Introduzione alla Programmazione
a.a. 2022/23

Eserciziario

19 dicembre 2022

Indice

Come usare l'eserciziario

In questo eserciziario trovate già gli esercizi per tutto l'anno. 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 facciate 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).

Per ogni laboratorio vi diremo fino a che punto potete arrivare, sulla base del materiale visto a lezione fino a quel momento.

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, simile ai "foglietti" usati per copiare agli esami, 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 urlata da un
  utente.

  // fare output di un messaggio che chiede di urlare una parola
  // dichiarare una variabile msg di tipo string
  // leggere dallo standard input msg
  // fare output di 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 urlata
  // da un utente.

  // fare output di un messaggio che chiede di urlare una parola
  cout << "Urla una singola parola (senza spazi)";
  // dichiarare una variabile msg di tipo string
  string msg;
  // leggere dallo standard input msg
  cin >> msg;
  // fare output di 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 (cout)` e
  - `leggere` come sinonimo di `leggere da standard input (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 e sorgente2.cpp -o esempio`
- Vedere i cheatsheet nell'Appendice per altri casi d'uso dei comandi indicati.

Elementi fondamentali di programmazione

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, quali aspetti ritenete vincenti? provate a elaborare il vostro algoritmo ideale.

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. Il programma deve essere scritto in un file chiamato `agecalc.cpp`.

9. Scrivere un programma che calcola perimetro e 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. 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). [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 o minore del valore assoluto della loro differenza. [File `triangleinequality.cpp`]

7. Scrivere un programma che chiede all'utente un numero reale e lo legge.

   Quindi, 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. [File `leapyear.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 e restituisce in output chi ha vinto. [File `morra.cpp`]

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

    - "Freddo dannato" 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...). 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 per difetto a 15 minuti. Il programma deve prendere in input due valori interi, uno tra 1 e 12 (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 “una") e dal valore dei minuti, approssimato al quarto d’ora (p.es. se è 18 scrive “ e un quarto", se è 39 scrive “ e mezza", se è 55 scrive “ e tre quarti"; se è 0 invece non scrive niente). Infine, se i minuti non sono divisibili esattamente per 15, scrive “ circa". [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` → da capo

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 666
// 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. Scrivere un programma che gioca più mani di morra cinese. Alla fine di ogni mano deve 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 positivo e ne stampa il numero di cifre (in base 10). Ad esempio su 27458 stampa 5. [File `numdigits.cpp`]

11. Scrivere un programma che chiede all'utente di inserire e leggere numeri interi e poi chiede all'utente se vuole continuare e legge la risposta finché l'utente non risponde di no. Finito il ciclo di lettura stampa la media dei numeri letti. [File `mediainterattiva.cpp`]

12. Scrivere un programma che legge due numeri interi strettamente positivi 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 sulla diagonale ha $2 * n + 1$ caratteri. Ad esempio su 8 stampa

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

    che sulla diagonale 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.]

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`]

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. [File `ispalindrome.cpp`]

16. Scrivere un programma che chiede all'utente un numero reale e lo legge. Poi chiede all'utente di provare a indovinarne la radice quadrata 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.
    Per provare questo programma 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 `guesstheroot.cpp`]

17. Scrivere un programma che chiede all'utente un numero intero maggiore di 1 e ne stampa la scomposizione in fattori primi. Ad esempio su 392 stampa `"392 = 2^3 * 7^2"`, usando il carattere `'^'` per rappresentare l'elevamento a potenza. [File `primefactors.cpp`]

3.3 Esercizi più avanzati

18. Scrivere un programma che verifica se un numero intero positivo dato in input è un *numero di Armstrong*. 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 `armstrongnum.cpp`]

19. 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`]

20. 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. **Crea un array di prova con elementi di tipo `int`.** Scrivere un programma che dichiara un array `v` 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
   ```

   Crea un array di `float`. Scrivere un programma uguale al precedente, ma che lavora su array di `float`.

   [File `creaArrayFloat.cpp`]

2. **Stampa un array di interi.** Scrivere un programma che, dati un array `a` di `int` e la sua lunghezza `N`, stampa tutto l'array.

   La prima parte dell'algoritmo è preparatoria, e coincide con l'esercizio precedente. La seconda svolge il compito richiesto.

   [File `stampaArrayInt.cpp`]

   ```
   // Creare e popolare un array a di lunghezza N.

   // per la variabile intera i che va da 0 incluso a N escluso:
   //      stampare l'elemento i-esimo di a
   //      stampare un a-capo
   ```

   Stampa un array di `float`. Scrivere un programma uguale al precedente, ma che lavora su array di `float`.

   [File `stampaArrayFloat.cpp`]

3. **Leggi un array di `int` da tastiera.** Scrivere un programma che dichiara un array `v` 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 da:
//         - la stringa "Valore n. "
/          - il valore di i
//         - il separatore ": "
//      Leggere da input un valore di val
//      Assegnare all'elemento i-esimo di a il valore di val
```

Leggi un array di `float` da tastiera. Scrivere un programma uguale al precedente, ma che lavora su array di `float`.

[File `leggiArrayFloat.cpp`]

4. 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''
// 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 v a sum
*/
// stampare la divisione di sum per N
```

5. Scrivere un programma che dati un array `a` di `float`, la sua lunghezza `N`, e un valore intero `i` tra `0` e `N`, memorizza nell'elemento `i`-esimo il valore $\frac{1}{2}i^2$. Poi stampa l'array.

[File `parabola.cpp`]

```
// creaArrayInt
// per j che va da 0 incluso a N escluso:
//      assegnare all'elemento j-esimo di a il valore N-j
// leggere da input un valore di i
// scrivere nell'elemento i-esimo di a il valore i quadrato mezzi
// stampare l'array (vedi algoritmo precedente)
```

[SUGGERIMENTO: L'espressione 1/2 con 1 e 2 interi ha valore 0. Però se anche solo un termine in una espressione ha tipo floating point, tutta l'espressione viene convertita in floating point. La costante 2 se scritta 2.0 o anche 2. è una costante in floating point.]

4.2 Esercizi di base

6. Scrivere un programma che legge `N` interi in un array `a` di `int` (vedi `leggiArrayInt`). Quindi stampa il valore massimo contenuto nell'array `a` e il numero di volte in cui questo appare. [File `maxInt.cpp`]

7. 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`]

8. 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`]

9. Scrivere un programma che, usando l'algoritmo "crivello di Eratostene", trova i numeri primi minori di 1000.

   [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 varra' true
se i e' primo, false altrimenti

2) Inizialmente, sia p pari a 2, il numero primo pi\`u piccolo.

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] \`e true), oppure finché non \`e finita la lista

5) Se la lista \`e  finita, stop. Altrimenti p diventa pari al numero trovato
   e si ricomincia (la prima volta sar\`a 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` e' `true`.

   [SUGGERIMENTO: Come visto a lezione, bisogna fare diversi cicli for:

   (a) Un ciclo su tutti gli elementi di `isprime`, per impostarli a `true`

   (b) Un ciclo sugli elementi di `isprime`, per cercare il primo elemento `true`

   (c) ALL'INTERNO DEL CICLO PRECEDENTE, un ciclo su tutti i multipli di `p` per marcarli `false`

   (d) Un ciclo finale per stampare gli elementi `true`

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

10. 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`, e lo riempie di valori 0.

    [File `zeros.cpp`]

11. 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, 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

12. 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`.]

13. 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.]

14. Scrivere un programma che legge un array di `int` e stampa il secondo valore più elevato.

    [File `secondo.cpp`]

15. 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`]

16. Scrivere un programma che legge un array di `int`, riordina i suoi elementi in modo crescente, e poi lo stampa.

    [File `sortInPlace.cpp`]

17. 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.]

18. 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 ternine 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.]

19. Scrivere un programma come il precedente, ma realizzato senza usare array ausiliari (dovete usare un algoritmo *in place*).

    [File `uniqueInPlace.cpp`]

20. 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 11.

    [File `tabellineTriang.cpp`]

Parte 5

Array: approfondimenti

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: ``Trovato in posizione ...''
// altrimenti scrivere ``Non 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`]

```
// 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 variabii 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 variabii 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. Modificare i precedenti esercizi 1 e 2 nel seguente modo:

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

   (b) Ogni volta che nel programma compare un riferimento a un elemento dell'array, incrementino count

   (c) Alla fine, stampino il valore di count (numero di accessi effettuati all'array)

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

8. Modificare i programmi fatti fino a qui perché

   (a) Chiedano all'utente il nome di un file da leggere

   (b) Leggano il contenuto degli array dal file

   La quantità di dati presenti nei file deve naturalmente essere adeguata per riempire gli array.

9. 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. [File `det2.cpp`]

10. 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:

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

]

11. Scrivere un programma che dichiari tre vettori di interi s1, s2 e d, il terzo dei quali abbia dimensioni pari alla somma delle dimensioni degli altri due.

    Dopo aver ordinato s1 e s2 in maniera crescente ([HINT: utilizzate il codice sviluppato negli esercizi precedenti!]) la funzione deve riempire d in modo che contenga tutti gli elementi dei due array ordinati tra loro in modo crescente. Quindi stampa l'array d risultante.

    [File `mergeArrays.cpp`]

    Attenzione: nel costruire il contenuto di d tenete conto del fatto che i due vettori di partenza sono stati precedentemente ordinati.

    Ad esempio, se s1 contiene gli elementi
    2 5 9 14 15 20 25 27 30
    e s2 contiene gli elementi
    3 5 10 11 12 22 23 24 26 27
    la funzione riempie d con i seguenti elementi
    2 3 5 5 9 10 11 12 14 15 20 22 23 24 25 26 27 27 30.

12. Fissata una costante intera positiva N, scrivere un programma che, preso in ingresso un numero intero positivo minore di 2^N, memorizzi la sua rappresentazione binaria su un array di lunghezza N. Quindi stampi il contenuto dell'array.

    [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.

13. Scrivere il programma che esegua la traslazione ("shift") verso sinistra degli elementi di un vettore letto in ingresso, 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] $\longrightarrow$ [10 15 18 0]

    [File `shiftLeft.cpp`]

14. Scrivere un programma che esegua la rotazione ("shift") verso destra degli elementi di un array letto in ingresso, 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] $\longrightarrow$ [18 1 10 15]

    [File `rotateRight`]

15. 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 $|N| \geq$ 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 $|N| == k \times$lunghezza array (k intero), rotate creerà un vettore uguale a quello dato, ovvero nessuna modifica.

16. 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

17. 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`

18. 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`]

19. 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;
```

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 una funzione 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)

   Scrivere un programma che legge una data D1 e, dopo averne verificato la correttezza 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`. Calcola poi area e perimetro del triangolo e memorizza le informazioni nei corrispondenti campi della struct. Infine verifica se il triangolo è o no un triangolo rettangolo e stampa un messaggio di output opportuno.
     [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
   - verifica se i lati hanno tutti la stessa lunghezza
   - verifica se la spezzata è chiusa, e se lo è scrive un messaggio: "La linea è chiusa e quindi definisce un poligono". Se i lati hanno tutti la stessa lunghezza, aggiungere: "regolare".

   [File `poly1.cpp`]

   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]

8. 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`, ne verifica la correttezza e calcola il tempo trascorso tra i due orari, assumendo che si riferiscano allo stesso giorno. [File `timeDiff.cpp`]

6.3 Esercizi più avanzati

9. 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`]

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`]

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 sia quello atteso. In questo programma avremo cura di fare solo chiamate passando argomenti corretti.

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 fumzioni 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!`

- Una 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. Se zero argomenti: parentesi vuote, ma comunque necessarie.
- 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

Cheatsheet

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.

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()
   {
     std::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)
   {
     std::cout << ``Hello, we are " << n << std::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 (letterale)
// chiamare la funzione givemefive stampando su std::cout il valore restituito
// chiamare la funzione prossimo passandole come argomento il valore 4...
//                                        ... e stampando su std::cout il valore restituito
// chiamare la funzione somma passandole come argomenti i valori 2 e 3...
//                                        ... e stampando su std::cout 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
   //      - 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

   ```
   // 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
   ```

   [File `testf_average.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 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
//      dichiarare la variabile float a
//      chiamare la funzione area assegnando ad a il valore che restituisce
//      stampare l'area
```

[File `testf_area.cpp`]

5. Scrivere una funzione senza argomenti che legge lettere maiuscole finché l'utente non inserisce un carattere che non è una lettera maiuscola, e restituisce la prima in ordine alfabetico (ovvero quella che numericamente è la minima).

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

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

```
// stampare la stringa "La lettera più piccola inserita è "
// stampare il risultato della chiamata di first_letter
```

[File `testf_first_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 num 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`:

```
// 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
```

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).

