s` volte `s_c`. Ad esempio `replicate2_line(3, 's', 7, 'q')` stampa

```
sssqqqqqqq
```

Quando per un carattere viene inserito un numero <=0 quel carattere non viene stampato.

[File `testf_replicate2_line.cpp`]

15. Scrivere una funzione con un parametro `n` di tipo intero che stampa un rombo di asterischi che sulla diagonale ha `2*n+1` caratteri. Ad esempio, dato 8 stampa

```
        *
       ***
      *****
     *******
    *********
   ***********
  *************
 ***************
*****************
 ***************
  *************
   ***********
    *********
     *******
      *****
       ***
        *
```

che sulla diagonale ha 17 caratteri.

[File `testf_rhombus.cpp`]

[SUGGERIMENTO: 1)è più facile stampare il rombo con due cicli, il primo per le righe in cui il numero di asterischi cresce e il secondo per le righe in cui il numero di asterischi diminuisce.]

[SUGGERIMENTO: 2) volendo si può usare la funzione `replicate2_line`]

16. Scrivere una funzione con un argomento intero `n` che restituisce un booleano, `true` se `n` è *palindromo*, ovvero se le sue cifre (in base 10) lette da destra a sinistra corrispondono alle cifre lette da sinistra a destra (altrimenti restituisce `false`, ma questo è sottinteso visto che per una espressione di tipo `bool` sono possibili due soli valori).

    [File `testf_is_palindrome.cpp`]

    [SUGGERIMENTO: usare la funzione `reverse`.]

17. Scrivere una funzione con due argomenti reali `x` e `sqrt_x` che restituisce un valore booleano, `true` se `sqrt_x` è la radice quadrata di `x`, ovvero se il quadrato di `sqrt_x` coincide con `x`.

    Per testare la funzione usate come dati 25.3268614564 la cui radice quadrata è 5.03258 (se preferite altri valori, vi conviene partire da un numero con cifre decimali e farne il quadrato, in modo da evitare errori di approssimazione dovuti ai troncamenti).

    [File `testf_is_sqroot.cpp`]

18. Scrivere una funzione di nome `divide` che prende quattro argomenti interi `a`, `b`, `q` e `r`, restituisce `void`, quando viene chiamata assegna a `q` il quoziente tra `a` e `b` (risultato della divisione intera) e a `r` il resto.

    [File `testf_divide.cpp`]

19. La crescita della popolazione in una città può essere stimata a partire dalla popolazione iniziale, aumentata di una certa percentuale (le nascite al netto delle morti) e di un numero (le persone che ci si trasferiscono al netto di quelle che l'abbandonano).

    Scrivere una funzione che presi come argomenti un intero non negativo (la popolazione iniziale), la percentuale di nascite al netto delle morti come intero fra −100 e 100 e il numero di persone che si trasferiscono nella città al netto di quelle che l'abbandonano, restituisce un intero pari al numero di abitanti dopo un anno.

    Si noti che sia la percentuale di nascite al netto delle morti che il numero di persone che si trasferiscono nella città al netto di quelle che l'abbandonano possono essere negativi, positivi o nulli.

    Nei casi particolari in cui il risultato della simulazione fornisca un numero negativo la funzione ritorna zero.

    [File `testf_population_sim.cpp`]

    [SUGGERIMENTO: si noti che tutti i parametri sono interi, per cui usando moltiplicazione e divisione fra interi (nel giusto ordine) il risultato sarà ancora un intero.]

20. Analogamente al punto precedente, scrivere una funzione che prende, oltre ai parametri della funzione al punto 19, anche un intero che rappresenta un numero di anni e restituisce la popolazione dopo quel numero di anni.

    [File `testf_population_sim2.cpp`]

    [SUGGERIMENTO: Queste due funzioni possono essere implementate secondo tre approcci:

    – potete scrivere la prima e usarla ripetutamente (ciclo) per calcolare la seconda

    – oppure potete scrivere la seconda in maniera indipendente; in questo caso la prima contiene una chiamata alla seconda, essendo un caso particolare, in cui il numero di anni è uno.]

21. Scrivere una funzione `computeRectInfo(...)` con 4 argomenti di tipo double: `l1`, `l2`, `area`, e `perimentro`. I due ultimi parametri sono usati per fornire in output i valori di `area` e `perimento` del rettangolo di lunghezza `l1` e altezza `l2`. Nel caso in cui `l1` e/o `l2` siano negativi sollevare un'eccezione di tipo string con il messaggio opportuno (3 messaggi possibili).

    [SUGGERIMENTO: Ricordiamo che il passaggio per riferimento consente ad una funzione di "ritornare" più valori di output.]

    [File `testf_RectInfo.cpp`]

8.3 Esercizi più avanzati

22. Scrivere una funzione con un argomento intero `n` che stampa la scomposizione in fattori primi di `n`. Ad esempio su 392 stampa `"392 = 2^3 * 7^2"`, usando il carattere `'^'` per rappresentare l'elevamento a potenza.

    [File `testf_primefactors.cpp`]

23. Scrivere una funzione con argomenti interi `n` e `d`, con `d` compreso fra 0 e 9 e `n` maggiore di 10, che restituisce il più grande numero compreso fra 0 e `n` che nella sua rappresentazione in base 10 usa la cifra `d`. Ad esempio la sua chiamata con argomenti `3` per `d` e `15` per `n` restituisce `13` e la sua chiamata con argomenti `3` per `d` e `42` per `n` restituisce `39`.

    [File `testf_usedigit.cpp`]

    Riuscite a generalizzare questa funzione al caso in cui invece di cercare una singola cifra ne cerchiamo una sequenza? Ad esempio se cerco il più grande numero fra 0 e 400 che nella sua rappresentazione in base 10 contiene 39 il risultato sarà 399.

24. Scrivere una funzione che prende come argomenti un intero non negativo (la popolazione iniziale), la percentuale di nascite al netto delle morti come intero fra 0 e 100 e restituisce il numero di anni necessario a raddoppiare gli abitanti se la popolazione è in crescita, o a dimezzarli se la popolazione sta diminuendo (nell'ipotesi che non vi siano trasferimenti).

    [File `testf_population_sim3.cpp`]

25. Scrivere una funzione che presi come argomenti tre interi strettamente positivi: `a`, `b` e `max` restituisce la somma dei numeri divisibili per almeno uno fra `a` e `b` compresi fra 0 e `max`.

    [File `testf_sum_divisible.cpp`]

Parte 9

Puntatori (no allocazione dinamica di memoria) + Compilazione separata

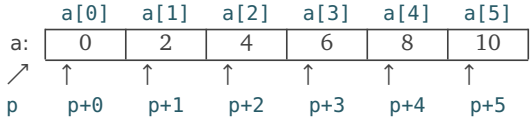
In questa sezione vedremo l'uso dei puntatori per accedere ad aree di memoria *già allocate altrimenti*, ad esempio variabili e parametri di funzioni.

Cheatsheet

» Puntatori «

- Un **puntatore** è una variabile atta a contenere come valore l'indirizzo di un'altra variabile. Un puntatore è sempre associato ad un tipo `T`.
- Definizione di una variabile di tipo "puntatore a un oggetto di tipo `T` ":
  - senza inizializzazione: `T* p;` Esempio: `int* p;`
    Meglio inizializzare le variabili, perché rischia di usarne il valore senza avergliene assegnato uno prima. Nel caso dei puntatori questo vuol dire accedere ad un'area di memoria a caso.
  - con inizializzazione: `T* p = <espressione-di-tipo-puntatore>;`
    Esempi: `char* q = p;` oppure `char* p = nullptr;` (`nullptr` è il puntatore nullo)

  Attenzione: Usare dichiarazioni separate per variabili separate: `T* p; T* q;`
- **Operatore di referenziazione &** (detto anche operatore "indirizzo di"). Data una variabile restituisce un valore indirizzo, corrispondente all'indirizzo in memoria della variabile.
  Esempio: `&a` permette di ottenere l'indirizzo di `a`. `float* p = &a;` dove `a` è una variabile di tipo `float`.
  Se abbiamo definito `T* p;` per assegnarle un valore useremo `p = &a;`.
  L'operatore di referenziazione `&` produce sempre un valore destro (l'indirizzo della variabile che non posso cambiare).
- **Operatore di dereferenziazione** `*`. Dato un puntatore, restituisce la variabile puntata dal puntatore.
  Esempi: `*p = 2;` (scrive 2 nell'area puntata da `p`); `a = *p + 4` (legge il valore contenuto nella memoria puntata da `p`, le somma 4 e assegna il risultato ad `a`)
  L'operatore di dereferenziazione `*` può essere usato sia come valore destro (il contenuto della variabile puntata da ...) che come valore sinistro (la variabile stessa).
- Dimensione in memoria di qualcosa che ha tipo `T`: `sizeof T`. esempio `sizeof int` (dimensione in bytes di un `int`) oppure `sizeof *int` (dimensione in bytes di un puntatore a `int` cioè di un indirizzo di memoria)
- Dimensione in memoria di una variabile `a`. esempio `sizeof(a)`
- **Aritmetica dei puntatori**: se `p` è un puntatore a un oggetto di tipo `T` ed `n` è un intero, allora `p+n` è l'indirizzo che si ottiene sommando `n` volte `sizeof(T)` all'indirizzo contenuto in `p`. Corrisponde a spostarsi, rispetto all'indirizzo puntato da `p`, di `n` elementi di tipo `T`. Esempio: `int a[6] = {0, 2, 4, 6, 8, 10};` `int* p = a;` (NB: un array è considerato la stessa cosa di un puntatore al primo elemento nell'array) si avrà:



dove ogni cella occupa quanto un intero = `sizeof int`.

Quindi l'istruzione `*(p+3)=7;` scrive 7 nella cella di indice 3 di a, sovrascrivendo il 6.

Analogamente, `cout << *(p+4);` stampa il contenuto della cella di indice 4 di a, ovvero stampa 8.

Uso frequente: usare un puntatore per scorrere un array, ad esempio per stamparlo:

```
for(int i=0;i<6;i++) cout<< *p++ <<endl;
```

N.B. `*(p++)` equivale a `*p++` perché l'operatore `++` ha la precedenza sul `*`

- **Passaggio di Array a Funzioni**: Il C++ non consente di passare un intero array come argomento di una funzione. Tuttavia, è possibile passare un puntatore all'array di tipo T e la dimensione dello stesso ad esempio nei seguenti modi:

```
void function(T* array, int size) { //corpo funzione }
void function(T array[], int size) { //corpo funzione }
```

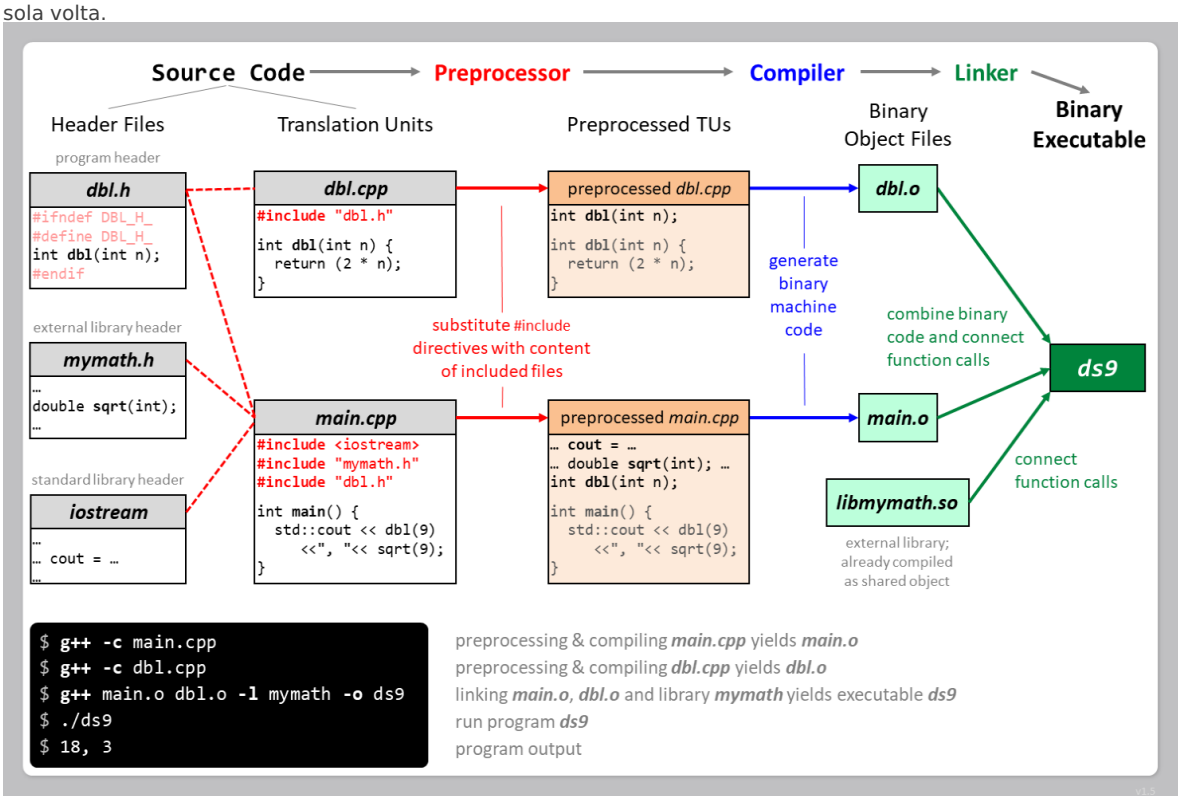
» Compilazione Separata «

Organizzare il codice di un programma in *file separati* rende lo stesso più facile da navigare e mantenere, e inoltre promuove la riutilizzabilità del codice. Le funzioni definite in diversi file sorgente (`.cpp`) possono essere *riutilizzate* in un altro programma includendo il relativo file di intestazione (`.h`). Questo favorisce la *creazione di componenti modulari e riutilizzabili*, che è fondamentale per ottenere un buon design del software.

Suddividere un programma in diversi file sorgente consente anche di effettuare la **compilazione separata**. La compilazione separata ottimizza il processo di compilazione. Infatti, quando si modifica un file sorgente e si ricompila, è sufficiente ricompilare solo quel file, e non l'intero programma. Questa strategia di compilazione è particolarmente vantaggiosa per progetti di grandi dimensioni, dove ricompilare tutto per una piccola modifica è molto dispendioso.

Creazione di File di Intestazione: In C++, i file di intestazione sono un componente chiave della compilazione separata. Contengono ad esempio i prototipi delle funzioni, le dichiarazioni delle struct etc, che informano il compilatore sulla struttura del codice senza rivelarne i dettagli implementativi. I file di intestazione solitamente hanno un'estensione `.h` e vengono inclusi nei file sorgente con la direttiva `#include` (vedi esempio sotto). Le Include Guard `#ifndef X #define X ... #endif` sono una parte fondamentale della creazione di file di intestazione. Impediscono che lo stesso file di intestazione venga incluso più di una volta all'interno di una singola unità di compilazione. Questo è cruciale perché includere un file più volte può causare errori di ridefinizione.

Per dettagli aggiuntivi analizzare la seguente figura dove è mostrato anche come compilare separatamente i vari files. Notare che la direttiva `#include "dbl.h"` è presente anche nel file `dbl.cpp` che contiene l'implementazione della funzione. In questo caso non sarebbe strettamente necessaria in quanto il `.h` contiene solo il prototipo della funzione (mentre è necessaria quando il `.h` contiene ad esempio la definizione di struct usate poi nelle funzioni). Aggiungere la direttiva `#include` del file `.h` nel file delle funzioni `.cpp` è considerata una buona pratica per evitare alcuni tipi di errori (ad esempio: disallineamento tra nomi/parametri delle funzioni tra il `.h` e il `.cpp`: con l'include vengono già rilevati durante la compilazione separata). Inoltre, se le funzioni nel `.cpp` si richiamano fra loro è necessario che i prototipi siano noti prima delle rispettive chiamate, quindi metterli tutti all'inizio includendo il `.h` lo garantisce. Un prototipo può comparire più volte (in quanto è una dichiarazione senza definizione) a differenza della definizione che deve comparire una sola volta.



