Introduzione alla Programmazione
a.a. 2025/26

Eserciziario

20 novembre 2025

Indice

1 Espressioni, lettura e assegnazione **7**
- 1.1 Esercizi di riscaldamento . . . 7
- 1.2 Esercizi di base . . . 8
- 1.3 Esercizi più avanzati . . . 9

2 Scelte condizionali **11**
- 2.1 Esercizi di riscaldamento . . . 12
- 2.2 Esercizi di base . . . 12
- 2.3 Per chi non si accontenta . . . 13

3 Funzioni **15**
- 3.1 Esercizi di riscaldamento . . . 16
- 3.2 Esercizi di base . . . 17

4 Cicli **19**
- 4.1 Esercizi di riscaldamento . . . 20
- 4.2 Esercizi di base . . . 23
- 4.3 Esercizi più avanzati . . . 24

5 Cicli e Funzioni **25**
- 5.1 Esercizi di riscaldamento . . . 25
- 5.2 Esercizi di base . . . 27
- 5.3 Esercizi più avanzati . . . 29

6 Vector **31**
- 6.1 Esercizi di riscaldamento . . . 31
- 6.2 Esercizi di base . . . 35
- 6.3 Esercizi più avanzati . . . 37

7 Vector avanzati **39**
- 7.1 Esercizi di riscaldamento . . . 39
- 7.2 Esercizi di base . . . 40
- 7.3 Esercizi più avanzati . . . 42

8 Funzioni avanzate ed Eccezioni **43**
- 8.1 Esercizi di riscaldamento . . . 43
- 8.2 Esercizi di base . . . 48
- 8.3 Esercizi più avanzati . . . 50

9 String **53**
- 9.1 Esercizi di riscaldamento . . . 53
- 9.2 Esercizi di base . . . 54

10 Struct **57**
- 10.1 Esercizi di riscaldamento . . . 57
- 10.2 Esercizi di base . . . 59
- 10.3 Esercizi più avanzati . . . 61

11 Compilazione separata **63**
11.1 Esercizi di riscaldamento . . . 64
11.2 Esercizi di base . . . 65

A Cheatsheet per lavorare su Linux **67**
A.1 Comandi bash . . . 67
A.2 Editing dei file sorgente . . . 68
A.3 Compilazione . . . 68
A.4 Debugging . . . 69

B Riepilogo introduzione a C++ **71**
B.1 Struttura del programma . . . 71
B.2 Regole grammaticali . . . 71
B.3 Dichiarazioni . . . 72
B.4 Variabili, inizializzazioni, assegnazioni . . . 72
B.5 Scope e visibilità . . . 73
B.6 Espressioni e operatori . . . 73
B.7 Errori comuni . . . 74

C Linguaggio C(++) **75**

Come usare l'eserciziario

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

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

Organizzazione dell'eserciziario

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

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

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

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

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

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

  Ci aspettiamo che voi produciate un programma simile a

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

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

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

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

Note utili

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

Parte 1

Espressioni, lettura e assegnazione

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

Cheatsheet

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

1.1 Esercizi di riscaldamento

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

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

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

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

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

Confrontando i due algoritmi.

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

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

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

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

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

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

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

1.2 Esercizi di base

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

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

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

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

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

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

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

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

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

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

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

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

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

1.3 Esercizi più avanzati

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

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

Parte 2

Scelte condizionali

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

Cheatsheet

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

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

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

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

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

Tipi primitivi:

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

Altri tipi comuni:

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

2.1 Esercizi di riscaldamento

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

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

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

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

2.2 Esercizi di base

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

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

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

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

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

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

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

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

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

    [File `temperature.cpp`]

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

2.3 Per chi non si accontenta

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

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

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

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

Parte 3

Funzioni

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

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

Cheatsheet

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

Dichiarazione:
Esempio

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

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

Definizione:
Esempio

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

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

Invocazione o chiamata:
Esempi:

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

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

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

Errori comuni:

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

3.1 Esercizi di riscaldamento

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

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

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

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

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

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

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

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

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

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

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

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

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

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

   [File `testf_basic.cpp`]

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

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

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

   Riprovare a compilare e a eseguire.