[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 funzone 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
}
```

[File `testf_replicate.cpp`]

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).

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`]

9. Questo esercizio illustra 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`]

- Scrivere un programma, per testare le funzioni implementate, 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`]

7.2 Esercizi di base

11. Scrivere una funzione con un argomento num di tipo intero che restituisce il numero di cifre (in base 10). Ad esempio su 27458 restituisce 5.

    [File `testf_ndigits.cpp`]

12. Scrivere una funzione senza argomenti che chiede all'utente di inserire e legge numeri interi e poi chiede all'utente se vuole continuare e legge la risposta finché l'utente non risponde di no. Finito il ciclo di lettura restituisce la media dei numeri letti (di tipo float).

    [File `testf_online_average.cpp`]

13. 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`.]

14. 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`]

15. 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
    ```

    [File `testf_replicate2_line.cpp`]

16. 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) usate la funzione `replicate2_line`]

17. 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`.]

18. 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`]

19. 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`]

20. 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 usando la funzione `dist`, e stampa un messaggio del tipo "La lunghezza della spezzata è xxx".

    [File `testf_poly1.cpp`]

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

    All'uscita, `a` e `b` devono avere i loro valori scambiati.

    [File `testf_swap.cpp`]

22. 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`]

7.3 Esercizi più avanzati

23. 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`]

24. 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`]

25. Scrivere una funzione con un argomento intero `n` compreso fra 1 e 3000 e lo stampa in notazione romana.

    [File `testf_roman.cpp`]

26. 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.

27. 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`]

28. 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 0 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.

    [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.]

29. Analogamente al punto precedente, scrivere una funzione che prende, oltre ai parametri della funzione al punto 28, 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.]

30. 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`]

31. 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`]

Argomenti di programmazione in C++

Parte 8

Puntatori - senza allocazione dinamica di memoria

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

- Puntatore nullo `nullptr`
- Operatore di referenziazione `&`. Opera su una variabile (o altro valore sinistro), prendendone l'indirizzo e trasformandolo in un puntatore. Esempio: `&a;` = l'indirizzo di `a`.
- Definire 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>;`

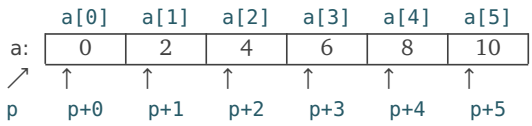
    Esempi: `char *p = nullptr; char *q = p;  float *p = &a;` dove `a` è una variabile di tipo `float`.

  Attenzione: Usare dichiarazioni separate per variabili separate: `T *p,; T *q;` anziché `T *p, *q;`.Consiglio valido in generale, non solo per puntatori.

  Se abbiamo definito `T *p;`, per assegnarle un valore useremo `p = &a;`.
- Accedere al valore della memoria puntata da un puntatore `p`: `*p`.

  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`)
- Dimensione in memoria di qualcosa che ha tipo `T`: `sizeof T`
- Dimensione in memoria di una variabile `a`: `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;` corrisponde a



  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 `*`

8.1 Esercizi di riscaldamento

1. 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 ==*.

   [File `provaptr.cpp`]

```
// dichiarare due variabili s1 e s2 di tipo string inizializzate rispettivamente...
// ... a "Hello" e "World"
// stampare il messaggio "Debug: ", gli indirizzi di s1 e di s2 e andare a capo
// stampare s1 e s2 e andare a capo
// dichiarare una variabile p di tipo puntatore a string inizializzata con...
// ... l'indirizzo di s1
// stampare il messaggio "Debug: ", il valore di p e il valore ...
// ...dell'area di memoria puntata da p e andare a capo
// assegnare all'area di memoria puntata da p la stringa "Ciao"
// assegnare a p l'indirizzo di s2
// stampare il messaggio "Debug: ", il valore di p e il valore ...
// ...dell'area di memoria puntata da p e andare a capo
// assegnare all'area di memoria puntata da p la stringa "Mondo"
// stampare s1 e s2 e andare a capo
```

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

2. Riprendete la vostra implementazione della funzione `reverse` dell'esercizio 8 in sezione 4 e provate a modificare il programma di test utilizzando `source` per instanziare entrambi i parametri di `reverse`. Se avete implementato `reverse` 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 `reverse` 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 17 in sezione 4. Però, in situazioni più complesse potrebbe essere impossibile risolvere i problemi in caso di aliasing fra i parametri.

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

   Modificare il programma di test in modo che esegua la chiamata `reverse(source, source)` e che venga catturata l'eccezione (stampate un messaggio di errore che dice che non si può chiamare `reverse` usando la stesso parametro attuale per entrambi i parametri formali).

   [File `alias.cpp`]

3. Scrivere una funzione che prende come argomenti 3 variabili di tipo char, propone all'utente di sceglierne una e ne restituisce l'indirizzo (ad esempio, per poterla modificare nel main).

```
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 "
// 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 "
// - 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 "
// --  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"
// confrontare selected con l'indirizzo di ch2 e se sono uguali
// - stampare il messaggio "hai scelto ch2"
// confrontare selected con l'indirizzo di ch3 e se sono uguali
// - stampare il messaggio "hai scelto ch3"
// stampare il messaggio "Inizialmente ch1==" seguito da ch1
// stampare il messaggio ", ch2==" seguito da ch2 e ", ch3==" seguito da ch3
// stampare un'andata a capo
// stampare il messaggio "ora cancello la variabile che hai scelto"
// assegnare uno spazio all'area di memoria puntata da selected
// stampare il messaggio "Ora ch1==" seguito da ch1, da ", ch2==" seguito da ch2
// e da ", ch3==" seguito da ch3 e un'andata a capo
```

[File `selectvar.cpp`]

4. 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 vv[N]) {
// stampare il messaggio "Dimensione del parametro == " seguito ...
// ...dalla dimensione di vv 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
```

[File `arrayptr.cpp`]

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

5. Modificare il programma che legge `N` valori reali, li memorizza in un array di lunghezza `N`, e ne restituisce la media richiesto dall'esercizio 4 della sezione 4 usando l'aritmetica dei puntatori per migliorarne l'efficienza, secondo il seguente schema

```
// dichiarare una costante N di tipo int inizializzata ad un valore...
// ... strettamente maggiore di 0
// dichiarare un array v di N interi
// dichiarare un puntatore a interi p inizializzato con v
/* iterare su i a partire da 0 e fino a N-1
        - leggere un valore intero memorizzandolo nella cella puntata da p
        - incrementate p
*/
// 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
*/
// stampare la divisione di sum per N
```

[File `averageptr.cpp`]

6. Scrivere una funzione `printarray`, che restituisce void, che prende due argomenti: un array di caratteri `s` e la sua dimensione `size`.

   Usare un puntatore per stampare tutti gli elementi dell'array.

   [File `testf_printarray.cpp`]

```
void printArray(const  char s[], int size) {
// definire un puntatore p al tipo const char inizializzato con s;
// iterare da 0 fino a size
// stampare il contenuto dell'indirizzo puntato da p ed incrementare p
}
```

   Realizzare un programma di test:

```
// definire un array di caratteri msg inizializzato con "Hello, world";
// definire una variabile N inizializzata al valore sizeof msg/sizeof(char)
// chiamare la funzione printArray
```

8.2 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. Modificare l'esercizio 3 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.

8. Scrivere una funzione 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']`, in `v`, usando l'aritmetica dei puntatori per scorrere `v`.

   [File `selectvarfn.cpp`]

8.3 Esercizi più avanzati

9. Considerate la funzione `selectVar` nella sua forma originale (esercizio 3) e in quella migliorata (esercizio 7). È possibile modificare il codice in modo da non usare puntatori ma solo variabili di tipo riferimento?

   [File `selectvarref.cpp`]

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

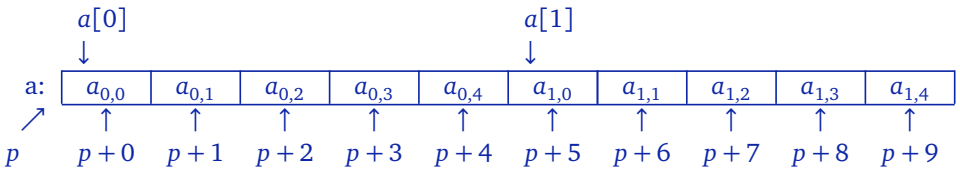
    [File `selectNvar.cpp`]

    [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.]

11. 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`.]

12. 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 9

Puntatori - allocazione dinamica di memoria

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`, 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`, 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).

9.1 Esercizi di riscaldamento

1. Scrivere un programma che implementa il seguente algoritmo:

```
// dichiarare una costante N intera inizializzandola a un valore moderato...
// ...(p.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 le funzioni necessarie a leggere e stampare un `array` dinamico di interi, ovvero un `array` la cui dimensione viene fissata al momento dell'esecuzione (invece che della dichiarazione). Per rappresentare gli `array` dinamici di interi utilizzeremo il seguente tipo:

   `dynamic_array`:

```
struct dynamic_array {
        int* 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 8, 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.

   (a) Scrivere una funzione per la lettura di un `array` dinamico di interi

```
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 del vettore" e andare a capo
```

```
        // leggere s
*/
// assegnare s al campo size di d
// allocare s interi assegnando l'area di memoria riservata al campo store di d
/* iterare s volte....
        // stampare "inserisci un valore"
        // leggere un valore 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.

(b) Scrivere una funzione per la stampa di un `array` dinamico di interi

```
void print_d_array(const dynamic_array& d) {
// definire un puntatore p e inizializzarlo con il campo array di d
// usando l'aritmetica dei puntatori su p per visitare il campo store di d...
// ...stampare gli elementi del campo store di d, ...
// ...ciascuno seguito dal carattere '\t'
// stampare una andata a capo
}
```

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

[File `d_array.cpp, d_array.h, testf_d_array.cpp`]

9.2 Esercizi di base

In questa sezione completeremo le funzioni per una mini-libreria che gestisca `array` dinamici. Per ciascuna funzione scrivere (ove possibile) un programma di test.

Aggiungere le funzioni ai file specificati all'esercizio 2b: [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 sar anche necessario crearlo).

3. Scrivere la funzione `delete_d_array` che riceve un argomento, il riferimento ad un `array` dinamico `d`, e lo *svuota*, 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). Valore restituito: nessuno.

   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 (http://www.valgrind.org/) che trovate già installato sulle macchine di laboratorio.

   [File `testf_delete_d_array.cpp`]

4. Scrivere la funzione `create_d_array` che riceve tre argomenti: il riferimento ad un `array` dinamico `d`, la sua dimensione `s` (intero) e un valore `v` (intero). La funzione inizializza `d` in modo che il suo `size` sia `s` e che il suo `store` sia un blocco di dimensione `s` in cui tutti gli elementi sono inizializzati al valore `v`.

   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.

   Valore restituito: nessuno.

   [File `testf_create_d_array.cpp`]

5. Scrivere la funzione `set` che riceve tre argomenti: un riferimento ad un array dinamico `d`, un indice (intero) `index` ed un valore `value` (intero). 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).

   Valore restituito: nessuno.

   [File `testf_set.cpp`]

6. Scrivere la funzione `get` 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.cpp`]

7. Scrivere la funzione `SelectionSort` che riceve un argomento, un riferimento a un array dinamico `d`.

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

   Valore restituito: nessuno.

   [File `testf_selectionsort.cpp`]

9.3 Esercizi più avanzati

In questa sezione produrremo una implementazione-giocattolo di `std::vector` chiamata `myvector`.

Un oggetto di tipo `myvector` 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 intero `store`, che conterrà l'indirizzo di un'area di memoria allocata in modo da contenere `capacity` interi.

Le funzioni richieste sul tipo `myvector` sono specificate negli esercizi seguenti, e vanno scritte in:
[File `myvector.h, myvector.cpp`]

Come nella sezione precedente, fate un file di test per ogni esercizio.

Notate che l'ultimo esercizio richiede di fare un upgrade a tutti i file: create una nuova versione della vostra libreria `myvector` come specificato.

Nota. Nella specifica delle funzioni richieste viene spesso indicato che un parametro di tipo `myvector` 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 `myvector` che state utilizzando, né che sia di dimensione pari a `capacity`. Potete solo controllare la consistenza fra `size` e `capacity` e che `size` sia non negativa e `capacity` positiva. Un uso delle funzioni con `myvector` che non sono stati inizializzati correttamente, quindi potrà causare errori impredicibili fra cui *segmentation fault* (se la funzione cerca di accedere ad un areea di memoria che non è stata riservata per il `myvector`).

8. Scrivere la funzione `void create(myvector& v, int capacity);` che dato per riferimento un `myvector v` non ancora inizializzato, fa l'allocazione di `v.store` con lunghezza `capacity`, assegna `v.capacity = capacity`, e assegna `v.size = 0`.

   [File `testf_set.cpp`]

9. Scrivere la funzione `push_back` che riceve due argomenti, un riferimento a un `myvector v` inizializzato e un intero `x`.

   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`.

   Se `v` è già completamente pieno (cioè `v.size == v.capacity`), la funzione solleva un'eccezione di tipo opportuno.

   Valore restituito: nessuno.

   [File `testf_push_back.cpp`]

10. Scrivere la funzione `pop_back` che riceve un argoment, un riferimento a un `myvector v` inizializzato.

    Se `v` è vuoto (cioè `0 == v.size`), la funzione solleva un'eccezione di tipo opportuno.

    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`.

    [File `testf_pop_back.cpp`]

11. Scrivere la funzione `void set(myvector& v, int value, int index);` che assegna `value` all'elemento con indice `index` di `v.store`. Solleva una eccezione (di tipo opportuno) se `index` non è un valore corretto, ovvero se non è compreso fra `0` e `v.size`.

    [File `testf_vset.cpp`]

12. Scrivere la funzione`int get(const myvector& v, int index);`.

    La funzione restituisce il valore presente in `v.store` all'indice `index`. Solleva una eccezione (di tipo opportuno) se `index` non è un valore corretto, ovvero se non è compreso fra `0` e `v.size`.

    [File `testf_vget.cpp`]

13. Scrivere la funzione `void destroy(myvector& v);`.

    La funzione dealloca `v.store` e pone a zero sia `v.size` che `v.capacity`.

    Valore restituito: nessuno.

    [File `testf_destroy.cpp`]

14. Scrivere la funzione `resize` che riceve due argomenti: un riferimento a un `myvector 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`,
    - copiarvi i valori contenuti in `v.store` (se ci stanno, mentre se `v.size>new_capacity` verranno copiati solo i primi `v,size`),
    - liberare lo spazio precedentemente occupato da `v.store` (per evitare *memroy leak*)
    - assegnare a `v.store` il nuovo blocco di memoria.
    - aggiornare `v.capacity` e `new_capacity` (e `v.size` (se necessario)).

    Se `new_capacity` non è strettamente positiva, la funzione solleverà un'eccezione di tipo opportuno.

    Valore restituito: nessuno.

    [File `testf_resize.cpp`]

15. Scrivere la funzione `safe_push_back`, analoga alla `push_back`, ma che in caso il `myvector` sia già completamente riempito invece di sollevare un'eccezione raddoppia la capacità del parametro e inserisce il valore dato (che a questo punto può trovare posto nello `store`).

    [Hint: utilizzate la funzione `push_back` 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 e poi nuovamente la funzione `push_back` per inserire l'elemento che a questo punto vi troverà posto.]

    [File `testf_safe_push_back.cpp`]

16. Scrivere la funzione `bool looks_consistent(const myvector& v)` 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` restituisce `false` su un `myvector 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.cpp`]

17. Scrivere la funzione `BubbleSort` che riceve un argomento, un riferimento a un `myvector v`.

    La funzione ordina il suo argomento secondo l'algoritmo *BubbleSort*.

    Valore restituito: nessuno.

    [File `testf_bubblesort.cpp`]

18. 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 `myvector` è evidentemente scorretto (cioè se `looks_consistent(v)` è falso) e in tal caso sollevino un'eccezione di tipo opportuno.

    [File `safe_myvector.cpp, safe_myvector.h, testf_safe_myvector.cpp`]

Parte 10

Vector

In questa sezione ci concentriamo sull'utilizzo degli `std::vector`, imparando a maneggiarli e a definire funzioni che li utilizzino.
NOTA. Dove necessario, gestire gli errori con la definizione di opportune eccezioni.

Cheatsheet

`std::vector<T> V`	Dichiara la variabile `V` di tipo | std::vector| con elementi di tipo `T`
`std::vector<T> V(A)`	Crea un `std::vector V` di `T` ed inizializza i suoi elementi con un array `A` pre-esistente
`std::vector<T> W(V)`	Crea `W` e lo inizializza con il contenuto di un altro `std::vector<T> V` ("costruttore di copia")
`std::vector<T>v(N)`	Crea un `std::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=std::string`)
`std::vector<T> V(N, val)`	Crea un `std::vector V` di `N` elementi inizializzati ad un valore `val`
`V.size()`	Lunghezza di `V` (numero elementi)
`V.capacity()`	Spazio occupato da `V`, include spazio non ancora utilizzato
`V.max_size()`	Dimensione massima possibile
`V[i]`	Accede all'elemento `i`-esimo (segmentation fault se `i>=V.size()` o `i<0`)
`V.at(i)`	Accede all'elemento `i`-esimo (errore trattabile se `i>=V.size()` o `i<0`)
`V.back()`	Accede all'ultimo elemento (possibile errore se `V` è vuoto)
`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 lunghezza `N`
`V.clear()`	Svuota (porta a lunghezza 0)

`using namespace std;` in testa al file per poter omettere il prefisso `std::`

10.1 Esercizi di riscaldamento

1. Scrivere due funzioni `void readVector(std::vector<int>& v)` e `void printVector(const std::vector<int>& v)` che permettano di riempire `v` con `N` valori letti (`N` positivo) e stamparli:

```
void readVector(std::vector<int>& v) {
// Stampare "Inserisci la dimensione della sequenza: "
// Dichiarare una variabile intera N
// Leggere N
// iterare finchÃ N non Ã¨ 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 std::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 std::vector vect di interi
// chiamare la funzione readVector su vect
// chiamare la funzione printVector su vect
```

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

Variante: modificare la funzione `readVector` in modo che l'utente, invece di inserire il numero di elementi da leggere N seguito dagli elementi stessi, inserisca direttamente i valori concludendo con il carattere 'y', ad esempio:

```
12
5
10
8
y
```

[File `iovector_alt.cpp`]

2. Scrivere una funzione `void SelectionSort_vector(std::vector<int>& v)` che effettui l'ordinamento di `v` secondo l'algoritmo *SelectionSort*

```
void SelectionSort_vector(std::vector<int>& v) {
// dichiarare una variabile int greatestIndex
/* iterare sul std::vector dalla prima all'ultima posizione
        - memorizzare in greatestIndex la posizione corrente (sia i)
        - *//* iterare sul std::vector 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
*/
}
```

Scrivere un programma per testare la funzione `void SelectionSort_vector(std::vector<int>& v)` secondo il seguente algoritmo

```
// dichiarare un std::vector vect di interi
// chiamare la funzione readVector su vect
// chiamare la funzione printVector  su vect
// chiamare la funzione SelectionSort_vector su vect
// chiamare la funzione printVector su vect
```

[File `selectionsort_vector.cpp, selectionsort_vector.h, testf_selectionsort_vector.cpp`]

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

```
int  SequentialSearch_vector(const std::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 \`e 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 std::vector<int>& v, int item )`

[File `sequentialsearch_vector.cpp, sequentialsearch_vector.h, testf_sequentialsearch_vector.cpp`]

Variante: modificare il corpo della funzione `SequentialSearch_vector` secondo lo schema seguente:

```
int  SequentialSearch_vector(const std::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?

[File `sequentialsearch_vector_alt.cpp`]

10.2 Esercizi di base

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

   Scrivere un programma di test che legge alcuni interi strettamente positivi in `std::vector<int> source`, si interrompe al primo negativo, e poi chiama `reverse`, assegnando il risultato a un altro `std::vector<int> dest`. Quindi stampa su una riga `source` e sulla riga seguente `dest` utilizzando la funzione `printVector`.

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

5. Scrivere una funzione `cat` che riceve due argomenti `v1` e `v2`, di tipo `std::vector<int>`.

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

   Valore restituito: `v12`.

   Scrivere un programma di test che legge interi strettamente positivi in due `std::vector<int> first` e `second`, Quindi chiama `cat` per concatenare i due vettori e memorizza il risultato in `std::vector<int> total`, stampando infine i tre elenchi contenuti in `first`, `second` e `total` .

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

6. Scrivere una funzione `insert` che riceve tre argomenti: `v` di tipo `std::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`$\in$ [`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 `std::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`]

7. Implementare le funzioni che compongono una libreria per la memorizzazione e gestione di una rubrica telefonica utilizzando `struct` e `std::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.

   [SUGGERIMENTO: 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, perché se poi dovete fare modifiche alle funzioni precedenti o vi accorgete di un caso che non avete provato, così vi trovate ancora il codice di test pronto, basta togliere i commenti.
   Per quanto riguarda l'input dei dati su cui provare le chiamate di funzione, potete fare lettura da input (più facile, ma ci mettete più tempo a far girare il programma) oppure da file (richiede più sforzo la prima volta, ma vi permette di ripetere i test molto rapidamente e senza ulteriore fatica).]

   (a) Definire una struct `Contact_Str` contenente almeno i campi `Name`, `Surname`, `PhoneNumber` (di un tipo opportuno).

   (b) Creare un tipo *vettore di contatti* dandogli il nome `PhoneBook`, usando `typedef`: `typedef std::vector<Contact_Str> PhoneBook`

   (c) Scrivere la funzione `void add(PhoneBook& B, string surname, string name, int phoneNumber)` per aggiungere un contatto `C` in coda alla rubrica `B`

   (d) Scrivere la funzione `void print(const PhoneBook& B)` per stampare il contenuto della rubrica `B`.

   (e) Scrivere una funzione `void sortSurnames(PhoneBook& B)` che data una rubrica ordini alfabeticamente gli elementi in essa contenuti rispetto al campo `Surname`

   (f) Scrivere una funzione `int FindPos(const PhoneBook& r, string S)` 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 l'indice di tale contatto nella rubrica (ossia nel vettore)
   - Se nella rubrica **non esiste** un contatto `C` il cui campo `C.Surname` è uguale all'argomento `S`, allora restituisca l'indice del contatto che sarebbe quello immediatamente precedente in ordine alfabetico.

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

   (g) Scrivere una funzione `void Shift_PhoneBook(PhoneBook& B, int pos)` 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+1.

   (h) Scrivere la funzione `bool add_ord(PhoneBook& B, string surname, string name, int phoneNumber)` che inserisce il nuovo contatto nella rubrica nella posizione giusta rispetto all'ordine alfabetico.
   [SUGGERIMENTO: Assumendo che la rubrica sia ordinata, usare la funzione `FindPos` per ottenere la posizione immediatamente precedente a quella in cui il contatto andrebbe inserito, seguita dalle funzioni `Shift_PhoneBook` e una assegnazione.]

   [File `phonebook.cpp, phonebook.h, testf_phonebook.cpp`]

10.3 Esercizi più avanzati

8. Ancora sul **Crivello di Eratostene**: scrivere la funzione `std::vector<int> primes(int n)` che dato un intero positivo `n` restituisce un `std::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 9).

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

9. **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 11

Librerie

In questa sezione i costrutti visti finora verranno usati per 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
  - 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>`.

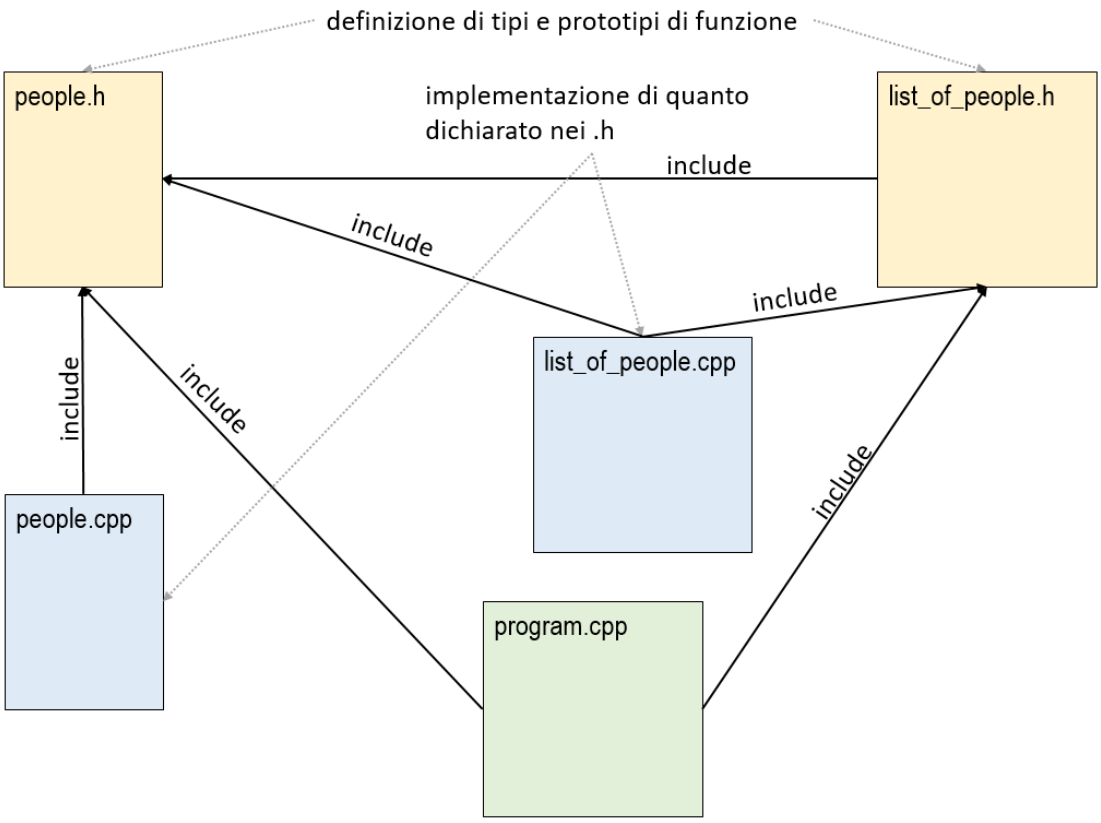
  Promemoria: Dove uso una libreria, includo il relativo header; dove non ne uso neanche una, non devo includerlo. Non serve includere tutto ovunque.

- 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.

11.1 Esercizi di riscaldamento

1. Per impratichirsi con il meccanismo della separazione fra header e implementazione
   - Definire un semplice file di header `hello.h`

```
#ifndef HELLO_H
#define HELLO_H

void hello();

#endif
```

Si noti il meccanismo di protezione dalla doppia inclusione `#ifndef ... #define..#endif`

- Definire l'implementazione nel file `hello.cpp`

```
#include <iostream>
#include "hello.h"

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

Si noti l'inclusione del file di header.