Sommario: negli esercizi che svolgeremo in generale sarà sufficiente creare un file `.cpp` contenente il main, uno o più file `.cpp` contenenti ognuno l'implementazione di uno o più funzioni e infine uno o più file header `.h` contenente i prototipi delle funzione.
IMPORTANTE: Per compilare: `g++ -Wall -std=c++14 eser.cpp fun1.cpp fun2.cpp -o eser` e poi eseguire con `./eser`
oppure `g++ -Wall -std=c++14 eser*.cpp -o eser` (se tutti i file `.cpp` dell'esercizio hanno un prefisso comune, e.g. `eser`)

9.1 Esercizi di riscaldamento

1. (***Puntatori***) Scrivete un programma in cui usate i puntatori per accedere alle locazioni di variabili. Siete incoraggiati a migliorare i messaggi di stampa minimali proposti nell'esercizio in modo da capire più facilmente quali valori state stampando, ad esempio aggiungendo frasi come *indirizzo di s1 ==* oppure *valore di p ==*. A inizio esecuzione si ha `s1="Hello"` e `s2="World"` mentre a termine esecuzione `s1="Ciao"` e `s2="Mondo"`. Tutte le modifiche per sostituire le stringhe sono effettuate tramite il puntatore `p`. Analizzare i valori e gli indirizzi restituiti e verificare che il comportamento sia corretto.

   [File `provaptr.cpp`]

```
// dichiarare due variabili s1 e s2 di tipo string inizializzate rispettivamente a "Hello" e "World"
// stampare "Variabile s1: " seguito dal contenuto di s1 e l'indirizzo di s1
// stampare "Variabile s2: " seguito dal contenuto di s2 e l'indirizzo di s2

// dichiarare una variabile p di tipo puntatore a string inizializzata con l'indirizzo di s1

// stampare il messaggio "p contiene " ... " (un indirizzo)"
// stampare il messaggio "mentre con *p si ottiene " ... " (il contenuto della variabile puntata da p)"

// assegnare all'area di memoria puntata da p la stringa "Ciao"

// stampare il messaggio "p contiene " ... " (un indirizzo)"
// stampare il messaggio "mentre con *p si ottiene " ... " (il contenuto della variabile puntata da p)"

// assegnare a p l'indirizzo di s2

// stampare il messaggio "p contiene " ... " (un indirizzo)"
// stampare il messaggio "mentre con *p si ottiene " ... " (il contenuto della variabile puntata da p)"

// assegnare all'area di memoria puntata da p la stringa "Mondo"

// stampare il messaggio "p contiene " ... " (un indirizzo)"
// stampare il messaggio "mentre con *p si ottiene " ... " (il contenuto della variabile puntata da p)"

// stampare "Variabile s1: " seguito dal contenuto di s1 e l'indirizzo di s1
// stampare "Variabile s2: " seguito dal contenuto di s2 e l'indirizzo di s2
```

   [SUGGERIMENTO: Per stampare gli indirizzi (riferimenti a variabili e valore di puntatori) in maniera leggibile, è preferibile usare `static_cast<void*>(...)` mettendo l'indirizzo al posto dei puntini.]

2. (***Puntatori + Compilazione Separata***) Scrivere una funzione in un file dedicato che prende come argomenti 3 variabili di tipo char (per riferimento), propone all'utente di scegliere una di esse stampando il valore che contengono e restituisce l'indirizzo di quella scelta (ad esempio, per poterla modificare nel main). Se non viene scelta nessuna variabile ritorna `nullptr`. Se ad una richiesta non si risponde `'y'` la funzione termina ritornando il puntatore alla variabile che è stata scelta (o a `nullptr` se era la prima domanda).

```
char* selectVar(char& a, char& b, char& c) {
// definire un puntatore di tipo char p inizializzato al puntatore nullo

// stampare i messaggi "Scegli fra queste variabili" e
// "potrai cambiare idea in seguito e sceglierne una diversa che preferisci"

// stampare il messaggio "Vuoi la prima (y/n)? contiene " e stampare a
// dichiarare una variabile answer di tipo char
// leggere answer
```

```
// se la risposta è 'y' o 'Y'
//    assegnare a p l'indirizzo di a

// stampare il messaggio "Preferisci la seconda (y/n)? contiene " e stampare b
// leggere answer
// se la risposta è 'y' o 'Y'
//    assegnare a p l'indirizzo di b

// stampare il messaggio "Preferisci la terza (y/n)? contiene " e stampare c
// leggere answer
// se la risposta è 'y' o 'Y'
//    assegnare a p l'indirizzo di c

// restituire p
}
```

Scrivere un programma di test

```
// dichiarare tre variabili di tipo char ch1, ch2, ch3 inizializzate con lettere fra loro diverse
// definire un puntatore di tipo char selected inizializzato con la chiamata di selectVar su ch1, ch2 e ch3
// confrontare selected con l'indirizzo di ch1 e se sono uguali
// - stampare il messaggio "hai scelto ch1 che ha valore = " ...
// confrontare selected con l'indirizzo di ch2 e se sono uguali
// - stampare il messaggio "hai scelto ch2 che ha valore = " ...
// confrontare selected con l'indirizzo di ch3 e se sono uguali
// - stampare il messaggio "hai scelto ch3 che ha valore = " ...
// altrimenti
// - stampare "non hai scelto niente"
```

[File `selectVar_funct.h`] e [File `selectVar_funct.cpp`] per dichiarazione e definizione della funzione `selectVar` e [File `selectVar_main.cpp`] per i test.

[SUGGERIMENTO: notare che i file hanno prefisso comune, quindi compilare come suggerito nel cheatsheet di questa sezione.]

[SUGGERIMENTO: ragionare su che cosa succede se le tre variabili sono passate per valore invece che per riferimento.]

3. (***Aritmetica dei Puntatori***) Modificare il programma che legge `N` valori reali, li memorizza in un array `v` di lunghezza `N`, e ne restituisce la media richiesto dall'Esercizio 7 della Sezione 4 usando questa volta l'aritmetica dei puntatori per migliorarne l'efficienza, secondo il seguente schema (quindi non accedere mai all'array tramite `[indice]` ma solo via puntatore):

```
// dichiarare una costante N di tipo int inizializzata ad un valore strettamente maggiore di 0
// dichiarare un array v di N float
// dichiarare un puntatore a float p inizializzato con v

// *** Inserimento Valori Array ***
/* iterare su i a partire da 0 e fino a N-1
   - leggere un valore intero memorizzandolo nella cella puntata da p
   - incrementate p
*/

// *** Stampa Valori Array ***
// assegnare v a p (per ricominciare da capo a scorrere v)
// stampare l'array in ordine crescente da v[0] a v[N-1] su un'unica riga usando solo il puntatore p

// *** Calcolo Media ***
// dichiarare una variabile sum di tipo float e inizializzarla a zero
// assegnare v a p (per ricominciare da capo a scorrere v)
/* iterare su i a partire da 0 e fino a N-1
   - sommare il contenuto della cella puntata da p a sum
   - incrementare p
```

```
*/

// *** Stampa Media ***
// stampare la divisione di sum per N
```

[File `averageptr.cpp`]

4. (***Puntatori Array + Funzioni***) Quando si passa un `array` come argomento ad una funzione, il passaggio è *per riferimento* ovvero in realtà viene passato un *puntatore* all'indirizzo dove inizia l'`array` ed è questa la ragione per cui all'interno di una funzione non si conosce la dimensione dell'`array`. Vediamo in pratica questo aspetto implementando il seguente programma.

```
// dichiarare una costante N di tipo int inizializzata ad un valore strettamente maggiore di 0
// dichiarare la funzione
// void f(int* array) {
// stampare il messaggio "Dimensione del parametro = " seguito dalla dimensione di array e andare a capo
// }
```

Nel `main`

```
// dichiarare un array v di N interi
// dichiarare un puntatore a interi p inizializzato con v
// stampare il messaggio "Dimensione di v = " seguito dalla dimensione di v e andare a capo
// stampare il messaggio "Dimensione di p = " seguito dalla dimensione di p e andare a capo
// chiamare f su v
```

[SUGGERIMENTO: Per ottenere la dimensione di un valore usare `sizeof`.]

Nota. Notiamo che la dimensione dell'array nella funzione è persa, per questo motivo è necessario passarla come parametro aggiuntivo come mostrato nel cheatsheet di questa sezione.

[File `arrayptr.cpp`]

5. (***Puntatori Array + Compilazione Separata***) Scrivere una funzione `printArray`, che restituisce void, che prende due argomenti: un array di interi `s` da stampare e la sua dimensione `size`. Usare il puntatore all'array per scorrere tutti gli elementi dell'array (cioè con l'aritmetica dei puntatori).

   La stampa dell'array deve essere effettuata su una singola riga come visto nelle lezioni precedenti: ad esempio dato un array s di 5 elementi deve stampare:

   ```
   I valori contenuti nell'array s sono: { 3, 4, 5, 3, 4 }
   ```

   Poi implementare una seconda funzione `printArrayWithIndex` che invece di scorrere l'array con l'aritmetica dei puntatori usa un indice (cioè per stampare la cella i-esima esegue `s[i]`). La stampa attesa deve essere equivalente a quella restituita da `printArray`.

   [File `printArray_funct.h`] e [File `printArray_funct.cpp`] per dichiarazione e definizione della funzioni e [File `printArray_main.cpp`] per i test

```
void printArray(int* s, int size) {
// iterare size volte (cioè per i = 0, ..., size-1)
// -- stampare il contenuto dell'indirizzo puntato da s (usando *s)
// -- incrementare s
}

void printArrayWithIndex(int s[], int size) {
// iterare size volte (cioè per i = 0, ..., size-1)
// -- stampare s[i]
}
```

Realizzare un programma di test:

```
// testare il funzionamento di printArray con almeno tre array
// array con N elementi (es N=5)
// array con 1 elemento
// array con zero elementi

// replicare gli stessi passi per printArrayWithIndex e verificare che i risultati sono equivalenti
```

6. (***Puntatori Array + Compilazione Separata***) Riprendete la vostra implementazione della programma `reverse` dell'Esercizio 11 in Sezione 4 e creare una funzione `reverseArray` che presi in input due array di interi `source` e `dest` copia nel secondo array gli elementi del primo in ordine inverso. Poi provate a modificare il programma di test utilizzando `source` nel main per instanziare entrambi i parametri di `reverseArray`. Se avete implementato `reverseArray` nel modo ovvio, non otterrete il risultato atteso, perché copiando gli elementi dalla testa di `source` nella coda di `dest` in realtà state modificando la coda di `source` stesso. Per evitare questo problema, vogliamo premettere al corpo della funzione `reverseArray` il controllo che i due parametri corrispondano ad aree di memoria distinte e se invece coincidono sollevare un'eccezione.

   Nota. In questo specifico caso si potrebbe evitare il problema usando la stessa tecnica suggerita per l'esercizio 20 in sezione 4. Però, in situazioni più complesse potrebbe essere impossibile risolvere i problemi in caso di aliasing fra i parametri.

```
void reverseArray(int* source, int* dest, int size) {
//void reverseArray(int source[], int dest[], int size) {  // alternativa
   // se indirizzo di source uguale a indirizzo di dest sollevare eccezione di tipo a vostra scelta
   for (int i = 0; i < size; ++i) {
      // copio l'i-esimo elemento da source a dest nella posizione corretta
   }
}
```

   Realizzare un programma di test:

```
// definire un array di interi source inizializzato con N interi a scelta;
// definire un array di interi dest inizializzato con N interi zeri;

// chiamare printArray su source
// chiamare printArray su dest

// chiamare reverseArray su source e dest

// chiamare printArray su source
// chiamare printArray su dest
```

   Il programma di test quando esegue la chiamata `reverseArray(source, source, N)` deve essere in grado di catturare l'eccezione (stampate un messaggio di errore che dice che non si può chiamare `reverseArray` usando la stesso parametro attuale per entrambi i parametri formali).

   [File `reverseArray_funct.h`] e [File `reverseArray_funct.cpp`] per dichiarazione e definizione delle funzioni `reverseArray` e `printArray` (vedi esercizio precedente). [File `reverseArray_main.cpp`] per i test.

9.2 Esercizi di base

Anche dove non esplicitamente indicato, oltre a implementare le funzioni richieste dovete produrre anche un opportuno `main` per testarne la correttezza.

7. Scrivere una funzione `countDigitsInArray` che preso un `array v` di `N` elementi (dove `N` è una costante positiva a vostra scelta) di tipo `char` restituisce il numero di cifre, cioè caratteri nell'intervallo `['0','9']`, presenti in `v`, usando l'aritmetica dei puntatori per scorrere `v`. Ad esempio su `{'f','4','c','r','5','R'}` ritorna 2.

   [File `countDigitsInArray_funct.h`] e [File `countDigitsInArray_funct.cpp`] per dichiarazione e definizione della funzione e [File `countDigitsInArray_main.cpp`] per i test.

8. Scrivere una funzione di nome `isPresentInArray` che preso un `array t` di interi, la sua dimensione `size` e un intero `x` restituisce `true` se `x` è presente in `t` e `false` altrimenti. Usate l'aritmetica dei puntatori per scorrere `t`.

   [File `isPresentInArray_funct.h`] e [File `isPresentInArray_funct.cpp`] per dichiarazione e definizione della funzione e [File `isPresentInArray_main.cpp`] per i test.

9. Scrivere una funzione di nome `maxArray` che preso un `array t` di interi e la sua dimensione `size` restituisce il valore del suo elemento più grande. Se `size` vale `0`, la funzione solleverà un eccezione di tipo `string`. Usate l'aritmetica dei puntatori per scorrere `t`.

   [File `maxArray_funct.h`] e [File `maxArray_funct.cpp`] per dichiarazione e definizione della funzione e [File `maxArray_main.cpp`] per i test.

10. Scrivere una funzione di nome `allDiffArrayElements` che preso un `array t` di interi e la sua dimensione `size` restituisce `true` se `t` non contiene due elementi uguali e `false` altrimenti. Usate l'aritmetica dei puntatori per scorrere `t`.

    [File `allDiffArrayElements_funct.h`] e [File `allDiffArrayElements_funct.cpp`] per dichiarazione e definizione della funzione e [File `allDiffArrayElements_main.cpp`] per i test.

11. Scrivere una funzione di nome `orderedStringArray` che preso un `array t` di `string` e la sua dimensione `size` restituisce `true` se gli elementi di `t` sono ordinati nel ordine crescente (secondo l'ordine del dizionario) e `false` altrimenti. Usate l'aritmetica dei puntatori per scorrere `t`.

    [File `orderedStringArray_funct.h`] e [File `orderedStringArray_funct.cpp`] per dichiarazione e definizione della funzione e [File `orderedStringArray_main.cpp`] per i test.

12. Scrivere una funzione di nome `arrayIncludedInArray` che preso un `array t1` di interi, la sua dimensione `size1`, un `array t2` di interi, la sua dimensione `size2` restituisce `true` se tutti gli elementi di `t1` sono presenti in `t2` e `false` altrimenti. Usate l'aritmetica dei puntatori per scorrere `t1` e `t2`.

    [File `arrayIncludedInArray_funct.h`] e [File `arrayIncludedInArray_funct.cpp`] per dichiarazione e definizione della funzione e [File `arrayIncludedInArray_main.cpp`] per i test.

13. Scrivere una funzione `void shiftRightArray(int* t,int size)` che preso un `array t` di interi e la sua dimensione `size` scala tutti gli elementi di `t` di una posizione verso destra (il primo elemento diventa il secondo, il secondo diventa il terzo, etc) mentre l'ultimo elemento diventa il primo.

    [File `shiftRightArray_funct.h`] e [File `shiftRightArray_funct.cpp`] per dichiarazione e definizione della funzione e [File `shiftRightArray_main.cpp`] per i test.

14. Scrivere una funzione `void sortArrayInv(int* t, int size)` che preso un `array t` di interi e la sua dimensione `size` ordina gli elementi del array `t` in ordine decrescente.

    [File `sortArrayInv_funct.h`] e [File `sortArrayInv_funct.cpp`] per dichiarazione e definizione della funzione e [File `sortArrayInv_main.cpp`] per i test.

    [SUGGERIMENTO: Ricordatevi quello che si è visto nel esercizio 3 della sezione 5.]

9.3 Esercizi più avanzati

15. Modificare l'Esercizio 2 introducendo funzioni per i frammenti di codice ripetuti:

    - una funzione `proposeVar` che prende come argomenti il messaggio da visualizzare e la variabile proposta, stampa il messaggio, legge la risposta dell'utente e se questa è positiva (`'y'` o `'Y'`) restituisce l'indirizzo della variabile, altrimenti restituisce `nullptr`.
    - una funzione `printChoice` che prende come argomenti il puntatore (che contiene la scelta fatta), una variabile e una stringa contenente il nome della variabile, confronta l'indirizzo della variabile con il puntatore e se sono uguali stampa la stringa `"hai scelto "` seguita dal nome della variabile, altrimenti non fa nulla.