- Definire un programma di test nel file `hellotest.cpp`

```
#include "hello.h"

int main()
{
  hello();
}
```

- Compilare assieme libreria e programma di test: `g++ -std=c++14 -Wall hello.cpp hellotest.cpp -o hellotest`
- Compilare separatamente libreria e programma di test e *linkarli*:
  - `g++ -std=c++14 -Wall -c hello.cpp`
  - `g++ -std=c++14 -Wall -c hellotest.cpp`
  - `g++ hello.o hellotest.o -o hellotest1`

Si veda anche l'esercizio 8 in merito a quest ultimo punto.

2. Riorganizzare la soluzione dell'esercizio 7 in sezione 10 in una libreria.

   (a) Creare un file (testo) `phone_book.h` e inseritevi le dichiarazioni del tipo `Contact_Str` e delle funzioni

```
#ifndef PHONEBOOK
#define PHONEBOOK
typedef struct Contact_Str_Impl Contact_Str;//nascondiamo implementazione ....
        //...di Contact_Str
// tutte le definizioni delle funzioni dell'esercizio
#endif
```

   (b) Creare un file `phone_book.cpp` e inseritevi le implementazioni di quanto avete dichiarato in `phone_book.h`.

```
/* Tutti gli include di librerie di sistema che vi servono*/
#include "phone_book.h"

struct Contact_Str_Impl {
        /*...tutti i campi che avevate messo in Contact_Str...*/
};
/*...le implementazioni delle funzioni...*/
```

   (c) Creare un file `phone_book_test.cpp` e inseritevi il main di test (anche in questo caso non dimenticatevi di fare `#include` di `phone_book.h`). Verificate che l'esecuzione coincida con la versione precedente alla ristrutturazione in file.

3. Realizzate la libreria delle liste semplici di stringhe, con le funzioni viste a lezione (ed alcune extra indicate)

(a) Creare un file (testo) `string_list.h` e inseritevi le dichiarazioni del tipo `cell` e delle funzioni che manipolano liste. Per nascondere l'implementazione delle liste a chi userà la libreria, usate il costrutto `typedef` per dare un nome al tipo "puntatore a cella" e la stessa tecnica usata nell'esercizio precedente per nascondere i campi del tipo `cell`.

Nel codice seguente trovate i prototipi delle prime funzioni completi e solo i nomi (ed eventualmente una breve descrizione) delle funzioni successive, che dovete completare scegliendo opportunamente parametri e tipi di ritorno.

```
#ifndef STRINGLIST
#define STRINGLIST
typedef struct cell_impl cell;//nascondiamo implementazione di cell
typedef cell* list;
void headInsert(list& l, std::string s);
void read(list& l);/* legge valori da input e li memorizza nella lista
        La funzione non deve rilasciare la memoria puntata da l, ma deve
        allocare la memoria necessaria a memorizzare gli elementi della lista
        Dopo la chiamata l==testa della lista letta*/
void print(const list l);
string getElem(const list l, int index);
// insertAt inserimento in posizione fissata
void deleteAt(list& l, int index);//cancella elemento in posizione fissata
// deleteOnce cancella la prima occorrenza di una stringa in una lista
// deleteAll cancella tutte le occorrenze di una stringa in una lista
// insertInOrder inserisce in una lista ordinata mantendola ordinata
#endif
```

(b) Creare un file `string_list.cpp` e inseritevi le implementazioni di 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 inseritevi 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 iintero pari al numero di occorrenze trovate e cancellate (0 se la stringa non è presente nella lista, quindi)

4. Realizzate la libreria delle liste semplici *ordinate* di stringhe, con le funzioni viste a lezione (ed alcune extra indicate). Per farlo appoggiatevi alla libreria delle stringhe realizzata nell'esercizio precedente

(a) Creare un file (testo) `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 prime funzioni completi e solo i nomi (ed eventualmente una breve descrizione) delle funzioni successive, che dovete completare scegliendo opportunamente parametri e tipi di ritorno.

```
#ifndef STRINGORDLIST
#define STRINGORDLIST
//riusiamo le liste per le liste ordinate
typedef struct cell_str cell;//nascondiamo implementazione di cell
typedef cell* list;
typedef list ordList;
// funzioni
void insert(ordList& l, std::string s);//inserimento ordinato
void read(list& l);/*legge elementi in ordine qualsiasi e li inserisce in ordine
```

```
Eventuale area di memoria inizialmente puntata da l NON deve essere rilasciata*/
void print(const list l);
string getElem(const list l, int index);
void deleteAt(list& l, int index);//cancella elemento in posizione fissata
// deleteOnce cancella la prima occorrenza di una stringa in una lista
// deleteAll cancella tutte le occorrenze di una stringa in una lista
/* are_equal restituisce vero se le due liste contengono le stesse stringhe
        (con la stessa molteplicit\`a) */
/* concat restituisce una lista ordinata contenente gli elementi di 2 liste date
        Il numero di occorrenze di un elemento nel risultato \`e la somma
        del numero delle sue occorrenze nei due argomenti */
/* union restituisce una lista ordinata senza ripetizioni contenente
        gli elementi presenti in almeno una delle liste argomento */
/* intersect restituisce una lista ordinata senza ripetizioni contenente
        gli elementi presenti in entrambe le liste argomento */
#endif
```

NOTA. Non abbiamo incluso `string_list.h` perché altrimenti chiunque importa `string_ord_list.h` avrebbe anche accesso alle funzioni sulle liste non ordinate e potrebbe "disordinare" una lista ordinata, ad esempio inserendo un elemento in mezzo in ordine sbagliato. Però, abbiamo dovuto copiare le dichiarazioni dei tipi indispensabili per ottenere una dichiarazione coerente del tipo delle liste ordinate. Vedrete in seguito l'es. 5 per una soluzione alternativa.

(b) Creare un file `string_ord_list.cpp` e inseritevi le implementazioni di quanto avete dichiarato in `string_ord_list.h`.

[SUGGERIMENTO: Se includete `string_list.h`, per implementare le prime funzioni potete semplicemente richiamare le funzioni corrispondenti sulle liste semplici (ad esempio `insertInOrder` per implementare `insert`).]

(c) Creare un file `string_ord_list_test.cpp` e inseritevi 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.

5. Ristrutturare i file `string_list.h` e `string_ord_list.h` estraendo da entrambi la definizione di `cell` e di `list`, mettendolo in un nuovo file header `list_basic_types.h` e includendolo nei file `string_list.h` e `string_ord_list.h`.

   Verificare che sia le liste che le liste ordinate continuino a funzionare dopo la ristrutturazione (= si riescano a compilare e l'esecuzione dei test sia invariata).

   Quali differenze vedete fra questa struttura e la precedente? quale vi sembra più appropriata?

11.2 Esercizi di base

Non fatevi scoraggiare dall'apparente lunghezza degli esercizi in questa sezione. Se avete fatto correttamente quelli nella sezione di riscaldamento, potrete riusare o adatttare molto codice e quindi il lavoro nuovo da fare sarà in realtà abbastanza limitato.

6. Approfondiamo la conoscenza sulle liste (implementate usando puntatori) definendone una variante. Le funzionalità richieste sulle liste sono molto simili a quelle che avete visto a lezione e implementato nella parte di riscaldamento, ma dovrete implementarle in maniera quanto più possibile indipendente dal tipo degli elementi della lista, così da poter riusare la nostra libreria giocattolo per vari tipi di elementi.

   Data la semplicità delle funzioni richieste sulle liste, per poterle implementare, basta che il tipo informazione contenuto in ciascuna cella a sua volta supporti tre operazioni: lettura da standard input, scrittura su standard output e confronto fra due valori per determinare se uno è minore, uguale o maggiore all'altro. Quindi incapsuleremo il tipo base in una libreria che fornisce `DataType` e le sue operazione base e la useremo per implementare le liste di `DataType` (invece che di `int` o `string`).

   Il programma finale deve essere composto dai seguenti file separati:

   - `BasicTestMain.cpp`, il programma principale che contiene le funzionalità di verifica (fornito)
   - `TestAuxFunc.h`, l'interfaccia con alcune funzioni ausiliarie per il test (fornito)
   - `TestAuxFunc.cpp`, che contiene l'implementazione delle funzionalità ausiliarie per il test (fornito in una variante, ma andrà ridefinito per ciascun tipo di dato base di cui vogliamo testare liste)
   - `BasicLists.cpp`, che contiene le funzionalità di gestione delle liste di `DataType`: `basic_list`
   - `BasicLists.h`, l'interfaccia di `basic_list` verso l'esterno (fornito)

- `BasicDataType.cpp`, che contiene le funzioni per gestire il tipo di dato `DataType` (bisognerà definirne uno diverso per ciascun tipo di dato base di cui vogliamo fare liste)
- `BasicDataType.h` parte relativa alla interfaccia delle funzionalità di `DataType` (fornito)
- `BasicDataTypeDef.h` parte relativa alla definizione del tipo `DataType`. Di questo file vi forniamo noi una versione semplice in cui `DataType` rappresenta gli interi, da usare nei primi esercizi. Dovrete poi voi definirne una variante per verificare di aver implementato le liste in maniera indipendente dal tipo base, nell'ultimo esercizio di questa sezione.

Le funzionalità richieste sono chiarite negli esercizi seguenti. Non è vietato realizzare ulteriori funzioni se lo ritenete utile.

Scaricate il file list-helper-files.zip e scompattatelo

(a) **`DataType` implementato con interi**

Create una directory `BasicDataTypeInt`, copiatevi `BasicDataTypeDef.h`, `BasicDataType.h` e createvi il file `BasicDataType.cpp`, in cui definire le funzionalità richieste in `BasicDataType.h`.

Indipendentemente dal tipo di dato specifico, `DataType` è associato a tre funzionalità con interfaccia fissata (stessi parametri):

- `ReadData`: legge un valore dallo standard input e lo memorizza in una variabile di tipo `DataType`
- `WriteData`: stampa un `DataType` sullo standard output
- `DataComparer`: confronta due variabili di tipo `DataType` first e second e ritorna un intero (0: se le variabili sono uguali, un numero negativo se first è minore di second, un numero positivo se first è maggiore di second

Siccome in questo caso `DataType` coincide con gli interi, dovete implementare:

```
void ReadData(DataType& value){
// stampare "Inserisci un numero: "
// leggere value
// stampare un a capo
}
```

```
void WriteData(const DataType& value);{
// stampare  value
}
```

```
int DataComparer(const DataType& first, const DataType& second){
// restituire first - second
}
```

Create in `BasicDataTypeInt` il file `TestBasicDataType.cpp` e scrivete un programma per testare le funzioni appena fatte:

```
// dichiarare un array A di DataType di lunghezza 10
// leggere i suoi valori usando ReadData
// stampare il contenuto
// assegnare alle celle di A (nell'ordine) i valori 7, 7, -7, -11, 78, 198,...
// ... -5, -5, 0, 0
// stampare il contenuto usando WriteData
// stampare il risultato della chiamata di DataComparer su A[i] e A[i+1] per...
// ... tutti i valori di i per cui ha senso
```

(b) **`BasicList` quando `DataType` è implementato con interi**

Create una directory `BasicListInt`, copiatevi `BasicDataTypeDef.h`, `BasicDataType.h`, `BasicLists.h`, `TestAuxFunc.h`, `TestAuxFunc.cpp`, `BasicTestMain.cpp`, il file `BasicDataType.cpp` risultato dell'esercizio precedente e createvi il file `BasicLists.cpp`, in cui definire le funzionalità richieste in `BasicLists.h`.

i. Implementate la funzione di inserimento in testa:

```
void head_insert(basic_list& list, DataType new_value)
```

Implementate la funzione di stampa di una lista:

```
void print_list(basic_list list)
```

Usare il programma di test fornito in `BasicTestMain.cpp` per testare inserimento e stampa. Notate che non possiamo testare le due funzioni indipendentemente usando un main: se non sappiamo come costruire una lista non abbiamo niente da stampare e se non sappiamo stampare la lista non sappiamo verificare se è stata costruita correttamente. In questa situazione si usano dei programmi apposta, chiamati debugger, che permettono di ispezionare il contenuto della memoria durante l'esecuzione di un programma. In questo modo si può verificare che la funzione di inserimento sia corretta senza avere la capacità di stampare le liste. A IP non impariamo a usare questi programmi (lo farete al secondo anno), ma abbiamo solo funzioni molto semplici da testare, per cui ci possiamo accontentare di testarle a coppie, se necessario.

ii. Implementare la funzione che rilascia tutto lo spazio occupato da una lista e le assegna il puntatore nullo:

```
void delete_list(basic_list& list)
```

Decommentare la riga con la chiamata a `TestDelete` nel main di test e verificate che la vostra implementazione funzioni.
Se l'esecuzione del main arriva a buon fine, potete essere sicuri che la vostra `delete_list` sia corretta, oppure c'è un aspetto della specifica di `delete_list` che non stiamo verificando?

iii. Implementare la funzione che legge una lista da `cin`:

```
void read_list(basic_list& list)
```

Commentare le righe con le chiamate a `TestHeadInsertAndPrintList` e a `TestDelete` e decommentare la riga con la chiamata a `TestRead` nel main di test e seguite le istruzioni.

iv. Implementare la funzione che rimuove un valore da una lista. Se il parametro booleano vale `true`, rimuove tutte le occorrenze di quel valore; altrimenti solo la prima incontrata.
La funzione restituisce `true` se il valore da cancellare era presente nella lista e quindi è stato effettivamente rimosso, `false` altrimenti

```
bool remove(basic_list& list, DataType to_be_removed, bool all_occurrences)
```

Commentare la riga con la chiamata a `TestRead` e decommentare la riga con la chiamata a `TestRemove` nel main di test e verificate che la vostra implementazione sia corretta.
Se tutti i test passano, che cosa potete dire della correttezza?