    Usare la prima funzione per migliorare il codice di `selectVar` e usare il programma di test originale per verificare di non aver introdotto errori. Poi usare la seconda funzione per migliorare il codice del programma di test e verificare che il comportamento non sia cambiato.

    [File `selectVarImproved_funct.h`] e [File `selectVarImproved_funct.cpp`] per dichiarazione e definizione della funzioni e [File `selectVarImproved_main.cpp`] per i test.

16. Considerate la funzione `selectVar` nella sua forma originale (Esercizio 2) e in quella migliorata (Esercizio 15). È possibile modificare il codice in modo da non usare puntatori?

    [File `selectVarImprovedNoPointers_funct.h`] e [File `selectVarImprovedNoPointers_funct.cpp`] per dichiarazione e definizione della funzioni e [File `selectVarImprovedNoPointers_main.cpp`] per i test.

17. Generalizzare la funzione `selectVar` in modo che proponga la scelta fra `N` variabili e non solo fra 3.

    [File `selectNVar_funct.h`] e [File `selectNVar_funct.cpp`] per dichiarazione e definizione della funzioni e [File `selectNVar_main.cpp`]

18. Come Esercizio 6 di questa sezione ma usardo array di stringhe incapsulati in struct come da esempio che segue:

```
// nel file di header .h

const int N = 5;

struct array_str {
   int size = N;
   int array[N];
};

// prototipi funzioni ...
```

```
// nel file delle funzioni .cpp

void reverseArrayInStruct(const array_str& source, array_str& dest) {
   // se indirizzo di source uguale a indirizzo di dest sollevare eccezione di tipo a vostra scelta
   // se size di source e dest non sono uguali sollevare eccezione di tipo a vostra scelta
   for (int i = 0; i < source.size; ++i) {
      dest.array[source.size - 1 - i] = source.array[i];
   }
}
```

```
// esempio di dichiarazione + inizializzazione di una struct di tipo array_str

array_str esempio = {5, {1,2,3,4,5}};
```

    [File `reverseArrayInStruct_funct.h`] e [File `reverseArrayInStruct_funct.cpp`] per dichiarazione e definizione delle funzioni `reverseArrayInStruct` e `printArray`. [File `reverseArrayInInStruct_main.cpp`] per i test.

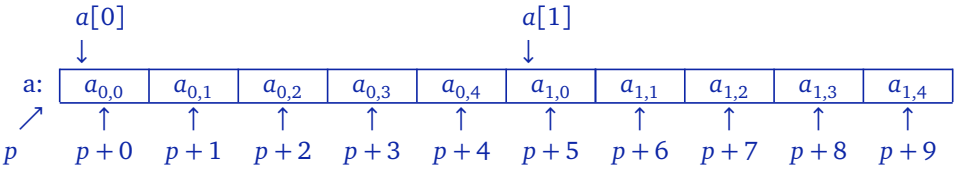
    [File `alias.cpp`]

    [SUGGERIMENTO: Usate un `array` per incapsulare le `N` variabili in un unico parametro; ricordatevi che i parametri di tipo `array` non conoscono la loro dimensione, quindi o incapsulate l'`array` in una `struct` (suggerito) o passate anche un parametro intero contenente la dimensione.
    Attenzione: non si possono fare `array` di valori di tipo riferimento, ma si possono fare `array` di valori di tipo puntatore.]

19. Scrivere una funzione `isUpper` che presa una matrice quadrata di caratteri restituisce `true` se tutti gli elementi contenuti sono lettere maiuscole, cioè caratteri nell'intervallo `['A','Z']`, `false` altrimenti usando l'aritmetica dei puntatori per scorrere la matrice.

    [File `testf_isupper.cpp`]

    [SUGGERIMENTO: Un `array` bidimensionale è un `array` di `array`; siccome gli elementi di un `array` sono memorizzati tutti di seguito, questo vuol dire anche gli elementi di un `array` bidimensionale sono memorizzati tutti di seguito come in questo esempio per il caso `int array[2][5] a`:



Quindi per scorrere tutti gli elementi di un vettore bidimensionale basta un unico puntatore incrementato sempre di uno anche per passare da una *riga* alla successiva]

[SUGGERIMENTO: Per inizializzare un puntatore all'indirizzo del primo elemento di un `array` basta assegnargli l'`array` stesso. Nel caso bidimensionale, però questo vuol dire avere un puntatore al tipo `array` interno e non all'elemento della matrice. Per ottenere un puntore al tipo della singola cella, quindi, bisogna assegnare al puntatore il primo elemento dell'`array`.]

20. Scrivere una funzione `diagonal` che presa una matrice quadrata di caratteri stampa gli elementi sulla diagonale usando l'aritmetica dei puntatori per visitare la matrice.

    [File `testf_diagonal.cpp`]

    [Hint: gli elementi sulla diagonale distano fra loro la lunghezza di una riga +1.]

Parte 10

Puntatori - allocazione dinamica di memoria + Valgrind

In questa sezione vedremo l'uso dei puntatori per *allocare dinamicamente memoria* e lavorarci.

Cheatsheet

Operazioni per allocare (ovvero riservare) memoria (su heap) e rilasciarla quando non serve più.

- **Per allocare la memoria necessaria per memorizzare un singolo elemento di tipo `T` si usa `new T`**

  Quindi ad esempio per allocare la memoria necessaria per memorizzare un `float` si usa `new float`, per allocare la memoria necessaria per memorizzare un `int` si usa `new int` e così via.

  Per evitare di allocare memoria a cui non si può accedere (che quindi sarebbe sprecata), quando si usa `new` bisogna sempre associare la memoria che si sta riservando ad un puntatore. Esempi tipici di uso:

  - *dichiarazione di puntatore con inizializzazione a una nuova area di memoria:* `T* p = new T;`

    Ad esempio nel caso `T` sia `float` avremo `float* p = new float;`

    nel caso `T` sia `int` avremo `int* p = new int;` e così via.
  - *assegnazione di una nuova area di memoria ad un puntatore già dichiarato:* `T* p; p = new T;`

    Ad esempio nel caso `T` sia `float` avremo `float* p; p = new float;`

    nel caso `T` sia `int` avremo `int* p; p = new int;` e così via.

- **Per allocare un blocco di memoria necessaria per memorizzare `N` elementi di tipo `T` si usa `new T[N]`**

  Ad esempio per allocare la memoria necessaria per memorizzare `7` `float` si usa `new float[7]`, per allocare la memoria necessaria per memorizzare 5 `int` si usa `new int[5]` e così via.

  Come nel caso in cui si riserva la memoria per un singolo elemento, anche quando si riserva un blocco contiguo per memorizzarvi `N` elementi bisogna sempre associare la memoria che si sta riservando ad un puntatore; il puntatore a cui viene assegnata l'area allocata (sia in fase di dichiarazione o con una assegnazione successiva) punta al primo elemento del blocco. Esempi tipici di uso:

  - *dichiarazione di puntatore con inizializzazione a una nuova area di memoria:* `T* p = new T[N];`

    Ad esempio nel caso `T` sia `float` e `N == 4` avremo `float* p = new float[4];`

    nel caso `T` sia `int` e `N == 11` avremo `int* p = new int[11];` e così via.
  - *assegnazione di una nuova area di memoria ad un puntatore già dichiarato:* `T* p; p = new T[N];`

    Ad esempio nel caso `T` sia `float` e `N == 9` avremo `float* p; p = new float[9];`

    nel caso `T` sia `int` e `N == 2` avremo `int* p; p = new int[2];` e così via.

  Bisogna fare molto attenzione a non perdere l'indirizzo iniziale di un blocco che si è allocato. Ad esempio se si vuole usare un puntatore per scorrerlo non si può usare quello impiegato per allocare la memoria (quanto meno non prima di averlo salvato altrove).

- **Per deallocare, ovvero rilasciare, la memoria precedentemente allocata con una `new` si usa `delete p` o `delete[] p`:**
  - `delete p`, dove `p` è il puntatore a cui è stata assegnata la memoria allocata, se è stato riservato spazio per un singolo elemento.
    Ad esempio `float* p = new float; delete p;` oppure `int* p; p = new int; delete p;`
  - `delete[] p`, dove `p` è il puntatore a cui è stata assegnata la memoria allocata, se è stato riservato spazio per più elementi. Ad esempio `float* p = new float[6]; delete[] p;`
    oppure `int* p; p = new int[3]; delete p[];`

Note

- Quando si crea *aliasing* fra puntatori, bisogna fare molta attenzione. Ad esempio, se facciamo `float* p = new float; q = p; /*...*/delete p;`
  - poiché è un errore fare due volte la `delete` della stessa area di memoria, l'istruzione `delete q;` causa errore.
  - il puntatore `q` punta ad un'area di memoria non più riservata. Se questa memoria viene riassegnata ad altri, usare `*q` può dare un errore, ma non è molto probabile. Nella maggior parte dei casi se la si usa per leggere avremo risultati inattesi (qualcuno ci ha scritto altri valori) e se la si usa per scrivere si modificano dati su cui stanno lavorando altri, causando verosimilmente errori in altre parti del programma.
- Dopo aver rilasciato la memoria puntata da un puntatore `p`, il puntatore si trova in uno stato pericoloso, perché punta ad un'area di memoria non più riservata per il nostro programma. È quindi buona norma seguire immediatamente una `delete` con un'assegnazione (eventualmente a `nullptr` se non si vuole riusare `p` con altro valore), a meno che `p` non sia una variabile locale che sta per essere eliminata all'uscita da uno scope (ad esempio la `delete` è l'ultima riga di una funzione e `p` è una variabile locale a quella funzione).

Valgrind è uno strumento per il debug di problemi di memoria e la ricerca dei memory leak. Per usarlo per verificare un programma C++ si possono seguire questi due passi:

1. Compilare il programma attivando le informazioni utili per il debug del programma (opzione `-g`): questo consente a Valgrind di fornire informazioni riguardo a quale LOC contiene la dichiarazione dell'oggetto che è stato coinvolto nel memory leak. Ad esempio:
   `g++ -Wall -std=c++14 -g leaks.cpp`
2. Eseguire poi Valgrind con il comando:
   `valgrind --leak-check=full --track-origins=yes -s ./a.out`
   Analizzare i risultati ottenuti.

Nota. Valgrind (http://www.valgrind.org/) ***rileva i memory leak solo sul codice effettivamente eseguito durante l'analisi***. Quindi se ad esempio un memory leak si trova in un ramo di un `if` che non è eseguito durante la verifica quel memory leak non sarà riportato. E' quindi importante ***cercare di testare in modo esaustivo i vari casi*** in modo da eseguire (coprire) il più possibile le varie porzioni del codice del vostro programma: in generale, pensate ad un insieme di test con input differenti che permettono di coprire i vari statement del vostro programma (criterio denominato ***Statement Coverage Testing***).

10.1 Esercizi di riscaldamento

1. Scrivere un programma che implementa il seguente algoritmo:

```
// dichiarare una costante N intera inizializzandola a un valore moderato (es. 5 o 10)
// dichiarare una variabile v di tipo puntatore a int
// allocare una quantità di memoria pari a N int, assegnandola a v
// scrivere nella memoria puntata da v la sequenza di valori  1, 3, 5, ... , 2*N-1 (i primi N dispari)
// stampare v usando l'aritmetica dei puntatori
// deallocare v
// allocare una quantità di memoria pari a 2*N int, assegnandola a v
// scrivere nella memoria puntata da v la sequenza di valori 1, 3, 5, ... , 4*N-3 (i primi 2*N dispari)
// stampare v usando l'aritmetica dei puntatori
// deallocare v
```

Nota. Fate attenzione a non perdere il riferimento a `v` nell'inizializzarne il contenuto o nella stampa. [File `alloc.cpp`]

2. Implementare un programma che causa dei memory leak e analizzarli. A tal fine usare usare **Valgrind** che trovate già installato sulle macchine di laboratorio.

```
// dichiarare una costante N intera inizializzandola a un valore moderato (es. 5)
// dichiarare due variabili v e p di tipo puntatore a int
// allocare una quantità di memoria pari a N int, assegnandola a v
// allocare una quantità di memoria pari a N int, assegnandola a p

// CASO 1 => unico caso corretto
// eseguire delete[] su v e p

// CASO 2
// allocare una quantità di memoria pari a N int, assegnandola a v
// eseguire delete[] su v e p

// CASO 3
// assegnare v a p
// eseguire delete[] su v

// CASO 4
// eseguire delete su v e p
```

Compilare il programma e analizzarlo con Valgrind seguendo la procedura descritta nel Cheatsheet di questa sezione.

Nel `CASO 1` Ci aspettiamo di ottenere un risultato simile a questo:

```
==1214== HEAP SUMMARY:
==1214==     in use at exit: 0 bytes in 0 blocks
==1214==   total heap usage: 4 allocs, 4 frees, 73,768 bytes allocated
==1214==
==1214== All heap blocks were freed -- no leaks are possible
==1214==
==1214== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Mentre degli altri casi ci saranno segnalati errori, ad esempio nel `CASO 2` ci aspettiamo di ottenere un risultato simile a questo, dato che il riferimento all'array di 5 `int` (che occupa 5 x 4 bytes) inizialmente puntato da `v` è stato perso:

```
==1242== HEAP SUMMARY:
==1242==     in use at exit: 20 bytes in 1 blocks
==1242==   total heap usage: 5 allocs, 4 frees, 73,788 bytes allocated
==1242==
==1242== 20 bytes in 1 blocks are definitely lost in loss record 1 of 1
==1242==    at 0x484A2F3: operator new[](unsigned long) (in .../valgrind/vgpreload_memcheck-amd64-linux.so)
==1242==    by 0x109234: main (leaks.cpp:10)
==1242==
==1242== LEAK SUMMARY:
==1242==    definitely lost: 20 bytes in 1 blocks
==1242==    indirectly lost: 0 bytes in 0 blocks
==1242==      possibly lost: 0 bytes in 0 blocks
==1242==    still reachable: 0 bytes in 0 blocks
==1242==         suppressed: 0 bytes in 0 blocks
==1242==
==1242== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

[File `leaks.cpp`]

10.2 Esercizi su libreria per array dinamici (d_array)

Obiettivo: in questa sezione dell'eserciziario implementeremo le funzioni per una mini-libreria che gestisca `array` dinamici. Per ciascuna funzione scrivere (ove possibile) un programma di test.

Files: aggiungere le funzioni ai file: [File `d_array.cpp, d_array.h`] – Fare programmi di test distinti, come specificato nei singoli esercizi, ciascuno dei quali usa le funzioni necessarie (quella da provare, più eventuali altre funzioni: per esempio, dovunque si usi un array sarà anche necessario crearlo).

Tipo di Dato: le funzioni opereranno su un `array` dinamico di double, ovvero un `array` la cui dimensione viene fissata al momento dell'esecuzione (invece che della dichiarazione). Per rappresentare gli `array` dinamici di double utilizzeremo il seguente tipo `dynamic_array` definito come segue:

```
struct dynamic_array {
        double* store;
        unsigned int size;
};
```

Usiamo un tipo struct per contenere sia l'array che la sua dimensione. L'array, tuttavia, in questo caso è realizzato usando un puntatore, che punta a un blocco di memoria di dimensione appropriata. Ricordate che, come visto nella Sezione 9, quando si ha un puntatore che punta a una zona di memoria che può contenere `N` elementi lo si può utilizzare con la stessa sintassi di un `array` per accedere ai suoi elementi, mediante l'indice fra parentesi quadre. Pertanto, l'uso è molto simile a quanto visto in precedenza per gli `array`. L'unica differenza notevole è che l'allocazione della memoria a un array è statica e immutabile, mentre quella ad un puntatore è dinamica e variabile.