(c) **`DataType` implementato con caratteri**

Create una directory `BasicDataTypeChar`, copiatevi `BasicDataTypeDef.h`, `BasicDataType.h` e createvi il file `BasicDataTypeChar.cpp`, in cui definire le funzionalità richieste in `BasicDataType.h`.

Nel file `BasicDataTypeDef.h` modificate la definizione di `DataType` in modo che sia associato a caratteri invece che a interi.

Implementare in `BasicDataTypeChar.cpp` le funzionalità del tipo base per la variante caratteri (considerando come ordinamento quello naturale, dato dal codice ASCII). Create in `BasicDataTypeChar` il file `TestBasicDataType.cpp` e scrivetevi un programma (main) per testare le funzioni appena fatte:

```
// dichiarate un array A di DataType di lunghezza 10
// leggete i suoi valori usando ReadData
// stampatene il contenuto
// assegnate ad A i valori `a'  `k'  `c'  `c'  `A'  `@'  `B'  `Z'  `F'  `1'
// stampatene il contenuto usando WriteData
// stampate il risultato della chiamata di DataComparer su A[i] e A[i+1]  per...
```

```
// ... tutti i valori di i per cui ha senso
```

(d) **`BasicList` quando `DataType` è implementato con caratteri**

Create una directory `BasicListChar`, copiatevi la variante di `BasicDataTypeDef.h` che avete creato al punto precedente, `BasicDataType.h`, `BasicLists.h`, `TestAuxFunc.h`, `TestAuxFunc.cpp`, `BasicTestMain.cpp`, il file `BasicDataType.cpp` risultato dell'esercizio precedente e il file `BasicLists.cpp`.

Modificate la copia di `TestAuxFunc.cpp` in modo che la `init_data` inizializzi il vettore con 10 caratteri a vostra scelta, purché tutti distinti fra loro.

Eseguite il main di test. Se avete seguito le istruzioni, deve funzionare tutto senza ulteriori modifiche.

(e) **`DataType` implementato con una struct**

Definire una nuova implementazione di `DataType` per rappresentare le persone.

- Create una directory `BasicDataTypePeople`, copiatevi `BasicDataTypeDef.h`, `BasicDataType.h` e createvi il file `BasicDataTypePeople.cpp`, in cui definire le funzionalità richieste in `BasicDataType.h`.
- Nel file `BasicDataTypeDef.h` modificate la definizione di `DataType` in modo che sia associato ad un tipo `struct` con campi per nome, cognome e anno di nascita.
- Implementare in `BasicDataTypeChar.cpp` le funzionalità del tipo base per questa variante, considerando come ordinamento l'età, seguito dal nome. Ovvero, una persona è maggiore di un'altra se è nata prima. A parità di anno di nascita una persona è maggiore di un'altra se il suo cognome segue, in ordine alfabetico, quello dell'altra persona. A parità di anno di nascita e cognome, una persona è maggiore di un'altra se il suo nome segue, in ordine alfabetico, quello dell'altra. persona.
- Scrivere un programma di test per le funzioni di lettura, scrittura e ordinamento, seguendo lo schema di quelli fatti per interi e caratteri.

(f) **Arricchire `BasicList` quando `DataType` è implementato con `BasicDataTypePeople`**

Create una directory `BasicListPeople`, copiatevi `BasicDataType.h`, `BasicLists.h`, `BasicLists.cpp`, la variante di `BasicDataTypeDe` e il file `BasicDataTypePeople.cpp` creati al punto precedente, e createvi un file con un vostro main per il test.

i. Arricchire le liste base con l'operazione di unione di due liste.
   - Modificare `BasicLists.h` aggiungendo la dichiarazione di una funzione `union` che prende due `basic_list` e ne restituisce una nuova contenente tutti e soli gli elementi che compaiono in una delle due liste, senza ripetizioni.
   - Implementare, in `BasicLists.cpp`, la funzione `union`.
   - Scrivere i test per `union`. Suggerimento: seguite lo stesso schema del programma di test degli esercizi di riscaldamento: definitevi un vettore di elementi di tipo `DataType` e usatelo come parametro di una funzione che fa il test di `union`. Nel `main` chiamate questa funzione. In questo modo vi sarà facile modificare il main per testare le prossime funzioni negli esercizi seguenti.
     I casi da provare sono (almeno)
     – unione di due liste vuote;
     – unione di una lista vuota con una lista che contiene elementi (nei due casi possibili);
     – unione di due liste non vuote senza elementi ripetuti;
     – unione di due liste non vuote con elementi ripetuti (sia nel senso di "lo stesso elemento in entrambe le liste", sia nel senso di "lo stesso elemento più volte in una stessa lista");

ii. Arricchire le liste base con l'operazione di complemento.
   - Modificare `BasicLists.h` aggiungendo la dichiarazione di una funzione `complement` che prende due `basic_list` e ne restituisce una nuova contenente tutti e soli gli elementi che compaiono nel suo primo argomento ma non nel secondo, senza ripetizioni.
   - Implementare, in `BasicLists.cpp`, la funzione `complement`.
   - Scrivere i test per `complement`. Stesso suggerimento del punto precedente.
     I casi da provare sono (almeno)
     – complemento di due liste vuote;
     – complemento di una lista vuota con una lista che contiene elementi e viceversa;
     – complemento di due liste non vuote senza elementi in comune e senza ripetizioni nella prima;
     – complemento di due liste non vuote senza elementi in comune, con ripetizioni nella prima;
     – complemento di due liste non vuote con elementi in comune e senza ripetizioni nella prima;

(g) **Applicazione di `BasicList` quando `DataType` è implementato con `BasicDataTypePeople`**

Vediamo ora un uso delle librerie implementate finora. La caratteristica del mercato dei servizi consumer nel campo dell'ICT spinge verso una situazione di oligopolio di poche grandi company. I seguenti due servizi:

- Facelift (social networking)
- Coldmail (posta elettronica)

erano offerti da due società indipendenti. Tuttavia sono stati entrambi acquisiti dal colosso Buuble, che ora intende sfruttare le due basi di utenti per le proprie attività di marketing web.

Usando quanto realizzato finora, fornire le seguenti funzionalità di data analytics:

- Estrazione e stampa dell'elenco completo degli utenti (senza duplicazioni)
- Estrazione in elenchi separati degli iscritti solo all'uno e solo all'altro dei due servizi
- Market segmentation su base demografica: estrazione dall'elenco generale (punto 6g) di elenchi di persone nate nel periodo compreso tra il 1980, incluso, e il 1996, escluso (la cosiddetta generazione dei millennial)

Per testare le funzioni richieste, definite nel main due liste di utenti per i due servizi e popolatele usando le seguenti funzioni di lettura

```
bool ReadData(std::istream& input_stream, string& family_name, string& first_name,
                        int& birthday_year) {

        string line;
        getline(input_stream, line);
        istringstream s(line);
        if (s >> family_name && s >> first_name && s >> birthday_year) return true;
        return false;
}
```

```
bool read_list(string user_filename, basic_list& list) {
        list = nullptr;

        ifstream input_stream;
        input_stream.open(user_filename);
        if (!input_stream.is_open()) {
                cout << "non sono riuscito ad aprire il file" <<
                             user_filename<< endl;
                return false;
        }

        string fam_n;
        string name;
        int y;
        while (ReadData(input_stream, fam_n,name,y)) {
                cell* aux = new cell;
                aux->payload = create_people(fam_n,name,y);
                aux->next = list;
                list = aux;
        }
        input_stream.close();
}
```

Per poterle usare dovete definire la funzione

```
DataType create_people(string family_name, string first_name,
                       int birthday_year);
```

che prende i valori da assegnare ai campi della struct che avete definito e li usa per inizializzarne una, che restituisce come risultato.

Dovete inoltre aggiungere

```
#include <sstream>
```

Nello zip fornito trovate anche i file `ColdmailUsers.txt` e `FaceliftUsers.txt` da usare come input per `read_list` in modo da poter inizializzare le liste degli utenti dei due servizi.

(h) Arricchire le liste base con l'operazione di intersezione.

- Modificare `BasicLists.h` aggiungendo la dichiarazione di una funzione `intersection` che prende due `basic_list` e ne restituisce una nuova contenente tutti e soli gli elementi che compaiono in entrambe le liste, senza ripetizioni.
- Implementare, in `BasicLists.cpp`, la funzione `intersection`.
- Scrivere i test per `intersection`. Stesso suggerimento del punto precedente.
  I casi da provare sono (almeno)
  - intersezione di due liste vuote;
  - intersezione di una lista vuota con una lista che contiene elementi (nei due casi possibili);
  - intersezione di due liste non vuote senza elementi in comune, ciascuna senza elementi ripetuti;
  - intersezione di due liste non vuote con elementi in comune, ciascuna senza elementi ripetuti;
  - intersezione di due liste non vuote con elementi in comune, alcuni dei quali ripetuti in entrambe le liste;

(i) Arricchire le liste base con un'operazione di calcolo della lunghezza della lista.

- Modificare `BasicLists.h` aggiungendo la dichiarazione di una funzione `list_count` che prende una `basic_list` e ne restituisce il numero di elementi.
- Implementare, in `BasicLists.cpp`, la funzione `list_count`.
- Scrivere i test per `list_count`. Stesso suggerimento del punto precedente.
  I casi da provare sono (almeno)
  - `list_count` della lista vuota;
  - `list_count` della lista che contiene un solo elemento;
  - `list_count` della lista che contiene vari elementi (con o senza ripetizioni);

(j) I test fatti negli esercizi delle sezioni precedenti non possono catturare alcuni problemi di cattiva gestione della memoria:

- scrittura in aree di memoria non allocate (che possono causare errori impredicibili, ma possono anche passare inosservati in molte esecuzioni)
- mancato rilascio di memoria allocata e non più accessibile.

Per poter verificare di non aver fatto errori nelle funzioni che cancellano un valore o un'intera lista, ad esempio, abbiamo bisogno di uno strumento apposta. Provate a usare `valgrind` e verificate se avete correttamente gestito la memoria.

`valgrind` è un programma open-source di utilizzo abbastanza intuitivo, già installato sulle vostre macchine virtuali. Per imparare ad usarlo cominciate da: `http://valgrind.org/docs/manual/quick-start.html#quick-start.intro`.

(k) Implementate la funzione di rimozione di elementi da una lista (es. 6(b)iv) seguendo uno schema ricorsivo invece che iterativo.

7. **Microblogging**

Il termine microblogging si riferisce ai servizi in rete che permettono di condividere pubblicamente brevi comunicazioni, come una frase, un link, un singolo contenuto multimediale. Il piú noto di tali servizi è Twitter, ma anche altri servizi popolari (tra cui Facebook) offrono funzioni di microblogging sotto la forma di "stati" associati a un profilo utente.

Un micropost è una riga di testo, nel caso di Twitter lunga al più 140 caratteri (un "tweet"), almeno tradizionalmente.

Gli argomenti trattati vengono etichettati inserendo nel testo #tag (hash tag = stringhe alfanumeriche senza spazi che iniziano con un hash mark #).

Gli utenti a cui specificamente si riferisce un tweet vengono menzionati con mention (at-mention = stringhe alfanumeriche senza spazi che iniziano con un at sign @)

Sia hash tags che at-mention sono case insensitive, ovvero due hash tag (o due at-mention) che differiscono solo per maiuscole/minuscole sono da considerare uguali.

- Scaricare il file esertweet.zip contente i file tweet.h, tweet.cpp e tweettest.cpp
- Completare il sorgente tweet.cpp fornito per ottenere un programma che:
  - Definisce un tipo Tweet (già dato)
  - Legge un file di testo (la lettura è già implementata in tweettest.cpp), interpretandone ogni riga come un micropost (un "tweet") e memorizzandola in un apposito `std::vector<Tweet>` (chiamiamolo tweetList). Ogni Tweet contiene il testo del tweet stesso, uno `std::vector<std::string>` di tutti gli hash tag contenuti nel testo, uno `std::vector<std::string>` di tutti gli at-mention contenuti nel testo.

  - Crea due liste (implementate come `std::vector<std::string>`) contenenti l'una tutti gli hash tag presenti in tweetList, l'altra tutti gli at-mention presenti in tweetList, memorizzati senza ripetizione.
  - Crea una lista (implementata come `std::vector<int>`) che nel suo elemento i-esimo contiene il conteggio di quanti tweet in `tweetList` contengono il tag i-esimo; stampa questa informazione.
  - Seleziona lo hash tag piú usato e crea uno `std::vector<Tweet>` composto di tutti i tweet che lo contengono.
- Completare secondo le specifiche seguenti (di cui troverete i prototipi nei file forniti):
  - `Tweet stringToTweet(string s)`: Riceve una stringa (un tweet in forma testuale) e ne scrive il contenuto in un Tweet (variabile strutturata) che poi restituisce
  - `vector<string> uniqueAtMentions(vector<Tweet> & t)`: Riceve un vettore `t` di `Tweet`, crea e restituisce un vettore di stringhe `s` contenente tutte le at-mention contenute in tutti i tweet contenuti in `t`. Il vettore `s` non contiene duplicati e il contenuto è case-insensitive, cioé @pippo è considerato un duplicato di @Pippo.
  - `vector<string> uniqueHashTags(vector<Tweet> & t)`: Riceve un vettore`t` di `Tweet`, crea e restituisce un vettore di stringhe `s` contenente tutti gli hash tag contenuti in tutti i tweet contenuti in `t`. Il vettore `s` non contiene duplicati e il contenuto è case-insensitive, cioé #pippo è considerato un duplicato di #Pippo.
  - `bool hasHashTag(Tweet t, string h)`: Restituisce true se il tweet `t` contiene tra i suoi hash tag la stringa `h`, false altrimenti.
  - `vector<int> countHashTagUsage(vector<Tweet> & tw, vector<string> & ht)`: Riceve un vettore `tw` di `Tweet`, e un vettore di stringhe `ht`. Crea e restituisce un vettore di interi `c` che ha tanti elementi quanti `ht`; l'elemento numero `i` è il numero di volte che la stringa numero `i` compare come hash tag nei tweet di `ht`.

Le seguenti funzioni ausiliarie non sono richieste, ma facilitano la scrittura e la lettura del resto del codice:

- `void addUnique(vector<string> & t, string s)`: Aggiunge la stringa `s` al vector di stringhe `t` solo se `s` non è giá presente in`t`
- `string normalize(string s)`: Restituisce la stringa `s` trasformata in modo case-insensitive (tutto maiuscolo, tutto minuscolo, ...)

NOTA. Per quanto riguarda la realizzazione

- Sono forniti un programma di prova tweettest.cpp, un file header `tweet.h`, un file di implementazione `tweet.cpp`. Il file `tweet.cpp` è l'unico da completare.
- Ogni funzione segnaposto contiene del codice minimale che ha come unico scopo rendere il programma compilabile. Cancellate tale codice e scrivete quello appropriato.
- Oltre ai sorgenti è fornito anche un file di tweet reali per fare i test del programma, `test.txt`, e un file, `test.out`, che riporta le uscite corrette del programma quando viene dato in input tale file di prova.
- Hash mark e at-mention dovrebbero essere insensibili alle maiuscole (case insensitive). Si puó includere lo header `<cctype>` e usare la funzione `tolower` (o anche `toupper`) lí definita.
- Notate che un hash tag e un at-mention terminano al primo carattere non alfanumerico, non al primo spazio. In `<cctype>` è definita anche la funzione `isalnum`.
- Eventuali limiti di lunghezza (140 caratteri o altro) non vanno considerati.
- Il programma di prova fornito deve essere invocato con lo stile dei comandi Unix, cioé specificando un argomento sulla linea di comando che rapresenta il nome del file di input. Per esempio: `tweettest tweets.txt`.

11.3 Esercizi più avanzati

8. Una delle ragioni per separare il codice in file, oltre a semplificare riuso e maintenance, è permettere la *compilazione separata*. L'idea è che una libreria, una volta completata, non dovrebbe essere compilata ogni volta che si usa, ma si dovrebbe usare sempre lo stesso file eseguibile generato dall'ultima compilazione e collegarlo (*linking*) ai programmi che lo usano per fornire l'implementazione dei simboli importati mediante gli `#include`.

   In questo esercizio dovete usare la compilazione separata per le librerie delle liste semplici (es. 3) e delle liste ordinate (es. 4).

   (a) Compilate separatamente il file `string_list.cpp`.
   [SUGGERIMENTO: L'opzione per compilare senza fare linking è `-c` e sul file `pippo.cpp` produce l'object file `pippo.o`, per cui è inutile usare l'opzione `-o` per cambiare il nome del file, che è comunque già un nome significativo.]

(b) Compilate separatamente il file `string_ord_list.cpp`.

(c) Compilate i file `string_ord_list_test.cpp`, `string_ord_list.o` e `string_list.o` (fate attenzione alle estensioni).
Provate ad eseguire il programma di test ottenuto.

(d) Inserite una stampa qualsiasi in ciascuno dei tre file sorgenti per potervi rendere conto delle modifiche.
Compilate i file `string_ord_list_test.cpp`, `string_ord_list.o` e `string_list.o` (fate attenzione alle estensioni).
Provate ad eseguire il programma di test ottenuto. Quali delle nuove stampe vengono visualizzate? (e vi è chiaro perché?)
Compilate i file `string_ord_list_test.cpp`, `string_ord_list.cpp` e `string_list.cpp` (fate attenzione alle estensioni, questa non è una compilazione separata).
Provate ad eseguire il programma di test ottenuto. Quali delle nuove stampe vengono visualizzate (e perché?)

9. Implementate le stesse funzioni richieste nell'esercizio 6f per un tipo `double_linked_list` in cui ciascuna cella della lista ha, oltre ai campi `payload` 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).

   [SUGGERIMENTO: Utilizzate quanto visto a lezione nel caso delle liste doppio linkate, applicando le tecniche di astrazione dal tipo base della lista, secondo quanto fatto nell'esercizio 6.]

Progetti di riepilogo e argomenti avanzati

Parte 12

Esercizio: Pac-Man

Per vedere un uso significativo degli array, impariamo a programmare una versione semplificata del celebre gioco *Pac-man* (https://it.wikip
Man).
Per farlo avrete, bisogno di definire e usare varie funzioni, per la cui progettazione (quali algoritmi usare, come definire le funzioni, ...) avrete maggiore libertà di decisione.
Vi vengono messi a disposizione dei file con alcune parti già pre-definite. Per ciascuno di essi dovete anzitutto leggerlo e capirlo. Solo a questo punto ha senso mettersi a modificarlo.

Il gioco base: solo movimento

I percorsi del labirinto sono cosparsi di puntini, che *Pac-man* "mangia" passandoci sopra, e delimitati da pareti. Ogni posizione all'interno del labirinto contiene dunque uno fra tre possibili elementi:

1. puntino (Pac-Man non è ancora passato di qui)

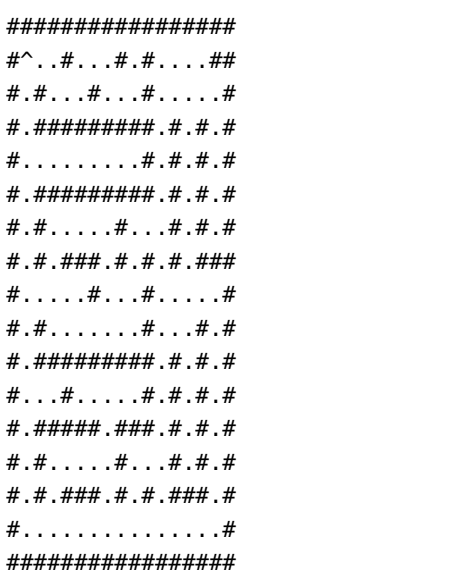
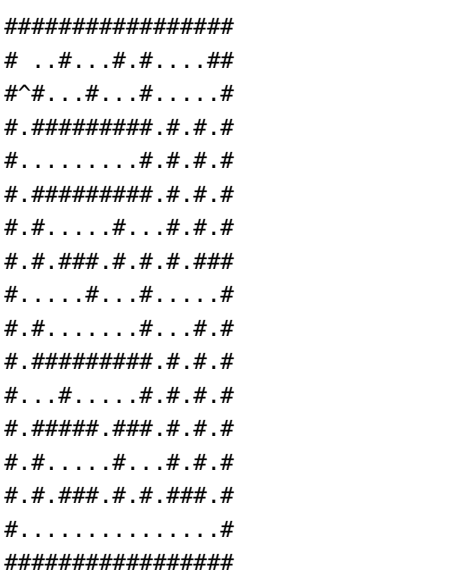
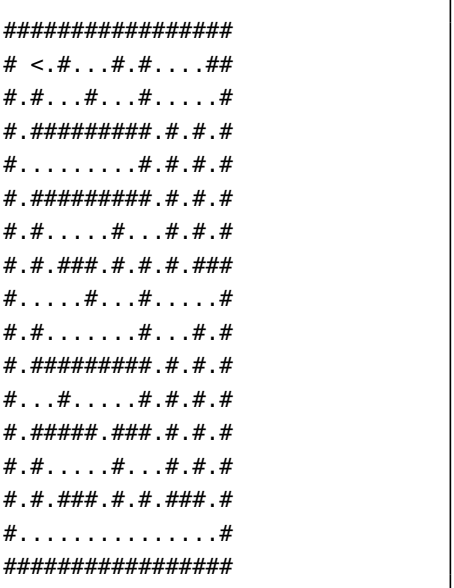
2. spazio vuoto (Pac-Man è passato di qui)

3. muro (Pac-Man non può passare qui), rappresentato usando il carattere `#'.

Il labirinto naturalmente è circondato da un perimetro di muri; questo va visualizzato al giocatore, ma è inutile memorizzarlo.
Il personaggio viene visualizzato con un carattere che dipende dall'orientamento, cioè dalla direzione dell'ultimo comando di spostamento:

```
North     v
East      <
South     ^
West      >
```

(la logica è di orientare la "bocca" del simbolo nella direzione del movimento). N.B.: il simbolo corrispondente a S è l'accento circonflesso.
L'utente ha a disposizione quattro tasti direzionali per far muovere il personaggio, scelti come se fossero tasti freccia da utilizzare con la mano sinistra: W = nord (su), A = ovest (sinistra), S = sud (giu), D = est (destra).
Ad esempio considerate la sequenza di gioco:

Inizio	Scegliendo giù	Scegliendo destra
		
All'inizio del gioco *Pac-man* si trova nell'angolo in alto a sinistra, rivolto verso il basso e tutte le posizioni accessibili contengono ancora il puntino, perché *Pac-man* non ne ha ancora mangiato nessuno.	Se nella situazione iniziale l'utente inserisce 'S' si passa a questa situazione, in cui *Pac-man* è sceso di una posizione e al suo posto è rimasta una casella vuota.	Mentre se nella situazione iniziale l'utente inserisce 'D' si passa a questa situazione, in cui *Pac-man* si è spostato verso destra di una posizione e al suo posto è rimasta una casella vuota. Si noti che avendo cambiata direzione è cambiata anche la rappresentazione.

Se nella situazione iniziale l'utente inserisce `` `W' `` o `` `A' `` bisogna segnalare che la mossa non è possibile (andrebbe contro il muro) e qualsiasi altro carattere deve essere rifiutato come comando sconosciuto e seguito dalla richiesta di inserimento di un comando corretto.

Che cosa dovete fare

Scaricate il file `PacMan.zip` nella directory in cui intendete lavorare e decomprimetelo. Otterrete una (sotto)directory `PacMan` che contiene

`PacMan_Config.h` con i dati necessari per configurare il gioco. In questa versione base, sono la dimensione del labirinto e il nome del file che contiene il labirinto da usare.

`LabirinthElem.h` con il tipo usato per rappresentare internamente al programma gli elementi del labirinto, comprensivo di quali caratteri si vogliono visualizzare gli elementi del labirinto e delle funzioni per tradurre la rappresentazione interna in caratteri e viceversa.

`LabirinthElem.cpp` con le definizione delle funzioni dichiarate in `LabirinthElem.h`; potete guardare il contenuto di questo file (soprattutto se cercate ispirazione per l'esercizio 6), ma non è indispensabile ai fini di risolvere gli esercizi.

`PacMan_Cmd.h` con il tipo usato per rappresentare internamente al programma i comandi di movimento dati dall'utente, una funzione per tradurre un carattere in un comando e una per chiedere all'utente quale comando vuole eseguire.

Notate che nel tipo elencazione che rappresenta i comandi, oltre a elementi per rappresentare i comandi che effettivamente l'utente può inviare (ovvero l'ordine di muoversi in una delle quattro direzioni o di abbandonare il gioco), è anche presente il comando `Unknown` per rappresentare un input che non corrisponde ad alcun comando noto. Questo elemento rende più usabile il tipo comandi per interagire con l'utente.

`PacMan_Cmd.cpp` con la definizione della funzione `char2command` dichiarata in `PacMan_Cmd.h`.

`PacMan_Type.h` con il tipo usato per rappresentare internamente al programma *Pac-man*, ovvero la sua posizione e orientamento, e una funzione per trasformare una direzione nel carattere usato per rappresentare *Pac-man* quando ha quell'orientamento.

`PacMan_Type.cpp` con le definizione delle funzioni dichiarate in `PacMan_Type.h`; potete guardare il contenuto di questo file, ma non è indispensabile ai fini di risolvere gli esercizi.

PacMan_BasicGame.h con le funzioni da implementare per il gioco base, ovvero per inizializzazione e spostamento nel labirinto.

PacMan.cpp con il main predefinito che usa tutte le funzioni per implementare il gioco e l'implementazione della funzione `init_maze`, che legge da un file di configurazione i dati relativi ad un labirinto e li usa per inizializzare la rappresentazione interna del labirinto (l'array).

maze.cfg con un esempio di labirinto da usare per testare il gioco.

Nei seguenti esercizi dovrete implementare le funzioni definite in queste header e scrivere dei programmi di test per provarle in isolamento.
Ricordatevi che se avete le definizione delle funzioni suddivise in più file, al momento di compilare il vostro programma dovete elencarli tutti nella linea di comando.