Funzioni: elenco funzioni da implementare nella libreria `d_array` (contenuto del file `d_array.h`):

```
#ifndef D_ARRAY
#define D_ARRAY

//*****************************
//         d_array.h
//*****************************

struct dynamic_array {
   double* store;
   unsigned int size;
};

// PROTOTIPI delle funzioni da sviluppare per libreria d_array

void read_d_array(dynamic_array &d);
void print_d_array(const dynamic_array &d);
void delete_d_array(dynamic_array &d);
void create_d_array(dynamic_array &d, int size, double value);
void set_d_array_element(dynamic_array &d, unsigned int index, double value);
double get_d_array_element(const dynamic_array &d, unsigned int index);
void concat_d_arrays(const dynamic_array &t1, const dynamic_array &t2, dynamic_array &t3);
void sort_d_array(dynamic_array &d); // usando Selection Sort

#endif
```

3. **`void read_d_array(dynamic_array &d)`**: scrivere una funzione per la lettura di un `array` dinamico di double. Valore restituito: nessuno.

```
void read_d_array(dynamic_array &d) {
   // definire una variabile intera s a un valore negativo
   // finché s non è strettamente positiva
   {
      // stampare "Inserisci la dimensione dell'array" e andare a capo
      // leggere s
   }
   // assegnare s al campo size di d
   // allocare s double assegnando l'area di memoria riservata al campo store di d
   // iterare s volte
   {
      // stampare "inserisci un valore"
      // leggere un valore salvandolo nell'i-esimo elemento del campo store di d
      // (usando la notazione con le quadre per accedervi)
   }
}
```

Nota. Ha senso restituire al chiamante lo spazio allocato esplicitamente all'interno della funzione `read_d_array` perché resta riservato finché non lo deallochiamo esplicitamente. Non avrebbe senso invece restituire un riferimento ad una variabile locale, perché lo spazio corrispondente verrà rilasciato automaticamente alla fine della chiamata. Quindi non avrebbe avuto senso (anche se avessimo conosciuto la dimensione a compile time) scrivere qualcosa tipo `int a[s]; d.store = a;` perché lo spazio allocato per `a` non sopravviverà alla chiamata.

[File `vedi esercizio successivo`]

4. **`void print_d_array(const dynamic_array &d)`**: Scrivere una funzione per la stampa di un `array` dinamico di double. La stampa dell'array deve essere effettuata su una singola riga come visto nelle lezioni precedenti: ad esempio dato un array s di 5 elementi deve stampare:

```
I valori contenuti nell'array sono: { 3.56, 45.67, 105.32, 0.56, -4.943 }
```

```
void print_d_array(const dynamic_array &d) {
   // definire un puntatore p e inizializzarlo con il campo d.store
   // stampare gli elementi dell'array usando l'aritmetica dei puntatori su p (che è lungo d.size)
   // stampare una andata a capo
}
```

Usare questa funzione e la precedente per un programma di test che legge e stampa un `array` dinamico.

[File `testf_read_print_d_array.cpp`]

5. **`void delete_d_array(dynamic_array &d)`**: scrivere una funzione che riceve un solo argomento, il riferimento ad un `array` dinamico `d`, e lo *cancella*, ovvero se `d.size` è positivo rilascia lo spazio allocato per `d.store` e assegna zero a `d.size`. Altrimenti, solleva un'eccezione (di tipo opportuno, nel definirla tenete conto che si tratterebbe di una doppia delete dello stesso puntatore).

Nota. Nel programma di test potete facilmente verificare se due chiamate di `delete_d_array` successive su uno stesso `array` dinamico `d` (inizializzato con `read_d_array`) sollevano un'eccezione. Non avete invece modo di verificare da programma che lo spazio sia effettivamente stato rilasciato. Per farlo potete però usare programmi che controllano che non ci siano *memory leak*, come ad esempio **Valgrind** descritto neal Cheatsheet

[File `testf_delete_d_array.cpp`]

6. **`void create_d_array(dynamic_array &d, int size, double value)`**: scrivere una funzione che riceve tre argomenti: il riferimento ad un `array` dinamico `d`, la sua dimensione `size` (intero) e un valore `value` (double). La funzione inizializza `d` in modo che il suo `size` sia il `size` ricevuto come parametro e che il suo `store` sia un blocco di dimensione `size` in cui tutti gli elementi sono inizializzati al valore `value`.

Se al momento della chiamata `d` non è vuoto, ovvero il suo campo `size` non è nullo, prima di assegnare nuovi valori bisogna invocare `delete_d_array`.

Solleva eccezione in caso di dimensione negativa.

[File `testf_create_d_array.cpp`]

7. **`void set_d_array_element(dynamic_array &d, unsigned int index, double value)`**: scrivere una funzione che riceve tre argomenti: un riferimento ad un array dinamico `d`, un indice (intero) `index` ed un valore `value` (double). La funzione assegna `value` all'elemento con indice `index` di `d.store`.

   Solleva una eccezione (di tipo opportuno) se `index` non è un valore corretto rispetto a `v.size` (indice out-of-range se minore di zero o maggiore della dimensione).

   [File `testf_set_d_array_element.cpp`]

8. **`double get_d_array_element(const dynamic_array &d, unsigned int index)`**: scrivere una funzione che riceve due argomenti: un riferimento ad un array dinamico `d` costante (la `get` non modifica l'`array` dinamico) e un indice (intero) `index`.

   Solleva una eccezione (di tipo opportuno) se `index` non è un valore corretto rispetto a `v.size` (indice out-of-range se minore di zero o maggiore della dimensione).

   Valore restituito: il valore dell'elemento con indice `index` di `d.store`.

   [File `testf_get_d_array_element.cpp`]

9. **`void concat_d_arrays(const dynamic_array &t1, const dynamic_array &t2, dynamic_array &t3)`**: scrivere una funzione che riceve tre riferimenti ad array dinamici `t1`, `t2`, e `t3`. La funzione crea dinamicamente un nuovo array (da assegnare a `t3`) di lunghezza `t1.size+t2.size` dove nelle prime `t1.size` caselle si trovano gli elementi di `t1` e nelle prime `t2.size` caselle si trovano gli elementi di `t2`.

   [File `testf_concat_d_arrays.cpp`]

10. **`void sort_d_array(dynamic_array &d)`**: Scrivere una funzione che riceve un argomento, un riferimento a un array dinamico `d`.

    La funzione ordina l'array secondo l'algoritmo *SelectionSort* visto a lezione.

    [File `testf_sort_d_array.cpp`]

10.3 Esercizi su libreria my_vector

Obiettivo: in questa sezione produrremo una implementazione-giocattolo di `std::vector` chiamata `my_vector`.
Tipo di Dato: un oggetto di tipo `my_vector` è una struct che contiene tre membri:

- Un intero `size` che rappresenta il numero di elementi effettivamente memorizzati nel vettore
- Un intero `capacity` che rappresenta il numero massimo di elementi che possono essere memorizzati nel vettore
- Un puntatore a double `store`, che conterrà l'indirizzo di un'area di memoria allocata in modo da contenere `capacity` elementi di tipo double.

Files: le funzioni richieste sul tipo `my_vector` sono specificate negli esercizi seguenti, e vanno organizzate come segue:

- [File `my_vector.cpp`] => inserire qua **tutte le implementazioni delle funzioni** richieste
- [File `my_vector.h`] => già fornito nel seguito (contiene **definizione struct `my_vector` e i prototipi**)
- Come nella sezione precedente, è richiesto di creare un **file di test per ogni esercizio** (contenente il main e con il nome come indicato) e di **verificare che non ci siano memory leaks** con Valgrind (vedi comandi nel Cheatsheet).

Nota 1. Inizialmente non potrete sempre testare completamente le funzioni che create. Ad esempio quando creerete la funzione `print_my_vector`, sarete in grado solo di passare un `my_vector` vuoto. Man mano che sviluppate le funzioni successive sarete però in grado di testare anche le funzioni precedenti su casi diversi, come ad esempio la funzione `print_my_vector` su `my_vector` di varie dimensioni. Quindi ricordatevi di ***completare i test delle funzioni precedenti non appena avrete sviluppato le funzioni necessarie***.
Nota 2. Nella specifica delle funzioni richieste viene spesso indicato che un parametro di tipo `my_vector` è già inizializzato, o viceversa non lo è ancora. Questa indicazione è un *contratto* fra voi che implementate le funzioni e chi le chiamerà. Non potete verificare che venga rispettato, perché non potete controllare se l'area puntata da `store` sia stata allocata per il `my_vector` che state utilizzando, né che sia di dimensione pari a `capacity`. Potete solo controllare la consistenza fra `size` e `capacity` e che `size`

e `capacity` siano non negative. Un uso delle funzioni con `my_vector` che non sono stati inizializzati correttamente, quindi potrà causare errori impredicibili fra cui *segmentation fault* (se la funzione cerca di accedere ad un area di memoria che non è stata riservata per il `my_vector`).

Funzioni: elenco funzioni da implementare nella libreria `my_vector` (contenuto del file **`my_vector.h`** **NON modificare**):

```
#ifndef MY_VECTOR
#define MY_VECTOR

//****************************
//         my_vector.h
//****************************

struct my_vector {
   unsigned int size;
   unsigned int capacity;
   double* store;
};

// PROTOTIPI delle funzioni da sviluppare per libreria my_vector

my_vector create_my_vector(unsigned int capacity);
void print_my_vector_status(const my_vector &v);
void push_back_my_vector_element(my_vector &v, double x);
double pop_back_my_vector_element(my_vector &v);
void set_my_vector_element(my_vector &v, double value, unsigned int index);
double get_my_vector_element(const my_vector &v, unsigned int index);
void resize_my_vector(my_vector &v, unsigned int new_capacity);
void safe_push_back_my_vector_element(my_vector &v, double x);
void destroy_my_vector(my_vector &v);
bool looks_consistent_my_vector(const my_vector &v);
void sort_my_vector(my_vector &v);

#endif
```

Tests e controllo Memory Leaks: Esempio di `main` minimale che effettua una possibile esecuzione coinvolgendo le varie funzioni. Ad ogni passo stampa lo stato del `my_vector` per capire cosa sta succedendo. **Usate questo `main` quando avete implementato tutte le funzioni**, come check ulteriore. Chiaramente se il risultato della vostra implementazione è equivalente a quello mostrato in seguito ***possono esserci ancora vari errori e memory leaks***, ad esempio non viene testato il funzionamento di alcuna eccezione:

```
#include <iostream>
#include "my_vector.h"
using namespace std;

int main(){
   try {
      my_vector v = create_my_vector(3);                                          print_my_vector_status(v);
      push_back_my_vector_element(v, 1.23);                                       print_my_vector_status(v);
      push_back_my_vector_element(v, 45.6);                                       print_my_vector_status(v);
      double res = pop_back_my_vector_element(v); cout << "Ottenuto: "<< res << endl; print_my_vector_status(v);
      push_back_my_vector_element(v, -34.0);                                      print_my_vector_status(v);
      set_my_vector_element(v, 99.9, 1);                                          print_my_vector_status(v);
      res = get_my_vector_element(v,0);           cout << "Ottenuto: "<< res << endl; print_my_vector_status(v);
      resize_my_vector(v, 10);                                                    print_my_vector_status(v);
      resize_my_vector(v, 1);                                                     print_my_vector_status(v);
      safe_push_back_my_vector_element(v, 55.5);                                  print_my_vector_status(v);
      destroy_my_vector(v);                                                       print_my_vector_status(v);
   } catch (string e) {cout  << e << endl;}
   return 0;
}
```

Risultato atteso dall'esecuzione del main sopra:

```
**********************
my_vector size     = 0
my_vector capacity = 3
my_vector store = {  }
**********************
**********************
my_vector size     = 1
my_vector capacity = 3
my_vector store = { 1.23 }
**********************
**********************
my_vector size     = 2
my_vector capacity = 3
my_vector store = { 1.23, 45.6 }
**********************
Ottenuto: 45.6
**********************
my_vector size     = 1
my_vector capacity = 3
my_vector store = { 1.23 }
**********************
**********************
my_vector size     = 2
my_vector capacity = 3
my_vector store = { 1.23, -34 }
**********************
**********************
my_vector size     = 2
my_vector capacity = 3
my_vector store = { 1.23, 99.9 }
**********************
Ottenuto: 1.23
**********************
my_vector size     = 2
my_vector capacity = 3
my_vector store = { 1.23, 99.9 }
**********************
**********************
my_vector size     = 2
my_vector capacity = 10
my_vector store = { 1.23, 99.9 }
**********************
**********************
my_vector size     = 1
my_vector capacity = 1
my_vector store = { 1.23 }
**********************
**********************
my_vector size     = 2
my_vector capacity = 2
my_vector store = { 1.23, 55.5 }
**********************
**********************
my_vector size     = 0
my_vector capacity = 0
my_vector store = {  }
**********************
```

11. `my_vector create_my_vector(unsigned int capacity)`: scrivere una funzione che ritorna una struttura `my_vector v` con `v.store` allocato con lunghezza `capacity` (che deve essere positivo), assegna `v.capacity = capacity`, e assegna `v.size = 0`. Se `capacity` non è strettamente positiva, la funzione solleverà un'eccezione di tipo `string` contente il messaggio:

`"create_my_vector: capacity fornita non positiva"`.

[File `testf_create_my_vector.cpp`] (solo per il main con i test, NON la funzione che va in `my_vector.cpp`)

12. **`void print_my_vector_status(const my_vector &v)`**: Scrivere una funzione per la stampa delle informazioni associate a un `my_vector`. Quindi la funzione stampa nelle due righe successive la size e la capacity poi stampa `v.store` effettuata su una singola riga come visto nelle lezioni precedenti. Ad esempio dato un `my_vector` di size = 5 elementi e capacity = 10 deve stampare:

```
**********************
my_vector size = 5
my_vector capacity = 10
my_vector store = { 3.56, 45.67, 105.32, 0.56, -4.943 }
**********************
```

Se `my_vector` è vuoto sarà stampato:

```
**********************
my_vector size = 0
my_vector capacity = 0
my_vector store = { }
**********************
```

[File `testf_print_my_vector.cpp`] (solo per il main con i test, NON la funzione che va in `my_vector.cpp`)

13. **`void push_back_my_vector_element(my_vector &v, double x)`**: scrivere una funzione che riceve due argomenti, un riferimento a un `my_vector v` inizializzato e un double `x`.

    Se `v` è già completamente pieno (cioè `v.size == v.capacity`), la funzione solleva un'eccezione di `string` contente il messaggio `"push_back_my_vector_element: Impossibile inserire elemento in Array pieno"`.

    Altrimenti, se `v.size` è minore di `v.capacity`, la funzione inserisce `x` in `v.store` all'indice `v.size` e incrementa di uno `v.size`. In pratica aggiunge l'elemento in fondo all'array.

    [File `testf_push_back_my_vector_element.cpp`] (solo per il main con i test, NON la funzione che va in `my_vector.cpp`)

14. **`double pop_back_my_vector_element(my_vector &v)`**: scrivere una funzione che riceve un riferimento a un `my_vector v` inizializzato.

    Se `v.size` è positivo, la funzione restituisce il valore memorizzato in `v.store` all'indice `v.size-1` e decrementa di uno `v.size`. In pratica ritorna l'elemento in fondo all'array e lo "elimina".

    Se `v` è vuoto (cioè `v.size == 0`), la funzione solleva un'eccezione di tipo `string` contente il messaggio `"pop_back_my_vector_element: Impossibile estrarre elemento da Array vuoto"`.

    [File `testf_pop_back_my_vector_element.cpp`] (solo per il main con i test, NON la funzione che va in `my_vector.cpp`)

15. **`void set_my_vector_element(my_vector &v, double value, unsigned int index)`**: scrivere una funzione che assegna `value` all'elemento con indice `index` di `v.store`. Solleva una eccezione:

    - se `v.size==0` in quanto non è possibile impostare alcun elemento in un `my_vector` vuoto. Eccezione di tipo `string` contente il messaggio `"set_my_vector_element: Impossibile impostare un elemento di un My_Vector vuoto"`;
    - altrimenti se `index` non è un valore corretto, ovvero se non è compreso fra `0` e `v.size-1`. Eccezione di tipo `string` contente il messaggio `"set_my_vector_element: Indice fornito non ammissibile"`.

    Nota: `v.size` è di tipo `unsigned int`. Se il `my_vector` è vuoto abbiamo `v.size==0` allora `v.size-1==4,294,967,295` cioè `2^32-1` se per gli int sono utilizzati 4 bytes (cioè 32 bits). Quindi la prima eccezione vi semplifica la gestione della seconda dato che vi evita di avere il caso `v.size==0` che vi complicherebbe il check dell'intervallo `[0, v.size-1]`.

    [File `testf_set_my_vector_element.cpp`] (solo per il main con i test, NON la funzione che va in `my_vector.cpp`)

16. **`double get_my_vector_element(const my_vector &v, unsigned int index)`**: scrivere una funzione che restituisce il valore presente in `v.store` all'indice `index`.

    Solleva una eccezione:

- se `v.size==0` in quanto non è possibile leggere alcun elemento da un `my_vector` vuoto. Eccezione di tipo `string` contente il messaggio `"get_my_vector_element: Impossibile leggere un elemento da un My_Vector vuoto"`;
- altrimenti se `index` non è un valore corretto, ovvero se non è compreso fra `0` e `v.size-1`. Eccezione di tipo `string` contente il messaggio `"get_my_vector_element: Indice fornito non ammissibile"`.