Attenzione: Tutti gli esercizi, nell'ordine in cui sono proposti, sono essenziali per poter completare il gioco, e molto spesso un esercizio si basa su quelli precedenti. Quindi procedete con ordine e se un esercizio non vi viene *chiedete aiuto*; non passate oltre sperando in una illuminazione futura.

12.1 Esercizi di riscaldamento

1. Completare il file `PacMan_Cmd.cpp` con una funzione che chiede all'utente di inserire un comando finché l'utente non ne inserisce uno noto e lo restituisce.

```
Command get_command(){
// dichiarare una variabile my_char di tipo char
// dichiarare una variabile my_cmd di tipo Command
/* ripetere
        - stampare la stringa "Inserisci un comando (W/w/S/s/A/a/D/d/Q/q): "
        - leggere my_char
        - assegnare a my_cmd il risultato della chiamata di char2command su my_char
   finchè my_cmd diverso da Unknown
*/
// restituire my_cmd
}
```

   Scrivere un programma per testare questa funzione:

```
// dichiarare una variabile my_char di tipo char
// dichiarare una variabile my_cmd di tipo Command
/* ripetere
        - assegnare a my_cmd il risultato della chiamata di get_command
        - stampare  un a capo seguito dalla stringa "hai chiesto di "
        - a seconda del valore di my_cmd
                -- caso Go_N: stampare la stringa "andare a nord"
                -- caso Go_S: stampare la stringa "andare a sud"
                -- caso Go_E: stampare la stringa "andare a est"
                -- caso Go_W: stampare la stringa "andare a ovest"
                -- caso Quit: stampare la stringa "uscire dal programma"
                -- caso Unknown: stampare le stringhe "..non ho capito che...
                ... cosa vuoi" e "Non dovrebbe succedere mai!!!"
        - stampare la stringa "Vuoi continuare? (y/n): "
        - leggere my_char
   finché my_char coincide con y
*/
```

   Provate a inserire come input del programma non solo i caratteri corrispondenti ai comandi (sia in maiuscolo che in minuscolo), ma anche alcuni caratteri a caso. Siete riusciti ad usare il caso `Unknown` del programma di test? Se no, per quale ragione?

2. Creare un file `PacMan_BasicGame.cpp` in cui andrete a implementare le funzioni richieste dai prossimi esercizi, dichiarate in `PacMan_BasicGame.h` e inserirvi gli `#include` che ritenete necessari (se ve ne dimenticate potrete aggiungerli man mano che fate gli esercizi e vi rendete conto che sono necessari).

3. Scrivere una funzione che prende come argomenti un `Pacman pac_man` per riferimento, due interi `x` e `y`, e un `PacmanDir d` e assegna ai campi di `pac_man` gli altri parametri

```
void set_pacman(PacMan& pac_man, int x, int y, PacmanDir d){
// assegnare x a pac_man.X
// assegnare y a pac_man.Y
// assegnare d a pac_man.direction
}
```

   Questa funzione è così banale che possiamo anche non testarla.

4. Scrivere una funzione che inizializza il gioco, ovvero il labirinto e *Pac-man*, a partire dal nome del file che contiene il labirinto.

```
void init(std::string config_file_name, Labirinth_Elems M[SIZE][SIZE],
             PacMan& pac_man){
// chiamare init_maze su config_file_name e M
// chiamare set_pacman su pac_man, 0, 0, South
}
```

   Anche questa funzione è così banale che possiamo non testarla.

5. Scrivere una funzione che visualizza lo stato del gioco, dati il labirinto e lo stato corrente di *Pac-man*.

```
void display(Labirinth_Elems m[SIZE][SIZE], PacMan pac_man){
// stampare una riga di WALL_C
/* ripetere per ciascuna riga
        - stampare un WALL_C
        *//* ripetere per ciascuna colonna
                - se la posizione coincide con quella di pac_man
                        -- stampare il risultato della chiamata di...
                        ... PacmanDir2char su pac_man.direction
                - altrimenti
                        -- stampare risultato di chiamata di lab_elem2char su...
                        ... elemento del labirinto
        *//*
        - stampare un WALL_C e una andata a capo
*/
// stampare una riga di WALL_C
}
```

   [Hint: per stampare una riga di `WALL_C` potete riusare la funzione `replicate` implementata in una delle parti precedenti dell'eserciziario.]

   Per testare la funzione, scrivere un programma che usa `init` per inizializzare il gioco e chiama la `display`.
   Provate poi a usare la `set_pacman` per spostare *Pac-man* in varie posizioni (e con vari orientamenti) e chiamare nuovamente la `display` dopo ogni spostamento.
   Verificare ogni volta che il risultato sia quello atteso.

6. Scrivere una funzione che dato un comando di movimento restituisce la direzione in cui si deve muovere *Pac-man*.

```
PacmanDir cmd2dir(Command c){
// se c coincide con Go_S
//      - restituisce South
// se c coincide con Go_N
//      - restituisce North
// se c coincide con Go_E
//      - restituisce East
// se c coincide con Go_W
//      - restituisce West
// in tutti gli altri casi
```

```
//      - solleva una eccezione (decidete voi se differenziare i possibili...
//      ... errori con tipi di eccezione diversi)
}
```

7. Scrivere una funzione che dato lo stato del gioco, ovvero *Pac-man* e il labirinto, e un comando di movimento restituisce `true` se la mossa è ammissibile, cioè se *Pac-man* si può muovere in quella direzione senza picchiare nel muro, `false` altrimenti. Se il risultato è `true`, aggiorna anche lo stato di *Pac-man*, che è passato per riferimento.

```
bool make_move(PacMan& pac_man, Command c, Labirinth_Elems M[SIZE][SIZE]){
// dichiarare una variabile intera x inizializzata con pac_man.X
// dichiarare una variabile intera y inizializzata con pac_man.Y
// se c coincide con Quit o con Unknown
//      - sollevare eccezione
// se c coincide con Go_S
//      - incrementare y
// se c coincide con Go_N
//      - decrementare y
// se c coincide con Go_E
//      - incrementare x
// se c coincide con Go_W
//      - decrementare x
// se x o y fuori dal range [0,SIZE)
//      - restituire false
// se posizione [x,y] occupata da un muro
//      - restituire false
// aggiornare pac_man con x, y e il risultato della chiamata di cmd2dir su...
// ... c usando la funzione set_pacman
// restituire true;
}
```

Per testare questa funzione potete usare il main fornito e giocare a muovere *Pac-man* nel labirinto.

8. Modificare il gioco in maniera che *Pac-man* venga inizialmente posizionato in un punto a caso del labirinto, purché ammissibile. [HINT: *Ammissibile* vuol dire che non è un muro e non è circondato da muri su tutti e quattro i lati (se no *Pac-man* non potrebbe muoversi).]
Per fare questo basta definire una funzione che genera in maniera pseudo-casuale coordinate ammissibili e modificare la funzione `init` in modo da usarla per creare gli argomenti della chiamata di `set_pacman`, ossia:

   (a) Definire una funzione che verifica se una coppia di coordinate, cioè di interi compresi fra 0 e SIZE-1, è ammissibile

```
bool acceptable(int x, int y, Labirinth_Elems M[SIZE][SIZE]){
// se posizione [x,y] occupata da un muro restituire false
// se x maggiore di zero e posizione [x-1,y] NON occupata da muri
//      - restituire true
// se y maggiore di zero e posizione [x,y-1] NON occupata da muri
//      - restituire true
// se x minore di SIZE-1 e posizione [x+1,y] NON occupata da muri
//      - restituire true
// se y minore di SIZE-1 e posizione [x,y+1] NON occupata da muri
//      - restituire true
// restituire false
}
```

   Per testare questa funzione definite un main che usa `init_maze` per inizializzare il labirinto e fate chiamate di `acceptable` con coordinate sui quattro bordi e nel centro, sia per coordinate accettabili, sia per coordinate non accettabili.

   (b) Definire una funzione che genera in maniera pseudo-casuale coordinate ammissibili.
   Per generare numeri casuali si usa la funzione `rand()` in `<cstdlib>`, che a ogni invocazione restituisce un numero intero non negativo preso da una sequenza di numeri generati casualmente a partire da un valore iniziale, il `seed` della sequenza.

```
void rand_coord(int& x, int& y, Labirinth_Elems M[SIZE][SIZE]){
/* ripetere
        - assegnare a x il resto della divisione intera fra il risultato...
        ... della chiamata di rand e SIZE
        - assegnare a y il resto della divisione intera fra il risultato...
        ... della chiamata di rand e SIZE
   finchè la chiamata di acceptable su x e y non diventa vera
*/
}
```

(c) Modificare la funzione `init` in maniera che si usi `rand_coord` per posizionare *Pac-man*.
Siccome per un `seed` fissato si ottiene sempre la stessa sequenza, per ottenere sequenze diverse in diverse partite, bisogna inizializzare *una sola volta prima di cominciare a produrre numeri casuali* la sequenza con una chiamata a `srand(...)` (sempre in `<cstdlib>`).
I puntini devono essere sostituiti da un'espressione di inizializzazione che idealmente dovrebbe essere diversa ad ogni esecuzione, ad esempio perché dipende dall'istante in cui avviene l'esecuzione.
Dalla documentazione del C++:
Standard practice is to use the result of a call to time(0) as the seed. However, time() returns a time_t value, and time_t is not guaranteed to be an integral type. In practice, though, every major implementation defines time_t to be an integral type, and this is also what POSIX requires.

```
void init(std::string config_file_name, Labirinth_Elems M[SIZE][SIZE],
             PacMan& pac_man){
// chiamare init_maze su config_file_name e m
// inizializzare il seed con la chiamata di srand su time(0)
// dichiarare variabili intere x e y
// inizializzare x e y usando la funzione rand_coord
// chiamare set_pacman su pac_man, x, y, South
}
```

Per testare la modifica basta usare il main fornito.

12.2 Esercizi di base

In questa parte dovete aggiungere frutti che se mangiati aumentano il punteggio di gioco. I frutti compaiono in istanti e in posizioni casuali nel labirinto, e restano visualizzati per un certo numero di passi, fissato a priori in modo che il giocatore sappia se è possibile raggiungerli prima che spariscano.

9. Introdurre nel gioco il *punteggio*. Per farlo bisognerà modificare
   - `PacMan_Config.h`, aggiungendo la costante per rappresentare quanti punti si guadagnano mangiando un puntino, da inizializzare a 10;
   - `init` e la sua chiamata nel `main` in maniera che abbia un ulteriore parametro intero per riferimento, il punteggio, da inizializzare a zero;
   - `make_move` e la sua chiamata nel `main` in maniera che abbia un ulteriore parametro intero per riferimento, il punteggio, e lo incrementi di 10 (ricordatevi la costante) se muovendo *Pac-man* si sposta in una cella del labirinto che conteneva un puntino (nel qual caso bisogna anche modificare il labirinto sostituendo il puntino ormai mangiato con uno spazio);
   - `display` in modo che visualizzi anche il punteggio corrente
10. Aggiungere a `PacMan_Config.h` le costanti necessarie per la gestione dei frutti, ovvero il punteggio totalizzato quando si riesce a mangiare un frutto (300), quanti turni di gioco dura la visualizzazione di ciascun frutto (7), il numero massimo di frutti contemporaneamente visibili (5) e il carattere usato per visualizzare un frutto (`` `o' ``).
11. Aggiungere un tipo per rappresentare i frutti, analogamente a quanto fatto per rappresentare *Pac-man* in `PacMan_Type.h` e `PacMan_Type.cpp`.

    Potete usare una `struct` per rappresentare lo stato del frutto, ovvero la sua posizione nel labirinto, se è visibile, e se lo è per quanti turni di gioco lo resterà.

12. Modificare `init` e la sua chiamata nel `main` aggiungendo un nuovo parametro di tipo array di frutti da inizializzare in maniera casuale.

    Per la gestione della casualità usate la stessa tecnica introdotta nell'ultimo esercizio della sezione precedente.

    Si noti che ci sono due elementi di casualità: se un frutto è effettivamente visualizzato (abbiamo fissato solo il massimo ammissibile) e la posizione di ciascuno di quelli visualizzati.

13. Modificare `make_move` in maniera che se *Pac-man* raggiunge un frutto (visibile)

    - guadagna i punti corrispondenti
    - il frutto diventa invisibile
    - al posto del frutto nel labirinto compare uno spazio

    Bisogna inoltre gestire il ciclo di vita dei frutti:

    - i frutti che avevano zero turni di visibilità residua a inizio mossa devono diventare invisibili e al loro posto nel labirinto deve essere messo uno spazio;
    - il numero di turni di visibilità residua dei frutti visibili va decrementato;
    - i frutti che erano invisibili a inizio mossa possono tornare visibili con probabilità del 50% in una nuova posizione casuale (non in conflitto con muri e altri elementi del gioco);

14. Modificare `display` in maniera che visualizzi anche i frutti

12.3 Esercizi più avanzati

15. Aggiungere quattro fantasmi (da rappresentare con il carattere `` `M' ``) che, a ogni turno di gioco, fanno un passo in una direzione casuale. Se la posizione del *Pac-man* viene a coincidere con quella di un fantasma, il gioco termina con la vittoria del calcolatore.