[File `testf_get_my_vector_element.cpp`] (solo per il main con i test, NON la funzione che va in `my_vector.cpp`)

17. **`void resize_my_vector(my_vector &v, unsigned int new_capacity)`**: scrivere una funzione che riceve due argomenti: un riferimento a un `my_vector v` già inizializzato, un intero `new_capacity` strettamente positivo.

    La funzione modifica la capacità del vettore `v` preservando i valori contenuti per quanto possibile (ovvero quando la capacità viene mantenuta o aumentata, non quando viene diminuita).

    La funzione deve quindi

    - allocare un blocco lungo `new_capacity`,
    - se `v.size > 0` copiarvi i valori contenuti in `v.store` (se ci stanno, mentre se `v.size>new_capacity` verranno copiati solo i primi `new_capacity`)
    - se `v.capacity > 0` allora liberare lo spazio precedentemente occupato da `v.store` (per evitare *memory leak*)
    - assegnare a `v.store` il nuovo blocco di memoria.
    - aggiornare `v.capacity` a `new_capacity` (e `v.size` (se necessario)).

    Se `new_capacity` non è strettamente positiva, la funzione solleverà un'eccezione di tipo `string` contente il messaggio `"resize_my_vector: new_capacity fornita non positiva"`.

    [File `testf_resize_my_vector.cpp`] (solo per il main con i test, NON la funzione che va in `my_vector.cpp`)

18. **`void safe_push_back_my_vector_element(my_vector &v, double x)`**: scrivere una funzione analoga alla `push_back`, ma che in caso il `my_vector`:

    - abbia `capacity==0` invece di sollevare un'eccezione ridimensiona l'array con capacity=1;
    - sia già completamente riempito (cioè `capacity==size`) invece di sollevare un'eccezione raddoppia la capacità dell'array;

    e poi in entrambi i casi inserisce il valore dato (che a questo punto può trovare posto nello `store`).

    [Hint: utilizzate la funzione `push_back_my_vector_element` in un blocco try, catturate l'eccezione che sollevate in caso di mancanza di spazio e nel corrispondente catch chiamate in ordine la funzione `resize` per raddoppiare la capacità (o porla pari a 1) e poi nuovamente la funzione `push_back_my_vector_element` per inserire l'elemento che a questo punto vi troverà posto.]

    [File `testf_safe_push_back_my_vector_element.cpp`] (solo per il main con i test, NON la funzione che va in `my_vector.cpp`)

19. **`void destroy_my_vector(my_vector &v)`**: scrivere una funzione che dealloca `v.store` e pone a zero sia `v.size` che `v.capacity` nel caso in cui `v.capacity > 0`. Invece se `v.capacity == 0` la funzione non fa nulla.

    [File `testf_destroy_my_vector.cpp`] (solo per il main con i test, NON la funzione che va in `my_vector.cpp`)

20. **`bool looks_consistent_my_vector(const my_vector &v)`**: scrivere una funzione che restituisce `true` se `v.store` non è nullo, `v.size` non è negativa né maggiore di `v.capacity`, e `v.capacity` è positiva.

    Nota. se `looks_consistent_my_vector` restituisce `false` su un `my_vector v`, allora sicuramente `v` non è stato correttamente inizializzato. Se restituisce `true` è possibile che `v` sia corretto, ma anche che non lo sia perché non c'è consistenza fra `v.store` e `v.capacity`; ad esempio può essere che `v.store` punti ad un'area di memoria non allocata, oppure ad un'area allocata ma più piccola o più grande di `v.capacity`.

    [File `testf_looks_consistent_my_vector.cpp`] (solo per il main con i test, NON la funzione che va in `my_vector.cpp`)

21. **`void sort_my_vector(my_vector &v)`**: scrivere una funzione che riceve un riferimento a un `my_vector v`.

    La funzione ordina il suo argomento secondo l'algoritmo *SelectionSort*. Per chi vuole, come esercizio avanzato, provare ad implementare il sorting seguendo l'algoritmo *BubbleSort*.

    [File `testf_sort_my_vector.cpp`] (solo per il main con i test, NON la funzione che va in `my_vector.cpp`)

10.4 Esercizi più avanzati

22. Create una nuova versione della libreria (nome file specificato qui sotto). In questa aggiungete un controllo a tutte le funzioni precedenti **ad eccezione di `create`**, in modo che come prima cosa verifichino se il loro argomento di tipo `my_vector` è evidentemente scorretto (cioè se `looks_consistent(v)` è falso) e in tal caso sollevino un'eccezione di tipo opportuno.

    [File `safe_my_vector.cpp, safe_my_vector.h, testf_safe_my_vector.cpp`]

Parte 11

Vector + I/O su Files

In questa sezione ci concentriamo sull'utilizzo degli `vector`, imparando a maneggiarli e a definire funzioni che ne facciano uso. Inoltre sarà mostrato come leggere e scrivere informazioni da/su files.

NOTA. Dove necessario, gestire gli errori con la definizione di opportune eccezioni.

Cheatsheet

`#include <vector>`	header per i Vector
`vector<T> V`	Dichiara la variabile `V` di tipo `vector` con elementi di tipo `T`
`vector<T> W(V)`	Crea `W` e lo inizializza con il contenuto di un altro `vector<T> V` ("costruttore di copia")
`vector<T> V = {...}`	Crea un `vector V` di `T` e lo inizializza con gli elementi dell'array specificati (che devono essere di tipo `T`)
`vector<T> V(n)`	Crea un `vector` di `n` elementi inizializzati a un valore "nullo" predefinito che dipende dal tipo T (es. 0 se `T=int`, 0.0 se `T=float`, stinga vuota `""` se `T=string`)
`vector<T> V(n, val)`	Crea un `vector V` di `n` elementi inizializzati ad un valore `val`
`V.size()`	Ritorna la lunghezza (size) di `V` (cioè il numero di elementi che contiene)
`V.capacity()`	Ritorna lo spazio occupato da `V`, include spazio non ancora utilizzato
`V.max_size()`	Ritorna la dimensione massima possibile
`V.empty()`	Ritorna `true` se il vector è vuoto e `false` altrimenti
`V[i]`	Accede all'elemento `i`-esimo (segmentation fault se `i>=V.size()` o `i<0`)
`V.at(i)`	Accede all'elemento `i`-esimo (ma con eccezione gestibile se `i>=V.size()` o `i<0`)
`V.front()`	Accede al primo elemento (il vector `V` NON deve essere empty)
`V.back()`	Accede all'ultimo elemento (il vector `V` NON deve essere empty)
`W = V`	Copia il contenuto da `V` a `W` (che deve esistere)
`V.push_back(val)`	Aggiunge `val` in coda (ultima posizione) incrementando automaticamente la dimensione
`V.pop_back()`	Elimina l'ultimo elemento decrementando automaticamente la dimensione
`V.resize(n)`	Ridimensiona a nuova size a `n`. Se `n` è più piccolo della size corrente, il vector viene ridotto ai suoi primi `n` elementi, rimuovendo quelli oltre. Se `n` è maggiore della dimensione corrente, il contenuto viene espanso inserendo alla fine quanti più elementi sono necessari per raggiungere una dimensione di `n`.
`V.clear()`	Svuota (porta la size 0)
`using namespace std;`	in testa al file per poter omettere il prefisso `std::` prima di ogni vector (come fatto per rendere il testo più conciso nel resto della sezione)
I/O su Files	vedi esempio di lettura e scrittura descritto dettagliatamente nell'Esercizio 6 di questa sezione

11.1 Esercizi di riscaldamento

1. Scrivere due funzioni `void readVector(vector<int> &v)` e `void printVector(const vector<int> &v)` che permettano rispettivamente di (1) inserire `N` valori letti da tastiera in un vector `v` (`N` positivo) e (2) stamparli:

```
void readVector(vector<int> &v) {
// Stampare "Inserisci la dimensione della sequenza: "
// Dichiarare una variabile intera N
// Leggere N
// iterare finché N non è positivo
    - Stampare "La dimensione deve essere positiva - riprova: "
    - Leggere N
/* iterare N volte
        - Leggere un valore intero
        - Memorizzare il valore letto in v
*/
}
```

```
void printVector(const vector<int> &v) {
/* iterare v.size() volte
        - stampare l'elemento corrente di v
*/
}
```

Scrivere un programma per testare le funzioni `readVector` e `printVector`:

```
// dichiarare un vector v di interi
// chiamare la funzione readVector su v
// chiamare la funzione printVector su v
```

[File `iovector.cpp, iovector.h, testf_iovector.cpp`]

2. Creare una variante di `readVector` chiamata `void readVectorAlt(vector<int> &v)`, da inserire sempre nel file `iovector.cpp`, che invece di richiedere il numero di elementi da leggere `N` seguito dagli elementi stessi, consenta di inserire direttamente i valori nel vector `v` sino a che l'utente non inserisce il carattere 'y', ad esempio:

```
12
5
10
8
y
```

Aggiungere al programma `testf_iovector.cpp` una chiamata a `readVectorAlt` per testarne il funzionamento ad esempio commentare la chiamata `readVector`).

[SUGGERIMENTO: Ogni numero inserito può essere salvato inizialmente in una `string s`. Per ogni inserimento si verifica che la stringa inserita non sia `"y"`. Se lo è la funzione termina altrimenti viene inserito nel vector il contenuto di `s` convertito in un `int` usando la funzione `int stoi (const string &str)` contenuta nella libreria `<string>`. Se la stringa `s` non rappresenta un intero viene sollevata un'eccezione (`std::invalid_argument`).]

3. Scrivere una funzione `void selectionSort_vector(vector<int> &v, bool ascending)` che effettui l'ordinamento di `v` secondo l'algoritmo *SelectionSort*. Se `ascending == true` allora l'ordinamento sarà crescente altrimenti decrescente.

```
void selectionSort_vector(vector<int> &v, bool ascending) {
// dichiarare una variabile int index
/* iterare sul vector dalla prima all'ultima posizione
   - memorizzare in index la posizione corrente (sia i)
   - *//* iterare sul vector dalla posizione successiva alla corrente (i+1) fino all'ultimo elemento
      -- se il valore alla pos corrente (j) e' (< se ascending==true, > altrimenti) del valore alla pos index
         --- memorizzare j in index
     *//*
   - scambiare il valore alla posizione i con quello alla pos index
*/
}
```

Scrivere un programma per testare la funzione `void selectionSort_vector(vector<int> &v, bool ascending)` secondo il seguente algoritmo:

```
// dichiarare un vector v di interi
// chiamare la funzione readVector su v
// chiamare la funzione printVector su v
// chiamare la funzione selectionSort_vector su v e true
// chiamare la funzione printVector su v  (test: deve essere stampato ordinato crescente)
// chiamare la funzione selectionSort_vector su v e false
// chiamare la funzione printVector su v  (test: deve essere stampato ordinato decrescente)
```

[File `selectionSort_vector.cpp, selectionSort_vector.h, testf_selectionSort_vector.cpp`]

4. Scrivere una funzione `int sequentialSearch_vector(const vector<int> &v, int item)` che effettui la ricerca dell'elemento item all'interno di `v`

```
int  sequentialSearch_vector(const vector<int> &v, int item) {
// dichiarare una variabile int loc e inizializzarla a -1
// dichiarare una variabile bool found e inizializzarla a false
/* iterare su v fino a che found diventa true o si è iterato su tutto il vector
   - se il valore alla pos corrente (i) e' uguale a item
      -- assegnare true a found e i a loc
*/
// restituire loc
}
```

Scrivere un programma per testare la funzione `int sequentialSearch_vector(const vector<int> &v, int item )`

[File `sequentialSearch_vector.cpp, sequentialSearch_vector.h, testf_sequentialSearch_vector.cpp`]

5. Implementare una una variante della funzione precedente secondo lo schema seguente (salvare in `sequentialSearch_vector.cpp`):

```
int  sequentialSearch_vector(const vector<int> &v, int item) {
/* iterare su v dalla prima all'ultima posizione
   - se il valore alla pos corrente (i) e' uguale a item
      -- restituire i
*/
// restituire -1
}
```

Quali sono le differenze fra le due implementazioni?

6. Scrivere un programma che effettua **una copia di un file**, cioè che **legge un contenuto da un file** e lo **scrive in un altro file**. In particolare, nel caso specifico, il file sorgente conterrà l'elenco di N città e per ognuna delle quali sarà riportata nella stessa riga la popolazione. Ad esempio:

```
5
Roma    2745819
Milano  1362551
Napoli   909422
Torino   836148
Palermo  627970
```

Notare che nella prima riga è riportato il numero N di città presenti nel file.

Sviluppare quindi due funzioni, la prima che legge le informazioni sulle città dal file e le inserisce in un vector. La seconda che leggere la informazioni dal vector e le scrive su un file (diverso).

Ogni elemento del vector sarà costituito da una city definita come riportato nella struct.

```
struct city{
   string name;
   int inhabitants;
};

typedef vector<city> cities;
```

Notare l'ultima riga. Lo scopo della typedef è quello di assegnare dei nomi alternativi (come `cities`) a dei tipi di dato esistenti (in questo caso `vector<city>`), solitamente a quelli la cui dichiarazione standard è troppo ingombrante.

Come leggere e scrivere dati su un file:

```
#include <fstream>

// Lettura di una stringa da un file
   ifstream infile;
   infile.open("fileDaLeggere.txt");
   string data;
   infile >> data;
   ....
   int num = stoi(data); // per trasformare il contenuto di una stringa (che rappresenta un int) in un int
   ....
   infile.close();

// Scrittura di una variabile var su un file
   ofstream outfile;
   outfile.open("fileDaScrivere.txt");
   outfile << var
   ...
   outfile.close();
```

Implementare quindi le due funzioni:

(a) **`void read_cities_from_file(cities &C, string fileName)`**: apre il file, legge il contenuto del file e lo inserisce nel vector C (nella prima riga trova l'informazione su quante città devono essere lette).

(b) **`void write_cities_to_file(const cities &C, string fileName)`**: scrive nel file il contenuto del vector C (nella prima riga scrive quante città conterrà il file).

Di seguito un esempio di main (da completare) che richiama le due funzioni per effettuare la copia. Verificare che il programma funzioni come richiesto (cioè che la copia dei file sia completa e corretta).

```cpp
#include <vector>
#include <string>
#include <fstream>

using namespace std;

struct city{
   string name;
   int inhabitants;
};

typedef vector<city> cities;

void write_cities_to_file(const cities &C, string fileName){
   // TO DO
}

void read_cities_to_file(cities &C, string fileName){
   // TO DO
}

int main(){
   cities C;
   read_cities_to_file(C,"citiesToRead.txt");
   write_cities_to_file(C,"citiesToWrite.txt");
}
```

[File `test_RW_Cities_on_file.cpp`]

11.2 Esercizi di base

7. Scrivere una funzione `vector<int> reverse(const vector<int> &v)` che prende in ingresso un `std::vector<int> v` e restituisce un nuovo `vector<int>` contenente il contenuto di `v` in ordine inverso.

   Scrivere un programma di test che legge alcuni interi in `vector<int> source` usando la funzione `readVectorAlt` implementata in precedenza (presente in `iovector.h`), e poi chiama `reverse`, assegnando il risultato a un altro `vector<int> dest`. Quindi stampa su una riga `source` e sulla riga seguente `dest` utilizzando la funzione `printVector` implementata in precedenza (presente in `iovector.h`).

   [File `reverse_vector.cpp, reverse_vector.h, testf_reverse_vector.cpp`]

8. Scrivere una funzione `vector<int> cat(const vector<int> &v1, const vector<int> &v2)` che riceve due argomenti `v1` e `v2`, di tipo `vector<int>`.

   La funzione crea un terzo `vector<int> v12` contenente il contenuto di `v1` seguito dal contenuto di `v2`.

   Valore restituito: `v12`.

   Scrivere un programma di test che legge alcuni interi usando (2 volte) la funzione `readVectorAlt` implementata in precedenza (presente in `iovector.h`), popolando quindi i `vector<int> first` e `second`.

   Quindi chiama `cat` per concatenare i due vettori e memorizza il risultato in `vector<int> total`, stampando infine i tre elenchi contenuti in `first`, `second` e `total` utilizzando la funzione `printVector` implementata in precedenza (presente in `iovector.h`).

   [File `cat.cpp, cat.h, testf_cat.cpp`]

9. Scrivere una funzione `void insert(vector<int> &v, unsigned long int i, int val)` che riceve tre argomenti: `v` di tipo `vector`, un intero `i`, un intero `val`.

   La funzione aggiunge a `v` in posizione `i` il valore di `val`.

   La lunghezza di `v` deve essere incrementata di 1 e tutto l'eventuale contenuto di `v`, dalla posizione `i` (compresa) fino alla fine, deve essere spostato in avanti di una posizione.

NOTA: siccome è previsto che il vettore si incrementi di dimensione, le posizioni di inserimento valide sono tutte quelle esistenti (da 0 a `v.size()-1`) e anche una oltre l'ultima (la posizione `v.size()`), quindi `i`∈ [`0`,`v.size()`]. Se `i` non è compreso in questo intervallo sollevare una eccezione `int`.

[SUGGERIMENTO: Inserire in coda a `v` il suo ultimo elemento (se esiste), spostare in avanti di una posizione gli elementi di v a partire dalla posizione i in avanti (copiandoli a partire dal fondo in modo da non sovrascriverli) e assegnare il valore da inserire nella posizione i.]

Valore restituito: `v` dopo la modifica.

Scrivere un programma di test che dichiara un `vector<int>` e fa il test della funzione `insert` nei seguenti casi:

(a) Inserimento in `v` vuoto

(b) Inserimento in testa (in posizione 0) a `v` non vuoto

(c) Inserimento in coda (dopo l'ultima posizione) a `v` non vuoto

(d) Inserimento in posizione generica (non testa, non coda) in `v` non vuoto

(e) Inserimento in posizione non valida (usare `try` ... `catch` per trattare l'eccezione).

[File `insert.cpp, insert.h, testf_insert.cpp`]

11.3 Esercizio Rubrica Telefonica con Vector di Struct e I/O su files

10. Implementare le funzioni che compongono una libreria per la memorizzazione e gestione di una rubrica telefonica utilizzando `struct` e `vector`. Oltre alla definizione delle funzioni e dei tipi richiesti per la libreria, dovete, naturalmente, scrivere un programma per testare le funzioni man mano che le implementate.

    [File `phoneBook.cpp, phoneBook.h, phoneBook_main_test.cpp`]

    File .h (struct, typedef e funzioni): per prima cosa definiamo il file `phoneBook.h`:

```
#ifndef phoneBook_H
#define phoneBook_H

#include <vector>
#include <string>

using namespace std;

struct contact{
   string surname;
   string name;
   string phoneNumber;
};

typedef vector<contact> phoneBook;

void push_contact_in_phoneBook(phoneBook &B, string surname, string name, string number);
void print_phoneBook(const phoneBook &B);
void read_phoneBook_from_file(phoneBook &B, string fileName);
void write_phoneBook_to_file(const phoneBook &B, string fileName);
void sort_phoneBook_by_surnames(phoneBook &B);
bool find_phoneBook_contact_by_surname(const phoneBook &B, string s, unsigned long int &pos);
void shift_phoneBook(phoneBook &B, unsigned long int pos);
void add_contact_to_phoneBook(phoneBook &B, string surname, string name, string number);

#endif
```

    All'inizio del file troviamo una struct `contact` contenente i campi `surname`, `name`, `phoneNumber` (tutti di tipo stringa dato che i numeri di telefono che iniziano per 0 o che contengono caratteri speciali come '+' devono essere salvati correttamente).

    Poi creiamo sempre nel file `phoneBook.h` il tipo *vettore di contatti* dandogli il nome `phoneBook`, usando `typedef vector<contact> phoneBook;` (vedi anche l'Esercizio 6 di questa sezione). In questo modo in qualsiasi parte del codice sia necessario creare una nuova phoneBook di nome `namePhoneBook` (ad esempio nel file dei test `phoneBook_main_test.cpp`) sarà sufficiente scrivere:

```
phoneBook namePhoneBook;
```

Analogamente le intestazioni delle funzioni saranno semplificate come vediamo nei prototipi riportati nel file `phoneBook.h`.

Testing: per scrivere i programmi intermedi per testare le funzioni prodotte fino ad un certo punto, potete modificare sempre lo stesso main, commentando i pezzi che non vi servono più perché avete testato in maniera soddisfacente la funzione a cui fanno riferimento; però non cancellate le sequenze di test, perché se poi dovete fare modifiche alle funzioni precedenti o vi accorgete di un caso che non avete provato, troverete ancora il codice di test pronto, basta togliere i commenti.

Lettura da file: per quanto riguarda l'input dei dati su cui provare le chiamate di funzione, potete inizialmente inserire gli elementi con la prima funzione che vi chiediamo di implementare. Più avanti poi implementerete funzioni di lettura e scrittura su file (come visto nell'Esercizio 6 di questa sezione) che vi permettono di modificare l'input da fornire nei test più rapidamente.

Funzioni da implementare: a questo punto possiamo implementare le varie funzioni richieste nel seguito, vanno salvate nel file `phoneBook.cpp`.

(a) `void push_contact_in_phoneBook(phoneBook &B, string surname, string name, string number)`: scrivere una funzione per aggiungere un contatto `C` (di cui ci vengono forniti i tre valori dei campi) in coda alla rubrica `B`.

(b) `void print_phoneBook(const phoneBook &B)`: scrivere una funzione per stampare a schermo l'intero contenuto della rubrica `B`. Una riga per ogni contatto, ad esempio:

```
Einstein Albert 012334454
Tesla Nikola 088388831
Gauss Carl-Friedrich 03966776111
Fibonacci Leonardo 11235813
```

(c) `void read_phoneBook_from_file(phoneBook &B, string fileName)`: scrivere una funzione che legge una `phoneBook` memorizzata sul file `fileName`. Il file dove è salvata la rubrica presenta una prima stringa contenente il numero (intero positivo) di contatti da leggere, seguito da triple costituite da tre stringhe (cognome, nome, telefono). Come base di partenza vedere Esercizio 6 di questa sezione. Per semplicità non consideriamo nomi e cognomi multipli. Esempio di file:

```
4
Einstein Albert 012334454
Tesla Nikola 088388831
Gauss Carl-Friedrich 03966776111
Fibonacci Leonardo 11235813
```

Assumiamo che i file siano sempre forniti nel formato corretto.

(d) `void write_phoneBook_to_file(const phoneBook &B, string fileName)`: scrivere una funzione che salva l'intera `phoneBook` sul file `fileName`. Il formato del file dove essere analogo a quello descritto sopra per la lettura. Come base di partenza vedere Esercizio 6 di questa sezione.

(e) `void sort_phoneBook_by_surnames(phoneBook &B)`: scrivere una funzione che data una rubrica ordini alfabeticamente gli elementi in essa contenuti rispetto al campo `Surname`.

(f) `bool find_phoneBook_contact_by_surname(const phoneBook &B, string s, unsigned long int &pos)`: scrivere una funzione che, **utilizzando la ricerca binaria**, abbia il seguente comportamento:

- se nella rubrica **esiste** un contatto `C` il cui campo `C.surname` è uguale all'argomento `S`, allora restituisca `true` e in `pos` viene salvato l'indice di tale contatto nella rubrica (ossia nel vettore);
- se nella rubrica **esistono** più contatti `C` il cui campo `C.surname` è uguale all'argomento `S`, allora restituisca `true` e in `pos` viene salvato l'indice del primo contatto nella rubrica (ossia nel vettore) incontrato durante l'esecuzione della ricerca binaria (che potrebbe non essere il primo incontrato leggendo il vector dalla posizione 0 in avanti);
- se nella rubrica **non esiste** un contatto `C` il cui campo `C.surname` è uguale all'argomento `S`, allora restituisca `false` e in `pos` l'indice del contatto che sarebbe quello immediatamente successivo in ordine alfabetico. Se tutti gli elementi della rubrica hanno il valore `surname` inferiore a `S` o se la rubrica è vuota, allora `pos` avrà per valore `B.size()`.

[SUGGERIMENTO: Ricordate che la ricerca binaria assume che il vettore sia ordinato]

(g) `void shift_phoneBook(phoneBook &B, unsigned long int pos)`: scrivere una funzione che incrementa di un elemento la dimensione del vettore `B` e poi sposta a destra di un elemento tutti gli elementi a partire dalla posizione `pos`. Questa funzione serve a creare uno spazio per poi inserire un elemento nel punto indicato (vedi funzione successiva).

- Se `pos > B.size()` solleva un eccezione di tipo string con il seguente messaggio: `"void shift_phoneBook: posizione fuori dai limiti ammissibili"`;
- se `pos == B.size()` non fa nulla (perchè non ci sono elementi in posizione `pos` in quanto è già finito il vector);
- se B è vuoto (cioè `B.size() == 0`) non fa nulla (non è possibile shiftare una rubrica vuota).

(h) `void add_contact_to_phoneBook(phoneBook &B, string surname, string name, string number)`: scrivere una funzione che inserisce il nuovo contatto nella rubrica nella posizione giusta rispetto all'ordine alfabetico. Come precondizione alla chiamata della funzione, assumiamo quindi che la rubrica `B` sia ordinata (se non lo è chiamare `sort_phoneBook_by_surnames` prima di chiamare la funzione).
[SUGGERIMENTO: Usare la funzione `find_phoneBook_contact_by_surname` per ottenere la posizione in cui deve essere inserito il nuovo contatto. Per evitare di sovrascrivere un contatto esistente può essere necessario fare spazio al nuovo contatto da inserire spostando gli elementi della rubrica usando la funzione `shift_phoneBook` (chiaramente salvo il caso in cui il contatto vada inserito in fondo alla rubrica, o la rubrica sia vuota).]

11.4 Esercizi più avanzati

11. Ancora sul **Crivello di Eratostene**: scrivere la funzione `vector<int> primes(int n)` che dato un intero positivo `n` restituisce un `vector<int>` che contiene tutti e soli i numeri primi compresi tra `2` e `n`, che quindi deve essere un vettore vuoto se `n<2`. Partire dal codice della funzione `isprime` (Parte 4, esercizio 12).

    [File `primes.cpp, primes.h, testf_primes.cpp`]

12. **GIOCO DELL'UNO** Scrivere un programma per realizzare il gioco di carte UNO. Potete trovare le istruzioni del gioco al seguente indirizzo `https://it.wikipedia.org/wiki/UNO_(gioco_di_carte)`.
[SUGGERIMENTO: Dovrete definire almeno una struct che contiene le informazioni di una carta, e i vector per memorizzare le carte in mano di ogni giocatore, nel mazzo e in tavola.]

    [File `uno.cpp, uno.h, play_uno.cpp`] – L'ultimo file contiene il programma di gioco, mentre il primo contiene le funzioni e i tipi necessari.

Parte 12

Liste e ripasso/approfondimento su Librerie

In questa sezione vedremo vari esercizi sulle Liste come spiegate a lezione e organizzeremo i file in librerie.

Ripasso/approfondimento su Librerie

Negli esercizi di questa sezione sarà richiesto di creare *librerie*. Si tratta quindi di esercizi sulla modularità: le funzioni che forniscono gli strumenti per risolvere un problema specifico vengono raccolte in un unico *modulo* = un file sorgente.

Chi usa tali funzioni deve naturalmente avere accesso alle definizioni (tipi, prototipi...). Queste sono però specificate in un file header, l'unico file che l'utente ha bisogno di conoscere. Le funzioni sono così *incapsulate* in un componente che può anche essere compilato separatamente: in tal caso non serve nemmeno più avere il file sorgente.

La modularità (file separato) rende più facile riusare il codice. L'incapsulamento (non conoscere il sorgente del modulo) rende possibile modificare l'implementazione senza dover modificare i programmi che le usano. Questo è utile in caso di errori, aggiornamenti tecnologici, miglioramenti e arricchimenti... Inoltre incapsulare le informazioni serve a nascondere i dettagli implementativi che non interessano chi usa le funzionalità fornite (*information hiding*).

Cheatsheet

Definire una libreria – si separa la sua *interfaccia*, ovvero tipi e prototipi di funzioni, dall'*implementazione*.
Promemoria: chi usa la libreria deve vedere solo l'interfaccia, chi la programma vede anche l'implementazione.

- Interfaccia ⟶ file *header* con estensione `.h`

  Cosa si può inserire in un file header? Solo **dichiarazioni** che è ammesso ripetere in più file, ovvero:

  - prototipi di funzioni
  - dichiarazioni di tipo come struct o enum
  - typedef

  NON si possono inserire né **corpi di funzioni** né **dichiarazioni di variabili/costanti**.
- Implementazione ⟶ file `.cpp`

  Cosa si può inserire in un file sorgente (.cpp)? **Definizioni**, cioè:

  - corpi di funzioni (ovviamente incluso `int main()`)
  - variabili/costanti globali

  Possono esserci anche **dichiarazioni** locali al file, cioè che vengono usate nell'implementazione ma non servono all'utilizzatore.

Usare una libreria

- Il file header va incluso in tutti i file che hanno bisogno di usare la libreria:

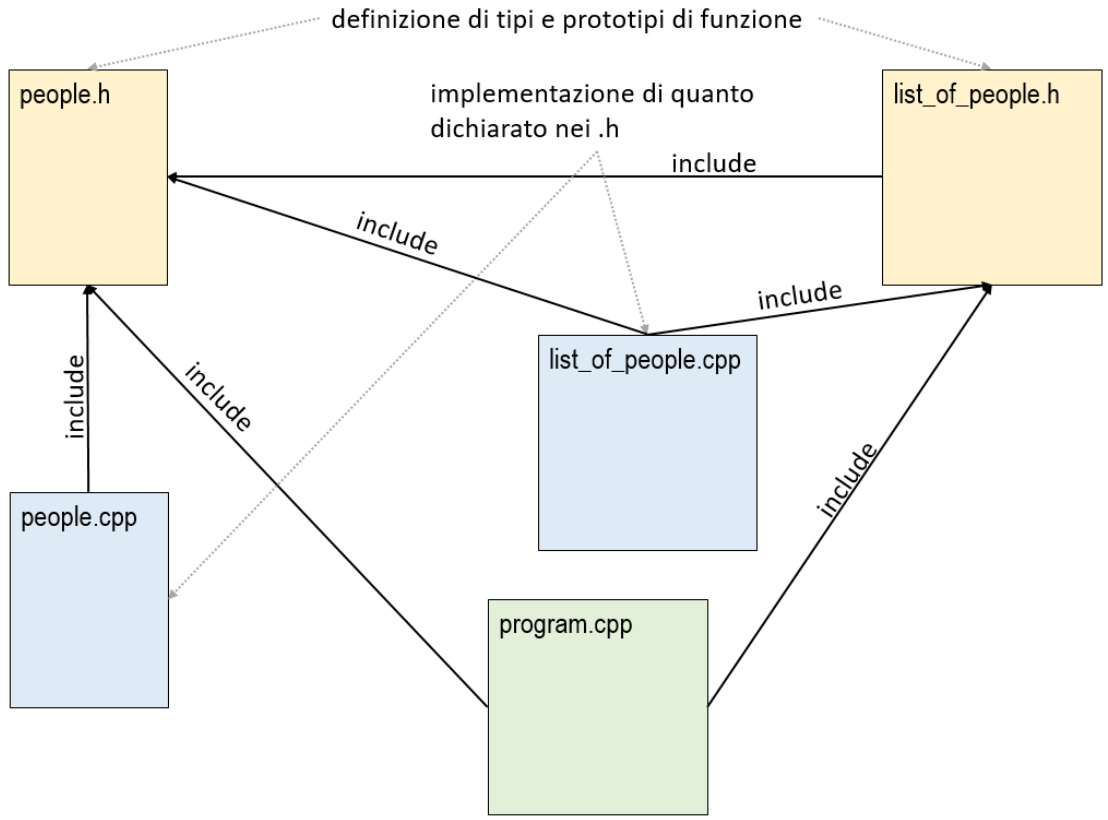
  - **sempre** nel file (`.cpp`) che contiene il programma che usa la libreria
  - spesso anche nel file che contiene l'implementazione (come buona pratica conviene inserirlo sempre)
  - eventualmente nei file di header di altre librerie basate su questa
    Esempio: la mia libreria usa il tipo `std::string` → il mio header include `<string>`.

  Promemoria: Dove uso una libreria, includo il relativo header; dove non ne uso neanche una, non devo includerlo. Non serve includere tutto ovunque.
- Nel comando di compilazione bisogna elencare tutti i file che fanno parte del vostro progetto: il file .cpp che contiene il main e tutti i file che contengono l'*implementazione* delle librerie che usate (.cpp se avete i sorgenti, .o se sono già compilati).

NON bisogna compilare i file header perché vengono automaticamente copiati all'interno dei .cpp.

Le librerie di sistema sono sottintese e linkate automaticamente; eventuali librerie non standard vanno invece specificate.