    Lo svolgimento di questo esercizio è molto simile al caso dei frutti, dal punto di vista delle funzioni da introdurre/modificare, dei tipi e delle costanti necessari. L'unica differenza sostanziale è che mentre i frutti restano fermi (al più appaiono/spariscono), i fantasmi si muovono subito dopo il giocatore in maniera pseudo-casuale. Per evitare troppa casualità nel movimento dei fantasmi, che li porterebbe a tornare troppo spesso sui loro passi, potete modificarne la direzione in modo casuale non ad ogni mossa, ma *con una certa probabilità*. Ovvero, potete

    - fissare la probabilità `P` con cui volete che i fantasmi cambino direzione (ad esempio 0.25 se volete che cambino una volta su 4),
    - generare un numero `p` casuale in $[0, 1]$
    - se `p` è minore di `P` cambiare casualmente la direzione del fantasma, altrimenti lasciarlo proseguire nella direzione che aveva finora (se non c'è un muro ovviamente).

    HINT: Per generare in modo casuale un numero `p` in $[0, 1]$ basta dividere il risultato di una chiamata a `rand()` per il massimo valore che può assumere il suo risultato, ovvero `RAND_MAX`.

    Attenzione ad usare la divisione fra `float` e non la divisione intera!

16. Modificare le regole del gioco in maniera che dopo aver mangiato un frutto il giocatore possa uccidere i fantasmi fino al quarto turno successivo. Per ogni fantasma ucciso *Pac-man* prende 1000 punti.

17. Scrivere una funzione che inizializza un labirinto creando un labirinto casuale. I labirinti devono avere una struttura ragionevole: essere articolati (non blob unico), seguire varie direzioni, alternare muri a spazi, utilizzare tutto lo spazio a disposizione.

    Un modo semplice è generare una tabella tutta di muri, e far muovere casualmente un *tarlo* che converte i muri che incontra in puntini. Per il movimento del *tarlo* si veda il suggerimento relativo al movimento dei fantasmi dato al punto 15 di questa sezione.

Parte 13

Esercizio: Matrici dense e sparse

In questa sezione realizzeremo una **libreria** di calcolo numerico che lavora con **matrici dense** e **sparse**. Ricordate di organizzare la vostra libreria in modo da tenere separati l'interfaccia delle funzioni (che esponete verso l'esterno) e la loro implementazione (che invece terrete nascosta).
Create programmi intermedi per verificare le funzioni che via via implementerete, e gestite eventuali errori con la definizione di opportune eccezioni.

13.1 Alcuni concetti sulle matrici

Una matrice può essere descritta come una tabella bidimensionale di valori, che noi supponiamo essere di tipo `double`, organizzati in righe e colonne. Ogni valore viene quindi individuato da una coppia di indici o coordinate di tipo `int`.
In molte applicazioni di analisi dati è frequente incontrare matrici **sparse**, ovvero contenenti un gran numero di valori nulli. Quando una matrice è grande, memorizzare solo i valori non-zero può portare a significativi guadagni di spazio e di velocità.
Una matrice non sparsa, ovvero **densa** è tipicamente realizzata come una tabella di valori, occupando tutte le possibili posizioni di un'area di memoria grande

```
nr x nc x sizeof(double)
```

bytes, dove `nr` e `nc` sono, rispettivamente, il numero di righe e di colonne della matrice. La posizione di un elemento in memoria corrisponde alle sue coordinate di riga e colonna, così che è necessario memorizzare solo gli elementi stessi.

$$\begin{pmatrix} 1.212 & -0.321 \\ 81.94 & -7.465 \\ 3.1415 & 10.76 \end{pmatrix}$$

1.212	-0.321	81.94	-7.465	3.1415	10.76
(1,1)	(1,2)	(2,1)	(2,2)	(3,1)	(3,2)

Una matrice sparsa è invece tipicamente realizzata come un elenco di triple
`(r,c,val)` dove `r` è un indice di riga, `c` un indice di colonna, `val` il valore dell?elemento in posizione `[r][c]` . Solo le triple corrispondenti ad elementi non zero sono memorizzate. Poiché la posizione in memoria non corrisponde alle coordinate, occorre esplicitamente memorizzare anche queste:

$$\begin{pmatrix} 1.212 & 0 \\ 81.94 & 0 \\ 0 & 10.76 \end{pmatrix}$$

(1,1,1.212)	(2,1,81.94)	(3,2,10.76)

NOTA: È conveniente usare questa rappresentazione quando il numero di elementi non-zero memorizzati `nnz` è abbastanza basso da compensare la maggiore occupazione di memoria di ciascun elemento, ovvero, per una matrice di dimensione `nr x nc`, quando si ha

```
nnz x dimensione di una tupla = nnz x (2 x sizeof(int) + sizeof(double)) < nr x nc x sizeof(double)
```

13.2 Esercizi di riscaldamento

1. Creare una `struct Matrix` per rappresentare matrici dense. La `struct` contiene un `vector` di `vector` di `double`, il numero di righe ed il numero di colonne:

```
struct Matrix {
        std::vector<std::vector<double> > store;
        int nc; // numero di righe
        int nr; // numero di colonne
}
```

2. Creare le strutture dati per rappresentare una matrice sparsa come lista linkata. Un elemento della matrice sparsa è rappresentato dalla `struct SparseEntry`, che contiene le coordinate di riga e colonna dell'elemento all'interno della matrice, il valore, ed un puntatore allo stesso tipo. La matrice sparsa è invece rappresentata dalla `struct SparseMatrix`, che contiene una lista di `SparseEntry`, il numero di righe e di colore, ed il numero di elementi non nulli.

```
struct SparseEntry {
        int r;
        int c;
        double val;
        SparseEntry *next:
}

struct SparseMatrix {
        SparseEntry *store;
        int nr;
        int nc;
        int nnz;
}
```

3. Realizzare una funzione `iszero` che verifica se un numero è zero, tenendo conto del fatto che i valori maneggiati sono di tipo `double`:

```
// Nel calcolo numerico si considera zero un numero sufficientemente piccolo.
// Definiziamo quindi una tolleranza arbitraria ma ragionevole
bool iszero(double val) {
        static const int tolerance = 1e-12;
        return ... completare...
}
```

   [HINT: nella verifica dovete considerare sia il caso in cui `val` è maggiore di zero che quello in cui risulta essere minore.]

4. Scrivere una funzione che crea una matrice densa a partire da un vettore di valori, secondo il seguente pseudo-codice:

```
Matrix matrix(const std::vector<double> &v, const int nr, const int nc) {
// Definire una Matrix m
// Porre le sue dimensioni a nr e nc
// Ridimensionare il vector m.store alla dimensione nr
/* Iterare r da 0 a nr-1
        - ridimensionare l'elemento r-esimo di m.store alla dimensione nc
*/
/* Iterare r da 0 a nr-1
        - *//* Iterare c da 0 a nc-1
                -- se l'elemento numero c+r*nc esiste nel vettore v
                        assegnare il suo valore all'elemento di m alla riga r e colonna c
                   altrimenti
                        porre a zero l'elemento di m alla riga r e colonna c
        *//*
*/
restituire m
```

```
}
```

NOTA: Se la lunghezza di `v` eccede il numero di elementi della matrice, i valori in eccesso non vengono utilizzati. Se invece la lunghezza è inferiore, i valori mancanti sono posti a zero.
Gli elementi del vettore vengono copiati nella matrice in ordine di riga o di colonna?

5. Scrivere una funzione `printFullMatrix` che stampa tutti i valori contenuti in una matrice densa

```
void printFullMatrix(const Matrix &m) {
/* ripetere per r da 0 a m.nr-1
        - *//* ripetere per c da 0 a m.nc-1
                -- stampare l'elemento di m in posizione (r,c)
        *//*
        - stampare un a capo
*/
}
```

13.3 Esercizi di base

6. Scrivere una funzione `SparseMatrix sparse(const Matrix &m)` che realizza una conversione da matrice densa a sparsa

7. Scrivere una funzione `Matrix full(const SparseMatrix &s)` che realizza una conversione da matrice sparsa a densa

8. Scrivere una funzione `SparseMatrix removeEntry(const SparseMatrix &s, const int r, const int c)` che elimina l'elemento di riga `r` e colonna `c` dalla lista che memorizza la matrice, e restituisce la matrice sparsa aggiornata.

9. Scrivere una funzione `SparseMatrix setEntry(const SparseMatrix &s, const double val, const int r, const int c)` che assegna all'elemento di riga `r` e colonna `c` della matrice sparse `s` il valore `val`, e restituisce la matrice aggiornata. La funzione dovrà rispettare i seguenti requisiti:

   - Se non esiste nella matrice sparsa un elemento con le stesse coordinate, la funzione aggiunge una tripla alla lista solo se il valore assegnato non è zero
   - Se nella matrice sparsa è già presente un elemento con le coordinate indicate, la funzione deve solo aggiornare il valore se quello che si vuole assegnare è diverso da zero
   - Se nella matrice sparsa è già presente un elemento con le coordinate indicate, ed il valore che si vuole assegnare è zero, allora la funzione deve eliminare la tripla dalla lista, per rispettare la definizione di matrice sparsa.

   Ricordate di aggiornare tutte le grandezze che vengono modificate per effetto della funzione.

10. Realizzare una funzione `double getEntry(const SparseMatrix &s, const int r, const int c)` che cerca un elemento in `s` con le coordinate `(r,c)` e ne restituisce il valore.
[HINT: Se un elemento non si trova nella lista significa che non è stato assegnato, ossia che il suo valore è zero.]

11. Scrivere una funzione `void printSparseMatrix(const SparseMatrix &s)` che stampa i valori di una matrice sparsa:

```
void printSparseMatrix(const SparseMatrix &s) {
/* ripetere per r da 0 a s.nr-1
        - *//* ripetere per c da 0 a s.nc-1
                -- stampare il risultato della funzione getEntry(s,r,c)
        *//*
        - stampare un a capo
*/
}
```

13.4 Esercizi più avanzati

12. Realizzare una funzione `SparseMatrix add(const SparseMatrix &s1, const SparseMatrix &s2)` che date due matrici sparse ne calcola la somma **elemento per elemento** e restituisce la matrice così ottenuta.
13. Scrivere una funzione `SparseMatrix mult(const SparseMatrix &s1, const SparseMatrix &s2)` che date due matrici sparse ne calcola il prodotto **elemento per elemento** e restituisce la matrice così ottenuta.
14. Scrivere una funzione `SparseMatrix prod(const SparseMatrix &s1, const SparseMatrix &s2)` che date due matrici sparse ne calcola il prodotto **righe colonne** e restituisce la matrice così ottenuta.
15. Creare un'implementazione alternativa della libreria utilizzando per memorizzare gli elementi della matrice densa, al posto di `std::vector`, un array dinamico di dimensione `nr x nc` elementi (ossia una versione bidimensionale degli array dinamici che avete visto in una delle parti precedenti dell'eserciziario).

Parte 14

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.

14.1 Esercizi di riscaldamento

1. Scrivere una funzione ricorsiva `int fact(const int& n)` per il calcolo del fattoriale di un intero positivo `n`:

```
int fact(const int& n) {
// se n < 0 solleva eccezione
// se n=0 ritorna 1
// altrimenti ritorna n moltiplicato per il fattoriale di n-1
}
```

2. Scrivere una funzione ricorsiva `int coeffBin(const int& n, const int& k)` per il calcolo del coefficiente binomiale: dati $n, k \in \mathbb{N}$ tali che $0 \leq k \leq n$ il coefficiente binomiale Ã¨

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

```
int coeffBin(const int& n, const int& k) {
// se k o n sono minori di zero, oppure 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 `int fibonacci(const int& 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)
```

```
int fibonacci(const int& n) {
// se n e' minore di zero solleva eccezione
// 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 `int MCD(const int& m, const int& 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
```

14.2 Esercizi di base

In questa sezione consideriamo funzioni che operano su liste semplici di un generico tipo `T`. La scelta dello specifico tipo è a vostra discrezione.

5. Scrivere una funzione ricorsiva `bool is_in(const list& l, T x)` per implementare la ricerca di un elemento di tipo `T` in una lista semplice. [SUGGERIMENTO: Se necessario tornate all'esercizio 4 della Parte 11 per ripassare le definizioni di base.]
6. Scrivere una funzione ricorsiva `int length(const list& l)` che calcola la lunghezza di una lista.
7. Scrivere una funzione ricorsiva `bool rec_insertElemInOrder(const list& l, T x)` che dati una lista **ordinata** `l` senza ripetizioni di elementi di tipo `T` inserisce il nuovo elemento `x` nella posizione giusta. La funzione restituisce `true` se l'operazione è andata a buon fine, `false` altrimenti.
8. Scrivere una funzione ricosiva `bool rec_removeElemInOrder(const list& l, T x)` che dati una lista **ordinata** `l` senza ripetizioni di elementi di tipo `T` elimina l'elemento `x` dalla listaa. La funzione restituisce `true` se l'operazione è andata a buon fine, `false` altrimenti.

14.3 Esercizi più avanzati

9. 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.
10. Scrivere una funzione ricorsiva che realizza l'algoritmo di *Merge Sort*. Implementare due varianti della funzione, utilizzando prima vector e poi liste.
11. 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.

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

`cd` ⟨`cartella`⟩	cambia directory (cartella) di lavoro
`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`⟩

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`

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