Ecco un esempio tipico di uso di librerie (un programma per gestire un'anagrafe, ad esempio, oppure il registro degli esami. . . la libreria *liste di persone* potrebbe essere riusata per molti scopi, corrispondenti a diversi `program.cpp`):



Notate che nell'esempio (come in molti casi concreti), i file `program.cpp` e `list_of_people.cpp` includono due volte il file `people.h` una volta direttamente e una indirettamente attraverso l'inclusione di `list_of_people.h`. Questo è un'inutile perdita di tempo, perché vengono aggiunte due volte le stesse cose. Inoltre, in casi complicati, è possibile che si creino *cicli* di inclusione che causano errore in compilazione. Per evitare questo problema, bisogna dare ai propri file .h questa struttura:

```
#ifndef NOME_VOSTRO_FILE
#define NOME_VOSTRO_FILE
/*tutto il codice del .h */
#endif
```

In questo modo la prima volta che viene processata l'inclusione del file la costante `NOME_VOSTRO_FILE` non è definita e il codice viene copiato tutto. Se si include una seconda volta lo stesso file, `NOME_VOSTRO_FILE` è già stata definita (la prima volta che lo si è incluso) e quindi si salta il blocco e non si copia nulla.
Inoltre, nello header:

- **NON** inserire #include, se non sono necessari a **tutti** i file che useranno l'include
- **NON** inserire using namespace (se necessario usare qualificatore std::, per esempio per string e vector)

NOTA. La direttiva `#include` va usata solo per includere file header `.h`, non per incorporare altri file sorgenti `.cpp`!! Tecnicamente funziona, ma è vietato dalle buone pratiche di programmazione.

12.1 Esercizio guidato su Liste

1. Realizzate la libreria delle liste semplici di stringhe, con le funzioni viste a lezione (ed alcune extra indicate)

   (a) Creare un file `string_list.h` (come indicato nel seguito) contenente le dichiarazioni del tipo `cell` e delle funzioni che manipolano liste.

```
#ifndef STRINGLIST
#define STRINGLIST
#include<string>

struct cell{ // elemento della lista
   std::string data;
   cell* next;
};
typedef cell* list;

void headInsert(list &l, std::string s);
// inserisce un elemento contenente s in testa alla lista l

void tailInsert(list &l, std::string s);
// inserisce un elemento contenente s in coda alla lista l

void read(list &l);
// legge valori da input e li memorizza nella lista l
// l'inserimento termina se l'utente inserisce "STOP!"

void print(const list &l);
// stampa tutti gli elementi della lista l
// in linea separati da "->"

unsigned int size(const list &l);
// ritorna il numero di elementi della lista l

bool isEmpty(const list &l);
// ritorna true se la lista l e vuota, false altrimenti

std::string getElem(const list &l, unsigned int i);
// ritorna il contenuto dell'i-esimo elemento della lista l
// - se la lista l è vuota solleva un'eccezione con messaggio di tipo string
// - se l'indice i è invalido solleva un'eccezione con messaggio di tipo string

void insertAt(list &l, std::string s, unsigned int i);
// inserisce un elemento contenente s in posizione i nella lista l
// - se la lista l è vuota solleva un'eccezione con messaggio di tipo string
// - se l'indice i è invalido solleva un'eccezione con messaggio di tipo string

void deleteAt(list &l, unsigned int i);
// cancella l'elemento in posizione i dalla lista l
// - se la lista l è vuota solleva un'eccezione con messaggio di tipo string
// - se l'indice i è invalido solleva un'eccezione con messaggio di tipo string

void deleteOnce(list &l, std::string s);
// cancella dalla lista l il primo elemento contenente s
// (partendo dall'inizio)

void deleteAll(list &l, std::string s);
// cancella dalla lista l tutti gli elementi contenenti s
#endif
```

   (b) Creare un file `string_list.cpp` dove implementare quanto avete dichiarato in `string_list.h`.
   Non dimenticatevi di fare `#include` di `string_list.h`.

(c) Creare un file `string_list_test.cpp` e implementare il main di test (anche in questo caso non dimenticatevi di fare `#include` di `string_list.h`) in cui andrete a inserire il codice necessario a verificare la corretta implementazione delle funzioni man mano che le implementate.

NOTA. Ricordatevi di testare le funzioni una alla volta man mano che le implementate. Commentate (ma non cancellate) il codice di test che non vi serve più, quando siete sicuri che una funzione sia corretta e volete testare la successiva; in questo modo se avete bisogno di testare nuovamente una funzione che consideravate già completata vi basterà scommentare/commentare pezzi del main e farete prima.

(d) Varianti

i. Modificare la funzione `deleteOnce` in modo che restituisca un booleano, vero se l'elemento da cancellare è stato trovato e rimosso, falso altrimenti

ii. Modificare la funzione `deleteAll` in modo che restituisca un intero pari al numero di occorrenze trovate e cancellate (0 se la stringa non è presente nella lista, quindi)

12.2 Esercizio su Liste Ordinate

2. Realizzate la libreria delle liste semplici *ordinate* di stringhe (in ordine crescente), con le funzioni viste a lezione ed alcune extra indicate. Se l'avete implementata, potete in parte riusare la libreria delle stringhe realizzata nell'esercizio precedente modificando le implementazioni dove necessario.

(a) Creare un file `string_ord_list.h` e inseritevi la dichiarazione del tipo `ordList` e delle funzioni che manipolano liste.

L'intuizione di questo esercizio è che le liste ordinate sono semplicemente liste che sappiamo, per come le abbiamo costruite, essere ordinate e su cui forniamo solo funzioni che non possono "disordinarle".

Nel codice seguente trovate i prototipi delle funzioni da implementare:

```
#ifndef STRINGORDLIST
#define STRINGORDLIST

#include<string>

struct cell{ // elemento della lista
   std::string data;
   cell* next;
};

typedef cell* ordList;

void insertElement(ordList &l, std::string s);
// inserisce un elemento contenente s nella lista ordinata l mantenendo la lista ordinata

void readList(ordList &l);
// legge valori da input e li memorizza (aggiunge) nella lista ordinata l
// al termine dell'esecuzione di readList la lista l deve essere una lista ordinata.
// l'inserimento termina se l'utente inserisce "STOP!"
// NB: differente rispetto alla versione per liste non ordinate

void printList(const ordList &l);
// stampa tutti i valori (data) degli elementi della lista ordinata l in linea separati da "->"
// NB: nessuna variazione rispetto alla versione per liste non ordinate

unsigned int listSize(const ordList &l);
// ritorna il numero di elementi della lista ordinata l
// NB: nessuna variazione rispetto alla versione per liste non ordinate

bool isEmptyList(const ordList &l);
// ritorna true se la lista ordinata l e vuota, false altrimenti
// NB: nessuna variazione rispetto alla versione per liste non ordinate

std::string getElement(const ordList &l, unsigned int i);
```

```
// ritorna il contenuto dell'i-esimo elemento della lista ordinata l
// - se la lista ordinata l è vuota solleva un'eccezione con messaggio di tipo string
// - se l'indice i è invalido solleva un'eccezione con messaggio di tipo string
// NB: nessuna variazione rispetto alla versione per liste non ordinate

void deleteElementAt(ordList &l, unsigned int i);
// cancella l'elemento in posizione i dalla lista ordinata l
// - se la lista ordinata l è vuota solleva un'eccezione con messaggio di tipo string
// - se l'indice i è invalido solleva un'eccezione con messaggio di tipo string
// NB: nessuna variazione rispetto alla versione per liste non ordinate

void deleteElementOnce(ordList &l, std::string s);
// cancella dalla lista ordinata l il primo elemento contenente s
// NB: nessuna variazione rispetto alla versione per liste non ordinate

void deleteAllElements(ordList &l, std::string s);
// cancella dalla lista ordinata l tutti gli elementi contenenti s
// NB: deve essere ottimizzata rispetto alla versione per liste non ordinate

bool listAreEqual(const ordList &l1, const ordList &l2);
// restituisce true se le due liste contengono le stesse stringhe (con la stessa molteplicità)
// false altrimenti

ordList concatLists(const ordList &l1, const ordList &l2);
// restituisce una nuova lista ordinata contenente gli elementi delle 2 liste ordinate l1 e l2
// Il numero di occorrenze di un elemento nel risultato
// è la somma del numero delle sue occorrenze nelle due liste l1 e l2
// NB: ogni elemento della nuova lista deve essere un duplicato di quelli contenuti in l1 e l2
//     in questo modo sia l1 che l2 non subiscono alcuna modifica

ordList unionLists(const ordList &l1, const ordList &l2);
// restituisce una nuova lista ordinata senza ripetizioni contenente
// gli elementi presenti in almeno una delle due liste l1 e l2
// NB1: ogni elemento della nuova lista deve essere un duplicato di quelli contenuti in l1 e l2
//      in questo modo sia l1 che l2 non subiscono alcuna modifica
// NB2: opzionalmente cercate di implementare la funzione in modo da sfruttare il fatto che
//      le due liste sono ordinate per minimizzare il numero di operazioni necessarie

ordList intersectLists(const ordList &l1, const ordList &l2);
// restituisce una nuova lista ordinata senza ripetizioni contenente
// gli elementi presenti in entrambe le liste l1 e l2
// NB1: ogni elemento della nuova lista deve essere un duplicato di quelli contenuti in l1 e l2
//      in questo modo sia l1 che l2 non subiscono alcuna modifica
// NB2: opzionalmente cercate di implementare la funzione in modo da sfruttare il fatto che
//      le due liste sono ordinate per minimizzare il numero di operazioni necessarie

#endif
```

(b) Creare un file `string_ord_list.cpp` e inserirvi le implementazioni di quanto avete dichiarato in `string_ord_list.h`.

(c) Creare un file `string_ord_list_test.cpp` e inserirvi il main di test (anche in questo caso non dimenticatevi di fare `#include` di `string_ord_list.h`) in cui andrete a inserire il codice necessario a verificare la corretta implementazione delle funzioni man mano che le implementate. Ricordatevi sempre di inizializzare le liste che create nel main con `nullptr` (o eventualmente con un valore ritornato da una funzione) prima di usarle come parametri nelle funzioni.

12.3 Esercizi più avanzati

3. Re-implementate tutte le funzioni richieste nell'Esercizio 1 per un tipo `double_linked_list` in cui ciascuna cella della lista ha, oltre ai campi `data` e `next`, anche il campo `prev`, che *punta* all'elemento precedente (ed è quindi nullo sulla testa della lista, analogamente a come `next` è nullo sull'ultimo elemento). Aggiungere poi una funzione `void printReverse(const list &l)` che stampa la lista dall'ultimo elemento al primo seguendo i campi `prev`.

Parte 13

Ricorsione

In questa parte dell'eserciziario ci concentriamo sulla definizione ricorsiva di funzioni. Come sempre, dopo aver scritto la funzione, occorrerà verificarla con un programma di test. Gli errori andranno gestiti mediante la definizione di opportune eccezioni.

13.1 Esercizi di riscaldamento

1. Scrivere una funzione ricorsiva `unsigned fact(unsigned n)` per il calcolo del fattoriale di un intero positivo `n`:

```
unsigned fact(unsigned n) {
// se n==0 ritorna 1
// altrimenti ritorna n moltiplicato per il fattoriale di n-1
}
```

2. Scrivere una funzione ricorsiva `unsigned coeffBin(unsigned n, unsigned k)` per il calcolo del coefficiente binomiale: dati $n, k \in \mathbb{N}$ tali che $0 \leq k \leq n$ il coefficiente binomiale è

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

```
unsigned coeffBin(unsigned n, unsigned k) {
// se  k>n solleva eccezione
// se n==k oppure k==0 ritorna 1
// altrimenti ritorna la somma del coefficiente binomiale di n-1 e k-1 e del coefficiente binomiale di n-1 e k
}
```

3. Scrivere una funzione ricorsiva `unsigned fibonacci(unsigned n)` per il calcolo del numero di Fibonacci, secondo la seguente definizione:

```
fibonacci(0) = 1
fibonacci(1) = 1
fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
```

```
unsigned fibonacci(unsigned n) {
// se n==0 oppure n==1 ritorna 1
// altrimenti ritorna la somma del numero di fibonacci di n-1 e del numero di fibonacci di n-2
}
```

4. Scrivere una funzione ricorsiva `unsigned MCD(unsigned m, unsigned n)` che dati due numeri interi `m` ed `n` calcola il massimo comune divisore utilizzando l'algoritmo di *Euclide*, descritto nel seguente modo:

```
MCD(m,n) = m se m==n
MCD(m,n) = MCD(m-n, n) se m > n
MCD(m,n) = MCD(m, n-m) se n > m
```

13.2 Esercizio: Funzioni Ricorsive su Liste

In questa sezione consideriamo funzioni che operano su liste ***semplici***, ***ordinate*** e ***senza ripetizioni*** di un generico tipo `T`. La scelta dello specifico tipo è a vostra discrezione.

```
struct cell{
   T data;
   cell* next;
};

typedef cell* list;
```

5. Per poter popolare e visualizzare una lista sulla quale testare le funzioni ricorsive da implementare nei prossimi punti di questo esercizio per prima cosa riprendiamo due funzioni di base dall'esercizio 2 della Parte 12, implementate in modo standard (cioè non ricorsivo):

   - `void insertElement(list &l, T s)`
   - `void printList(const list &l)`

   – Rinominare la `insertElement` in `insertElemInOrderedList` e adattarla al fatto che questa volta si trattano liste senza ripetizione degli elementi (cioè se un elemento è già presente non lo inserisce).
   – Modificare la `printList` in modo che stampi semplicemente gli elementi separati da spazio (es. `1 5 6 10 21`).
   – Creare una lista e stamparla e verificare che tutto funzioni.

6. Scrivere una funzione ricorsiva `void recursive_printList(const list &l)` che stampa il contenuto della lista su una singola riga con elementi separati da spazio `" "`: (es. `1 5 6 10 21`). Confrontare la stampa di questa funzione con quella standard al punto precedente.

7. Scrivere una funzione ricorsiva `void recursive_reversePrintList(const list &l)` che stampa il contenuto della lista su una singola riga con elementi separati da spazio `" "`: (es. `21 10 6 5 1`). La versione non ricorsiva sarebbe stata un po' più complessa da implementare, perchè?

8. Scrivere una funzione ricorsiva `void recursive_deleteAllElementsFromList(list &l)` che elimina tutti gli elementi dalla lista e alla fine assegna `l = nullptr`. Verificare con Valgrind che non ci siano memory leaks.

9. Scrivere una funzione ricorsiva `bool recursive_isElementInOrderedList(const list &l, T x)` per implementare la ricerca di un elemento di tipo `T` nella list `l`. Ottimizzare la funzione grazie al fatto che sappiamo che la lista è ordinata.

10. Scrivere una funzione ricorsiva `unsigned recursive_ListLength(const list &l)` che calcola la lunghezza di una lista `l`.

11. Scrivere una funzione ricorsiva `bool recursive_insertElemInOrderedList(list &l, T x)` che dati una lista `l` di elementi di tipo `T` inserisce il nuovo elemento contenente il valore `x` nella posizione giusta. La funzione restituisce `true` se l'operazione è andata a buon fine, `false` altrimenti. Verificare con Valgrind che non ci siano memory leaks.

12. Scrivere una funzione ricorsiva `bool recursive_removeElemFromOrderedList(list &l, T x)` che dati una lista `l` di elementi di tipo `T` elimina l'elemento `x` dalla lista (che è sempre unico). La funzione restituisce `true` se l'operazione è andata a buon fine, `false` altrimenti. Verificare con Valgrind che non ci siano memory leaks.

13.3 Esercizio: Funzioni Ricorsive (e non) su Insiemi

13. In questa sezione lavoreremo su una possibile implementazione del tipo di dato ***insieme*** implementando alcune funzioni fondamentali per la manipolazione di questo tipo di dato in modo ricorsivo.

    In particolare l'*insieme* sarà implementato usando una *lista* ***doppiamente linkata***, ***circolare***, ***ordinata*** e ***senza ripetizioni*** in cui ciascuna cella della lista ha, oltre ai campi `data` e `next`, anche il campo `prev`, che *punta* alla cella precedente.

    Inoltre essendo la lista circolare:

    - il campo `next` dell'ultima cella anziché puntare a `nullptr` punterà alla prima cella della lista;
    - il campo `prev` della prima cella anziché puntare a `nullptr` punterà all'ultima cella della lista;

    (a) Creare un file `set_dll.h` e inseritevi la dichiarazione del tipo `set_dll` e delle funzioni che manipolano il `set`. Nel codice seguente trovate i prototipi delle funzioni da implementare:

```
#ifndef SETDLL
#define SETDLL

struct set_cell{
   int data;
   set_cell* next;
   set_cell* prev;
};

typedef set_cell* set_dll; //dll => double linked list
// NB:  set_dll DEVE essere implementato tramite una lista con le seguenti caratteristiche:
//  - doppiamente linkata
//  - circolare
//  - ordinata
//  - senza ripetizioni

void insert_set_dll(set_dll &s, int v);
// inserisce un nuovo elemento (con data = v) nel set s

void print_set_dll(const set_dll &s);
// stampa il contenuto set s dal primo all'ultimo elemento
// (in linea con elementi separati da "->")

void revprint_set_dll(const set_dll &s);
// stampa il contenuto set s dall'ultimo al primo elemento
// (in linea e con elementi separati da "->")

set_dll union_set_dll(const set_dll &s1, const set_dll &s2);
// ritorna un set che rappresenta  l'unione dei set s1 e s2
// NB: ogni elemento del nuovo set deve essere un duplicato di quelli contenuti in s1 e s2
//     in questo modo sia s1 che s2 non subiscono alcuna modifica

set_dll intersect_set_dll(const set_dll &s1, const set_dll &s2);
// ritorna un set che rappresenta  l'intersezione dei set s1 e s2
// NB: ogni elemento del nuovo set deve essere un duplicato di quelli contenuti in s1 e s2
//     in questo modo sia s1 che s2 non subiscono alcuna modifica

#endif
```

    (b) Creare un file `set_dll.cpp` e inserirvi le implementazioni di quanto avete dichiarato in `set_dll.h` usando funzioni comuni NON ricorsive.

    (c) Come esercizio avanzato, creare un file `set_dll_recursive.cpp` e inserirvi le implementazioni di quanto avete dichiarato in `set_dll.h` usando funzioni ricorsive.

    (d) Creare un file `set_dll_test.cpp` e inserirvi il main di test in cui andrete a inserire il codice necessario a verificare la corretta implementazione delle funzioni man mano che le implementate. Ricordatevi sempre di inizializzare i set che

create nel main con `nullptr` (o eventualmente con un valore ritornato da una funzione) prima di usarli come parametri nelle funzioni.

13.4 Esercizi più avanzati

14. Scrivere una funzione ricorsiva che, data una lista, ne restituisce una con gli elementi in ordine inverso. Per la progettazione della funzione, considerare due varianti, ossia il caso in cui si abbia una diversa lista da restituire in output, e quello in cui modifica direttamente la lista di input.

15. Scrivere una funzione ricorsiva per la ricerca binaria su sequenze ordinate. Come per il punto precedente, implementare la versione basata su vector e su liste.

16. Scrivere una funzione ricorsiva che realizza l'algoritmo di *Merge Sort*. Implementare due varianti della funzione, utilizzando prima vector e poi liste.

Appendice A

Cheatsheet per lavorare su Linux

Convenzioni grafiche:

`comando`	nome comando
`comando` ⟨`argomento`⟩	argomento obbligatorio
`comando [opzione]`	argomento opzionale
`comando {opz1|opz2}`	argomenti in alternativa

A.1 Comandi bash

Cheatsheet

Abbreviazioni nomi file e cartelle (Si usano come argomenti dei comandi, non sono comandi!

`?`	qualunque stringa di lunghezza 1 Es: `file1 file2 file3` si abbrevia con ⟶ `file?`
`*`	qualunque stringa di qualunque lunghezza Es: `main.cpp aux.cpp library.cpp` si abbrevia con ⟶ `*.cpp`
`.`	cartella corrente
`..`	cartella che contiene quella corrente

Comandi

`ls`	visualizza contenuto della directory corrente
`cd` ⟨`cartella`⟩	cambia directory (cartella) di lavoro
`cd ..`	per spostarsi nella cartella superiore
`pwd`	print working directory, mostra cartella corrente
`rm [-i]` ⟨`file`⟩ `[file2 file3 ...]`	cancella file (con `-i`: chiedi conferma)
`mkdir` ⟨`cartella`⟩	crea cartella
`rmdir` ⟨`cartella`⟩	cancella cartella se vuota
`rm -r` ⟨`cartella`⟩	cancella cartella e il suo contenuto
`rm -r /*`	cancella tutto il filesystem (e non c'è undelete!)
`edit` ⟨`file`⟩	chiama text editor predefinito e apre ⟨`file`⟩
`zip` ⟨`archiveName`⟩ ⟨`files`⟩	crea uno archivio zip contenente i file specificati
`zip CognomeN.zip ese1.cpp ese2.cpp`	(esempio)
`unzip` ⟨`archiveName`⟩	decomprime il contenuto dell'archivio nella cartella corrente

N.B. Un file di testo non deve necessariamente chiamarsi `qualcosa.txt`. Esempi:

- i file sorgenti (estensione `.cpp`)
- i file header (estensione `.h`)
- i file di configurazione di programmi, es. per bash il nome completo è `.bashrc`

Cheatsheet

Controllo I/O ed esecuzione

`prog <⟨file⟩`	`prog` legge standard input (`cin`) da ⟨`file`⟩ anziché da tastiera – *redirection*
`prog >⟨file⟩`	`prog` scrive standard output (`cout`) su ⟨`file`⟩ anziché su schermo
`prog 2>⟨file⟩`	`prog` scrive standard error (`cerr`) su ⟨`file`⟩ anziché su schermo
	Nota: in scrittura ⟨`file`⟩ viene creato o sovrascritto senza chiedere conferma
`prog1 | prog2`	scrive standard output di `prog1` su standard input di `prog2` – *pipeline*
`prog &`	avvia `prog` in background e torna a bash
`prog1; prog2`	avvia `prog1` e al termine avvia `prog2`
Tasto `control-C`	arresta programma in esecuzione
Tasto `control-Z`	mette in pausa programma in esecuzione
`fg`	il programma in pausa riprende esecuzione in foreground
`bg`	il programma in pausa riprende esecuzione in background

A.2 Editing dei file sorgente

Cheatsheet

Qualunque text editor va bene

Visualizzare numeri di linea:

gedit, pluma	menu *Preferences* → linguetta *View* → *Display line numbers*
jedit	menu *Global Options* → linguetta *Gutter option* → *Line Numbering*
vim	`(esc):set numbers` o `:se nu`
emacs (in finestra grafica)	menu *Options* → sottomenu *Show/Hide* → *Line Numbers*
emacs (in terminale)	(esc)x linum-mode

A.3 Compilazione

Cheatsheet

Usi comuni del comando `g++`

`g++` ⟨`filesorgente.cpp`⟩	compila ed esegue linking, crea eseguibile `a.out`
`g++` ⟨`filesorg1.cpp`⟩ ⟨`filesorg2.cpp`⟩	compila tutti ed esegue linking insieme, crea eseguibile `a.out`
`g++` ⟨`filesorgente.o`⟩	esegue linking di file già compilato
`g++` ⟨`fileheader.h`⟩	**NO!**
`g++ ...argomenti, opzioni... 2> file`	scrive errori e warning su ⟨`file`⟩
`g++` ⟨`...argomenti...`⟩ `2> file; head file`	scrive errori e warning su ⟨`file`⟩ e subito dopo mostra le prime 10 righe

Opzioni utili di `g++`

`-std=c++14`	Segue le regole del linguaggio C++ standard 2014
`-Wall`	Stampa tutti i warning, inclusi quelli poco importanti
`-o` ⟨`nome`⟩	l'eseguibile si chiamerà ⟨`nome`⟩ anziché `a.out`
`-g`	Crea eseguibile contenente simboli leggibili per debugging

A.4 Debugging

Cheatsheet

Procedura di debugging

1) leggi i messaggi (suggerimenti)
2) stampe per isolare il punto
3) gdb: `g++ -g`
4) valgrind

Programmi utili per il debugging

`gdb` ⟨`...argomenti...`⟩	linea gdb
Comandi gdb	
`valgrind`	memoria (dettagli ed esempi forniti in CheatSheet della Parte 10)

Appendice B

Riepilogo introduzione a C++

B.1 Struttura del programma

- Un blocco di istruzioni è racchiuso fra parentesi graffe, aperta e chiusa

```
{
        // istruzione
        // istruzione
        // ...
}
```

- Un blocco di istruzioni può contenere altri blocchi al suo interno

```
{
        // istruzione
        {
                // istruzione
                // istruzione
        }
        // ...
}
```

- Ogni istruzione o blocco contenuti in un altro blocco va graficamente rientrato (*indentato*) di un numero fisso di caratteri per renderlo visivamente evidente.

  N.B. Nel linguaggio C++ questa non è una regola di sintassi ma solo di stile, però è così importante che altri linguaggi (Python) è invece obbligatoria.

- Un programma deve sempre contenere una funzione chiamata `main` che restituisce un valore di tipo `int` (che normalmente non ci interessa).

- L'intestazione di una funzione è seguita da un blocco di codice che costituisce il *corpo* della funzione

```
int main()
{
        // istruzione
        // istruzione
        // ...
}
```

B.2 Regole grammaticali

- I caratteri whitespace (spazio, tabulazione e a-capo) si possono usare a piacere. Ogni istruzione è quindi terminata da un punto e virgola `;` .

- Nel codice si possono inserire commenti mono-riga (da `//` a fine riga) o multi-riga (da `/*` a `*/`).

- Nella sintassi C++ le funzioni sono le componenti “attive” (agiscono). Le componenti “passive” su cui le funzioni agiscono sono:
  - Literal (costanti letterali) come `1`, `3.14159`, `'A'`, `"Hello, world"`.
  - Variabili, come `i`, `x`, `cateto`, `num_elementi`
  - Costanti con nome, come `PI`, `OREINUNGIORNO`, `MAXMEM`
- Sia le funzioni che le variabili e costanti sono identificate da un nome (o, appunto, *identificatore*)
- Le regole per comporre gli identificatori sono:
  - Lunghezza praticamente illimitata (dipende dal compilatore)
  - Solo caratteri alfabetici, numerici e underscore `_`
  - I caratteri alfabetici maiuscoli e minuscoli sono diversi
  - Il primo carattere non può essere numerico

  N.B. L’uso di identificatori tutti maiuscoli per le costanti è solo una possibile convenzione, non è una regola del linguaggio.

B.3 Dichiarazioni

- Gli oggetti con nome, prima di essere usati, vanno definiti attraverso una *dichiarazione*. Una dichiarazione si può mettere in qualunque punto del codice.
- In una dichiarazione deve essere indicato il tipo di un oggetto (o il tipo del valore restituito, per una funzione – vedi `main` sopra)
- Alcune dichiarazioni-tipo:
  - `int i; //variabile`
  - `const float PI = 3.14159; //costante`
  - `int main() //funzione`

B.4 Variabili, inizializzazioni, assegnazioni

- Una funzione che restituisce un valore di tipo `int` deve sempre terminare la sua esecuzione con l’istruzione “restituisci un valore di tipo intero” (per esempio `return 0`). Lo stesso vale per qualunque altro tipo. Lo standard ci concede un’eccezione (possiamo non fare alcun `return`) solo per la funzione `main`.
- Una dichiarazione di variabile può contenere anche una *inizializzazione* che stabilisce il valore iniziale della variabile: `float angolo = 90;`. Vedi caso costante.
- Una variabile non inizializzata contiene un valore (non esiste una variabile che contiene “niente”), ma questo valore non è prevedibile e non ha alcun senso.
- Successivamente all’inizializzazione, il valore di una variabile viene impostato con una istruzione di assegnazione, usando l’operatore di assegnazione `=` . P.es. `x = 10;` .
- L’assegnazione prevede una variabile sul lato sinistro dell’operatore `=`, un valore o una espressione più complessa sul lato destro. I ruoli non sono interscambiabili. L’espressione viene valutata, e al termine dell’esecuzione la variabile contiene il valore risultante.
- Inizializzazione ed assegnazione hanno sintassi simili ma non sono la stessa operazione. Con i tipi semplici questa differenza non si nota, ma occorre saperlo.

B.5 Scope e visibilità

- Una dichiarazione, e quindi l'oggetto che questa definisce, esiste (scope) ed è accessibile (visibility) nel blocco in cui è definita e in tutti gli eventuali blocchi contenuti in esso, a partire dalla riga che contiene la definizione.

  (Nota: Questo perché il compilatore legge il codice sorgente una volta sola, dall'inizio alla fine, e non può usare prima una definizione che incontrerà dopo)

- Le funzioni possono essere dichiarate solo nello "scope globale", cioè al di fuori di qualunque altra funzione.

- Se in un blocco viene dichiarata una variabile con lo stesso nome di un'altra definita nello **stesso blocco**, è errore di sintassi e la compilazione non ha successo.

```
{
        int i;
        int i; // ERRORE
}
```

- Se in un blocco viene dichiarata una variabile con lo stesso nome di un'altra definita in un blocco **più esterno** (che lo contiene), non è errore; la variabile più esterna però non è *visibile* nel blocco più interno

```
{
        int i;
        {
                int i; // non e' errore, ma in questo blocco esiste solo questa i
        }
}
```

B.6 Espressioni e operatori

- Una espressione è un costrutto linguistico del quale si può calcolare il valore.

- Una espressione è:

  – Una costante literal (es. `1`)

  – Una costante con nome (es. `PI`)

  – Una variabile (es. `x`)

  – Una chiamata di funzione (es. `sqrt(2)`, "square root")

  – Più elementi tra quelli elencati, combinati attraverso operatori

  Il valore dell'espressione è uno, indipendentemente dal numero di elementi e operatori, e ha un suo tipo. P.es. `2.1 + 3.9` ha il valore `6.0` che è un unico valore numerico floating point.

- Esistono molti operatori, ciascuno dei quali agisce su determinati tipi; quelli più intuitivi sono gli operatori aritmetici `+ - * /`, che operano su interi o su floating point. L'operatore modulo `%` opera solo su interi.

- Ogni operatore ha un determinato livello di precedenza. P.es. somma e sottrazione hanno un livello di precedenza più basso di moltiplicazione e divisione. Le parentesi tonde `( )` si possono usare per cambiare la precedenza in una espressione.

- In una espressione che include elementi di tipi compatibili ma misti, per esempio float+double o int—float, il tipo "meno capace" viene convertito (*promosso*) al tipo "più capace".

- Una espressione che include elementi di tipi non compatibili, per esempio intero*literal stringa, genera errore. Ogni operatore accetta certe combinazioni di tipi e non altre.

Dal codice sorgente al programma eseguibile

- Fasi del processo:
  1. Preprocessore: riceve il mio sorgente (uno o più) contenente direttive e codice C++, esegue direttive, produce un sorgente in solo codice C++
  2. Compilatore: riceve sorgente in solo codice C++ (uno o più), traduce in linguagio macchina, produce file "oggetto"
  3. Linker/loader: riceve file oggetto, traduce riferimenti simbolici in indirizzi di memoria, unisce moduli di libreria standard (o anche altre se da me indicate); produce un unico file eseguibile
- Comando da usare con il compilatore "GCC – Gnu Compiler Collection":
  - minimale (produce a.out): `g++ miosorgente.cpp`
  - voglio dare un nome al file eseguibile: `g++ miosorgente.cpp -o mioeseguibile`
  - voglio compilare più file: `g++ miosrg1.cpp miosrg2.cpp -o mioeseg`

  N.B. Il comando `g++` esegue automaticamente tutti i tre passi indicati sopra.
- Per eseguire il mio eseguibile: `./mioeseguibile`
- Per usare una funzione o altro, che sia definito in un modulo della libreria, occorre che il sorgente includa le relative definizioni – esattamente come per gli oggetti creati da me. Le definizioni si trovano in *file header* forniti con la libreria.
- Direttiva del preprocessore per includere uno header di libreria: `#include`
- Sintassi per libreria installata nel sistema: `#include <iostream>`
- Sintassi per header mio contenuto nella mia cartella dei file sorgenti: `#include "mioheader"`

Input/output

- Le funzionalità sono fornite dal modulo "iostream" della libreria standard, quindi occorre `#include <iostream>`
- Ingresso dall'"oggetto" cin: `cin >> var1 >> var2;`
- Uscita sull'"oggetto" cout: `cout << var1 << var2;`
- Gli identificatori nella libreria standard sono "racchiusi" nel namespace `std`, quindi
  - o uso ogni volta l'operatore `::` per riferirmi al namespace corretto: `std::cin`, `std::cout`, `std::endl`
  - oppure una volta per tutte indico `using namespace std;` a inizio programma

B.7 Errori comuni

- Usare variabile non dichiarata
- Usare variabile prima di dichiararla
- Usare variabile con un nome simile, ma non uguale, a quello nella dichiarazione
- Dimenticare il punto e virgola
- Dimenticare il qualificatore di namespace, o l'istruzione `using namespace ...`
- Il literal "stringa vuota" `""` esiste. Il literal "carattere vuoto" `''` non esiste.