   [File `testf_basic2.cpp`]

3.2 Esercizi di base

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

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

Scrivere un programma per testare la funzione `area`:

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

[File `testf_area.cpp`]

4. Partendo dall'esercizio svolto nella sezione precedente, scrivere un programma in linguaggio **C++** che legga da tastiera una temperatura (in gradi Celsius) e stampi un messaggio che descriva la situazione corrispondente. Il messaggio deve essere scelto in base ai seguenti intervalli:

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

   Il programma deve essere strutturato utilizzando **funzioni**, in particolare:

   (a) una funzione per leggere la temperatura da tastiera;

   (b) una funzione che, dato il valore della temperatura, restituisca il messaggio corrispondente;

   (c) una funzione per stampare il messaggio risultante.

   Nel `main()`, il programma deve semplicemente:

   - chiamare la funzione che legge la temperatura;
   - ottenere la descrizione tramite la funzione dedicata;
   - stampare il risultato.

```
int main() {
    int temperatura = leggiTemperatura();
    string messaggio = descriviTemperatura(temperatura);
    stampaRisultato(messaggio);
}
```

[File `temperatureFunctions.cpp`]

5. Partendo dall'esercizio svolto nella sezione precedente, scrivere un programma in linguaggio **C++** che legga da tastiera un numero intero compreso tra 1 e 12 e stampi il **nome del mese** corrispondente (1 = gennaio, 2 = febbraio, ..., 12 = dicembre).

   Il programma deve utilizzare un'istruzione `switch` per determinare il nome del mese e deve essere strutturato utilizzando **funzioni**.

   In particolare, devono essere implementate almeno le seguenti funzioni:

   (a) una funzione per leggere da tastiera il numero del mese;

   (b) una funzione che, dato il numero del mese, restituisca il nome corrispondente come stringa;

   (c) una funzione per stampare il risultato ottenuto.

   Nel `main()` il programma deve semplicemente:

   - leggere il numero del mese chiamando la prima funzione;
   - ottenere il nome del mese chiamando la seconda funzione;
   - stampare il risultato tramite la terza funzione.

   Se il numero inserito non è compreso tra 1 e 12, il programma deve stampare il messaggio:

   Mese NON valido!

```
int main() {
    int mese = leggiMese();              // Lettura input
    string risultato = nomeMese(mese);   // Elaborazione
    stampaRisultato(risultato);          // Output
}
```

[File `monthnameFunctions.cpp`]

Parte 4

Cicli

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

Cheatsheet

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

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

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

Ciclo `for` in generale:

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

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

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

Ciclo while

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

Ciclo do. . . while

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

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

Quale costrutto devo usare? Regola di prima approssimazione:

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

Tecnica del look-ahead

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

4.1 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 positivo (>0)
//      - 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'(<'A') o maggiore di 'Z' (>'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 (<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' (>='A') e minore o uguale a 'Z' (<='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 o uguale a 0: "
// dichiarare una variabile intera n
// leggere n
// se n è negativo (<0)
//      - stampare "Avevo detto positivo o uguale a 0!"
//      - 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 positivo: "
// dichiarare una variabile len di tipo int
// leggere len
// se len non e` maggiore di zero (>0)
//      - 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 senza zeri finali (e.g., 100 o 1230 sono vietati) 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 non è positivo (>0)
//      - stampare "Valore non valido"
//      - uscire dal programma ritornando il codice di errore 66
// stampare su una nuova riga la stringa "Rovesciando " seguita da k
// dichiarare una variabile intera inv inizializzata a zero
/* finché k è maggiore di zero
        - dichiarare una variabile intera mod inizializzata con il resto della divisione intera di k per 10
        - assegnare a k il quoziente di k per 10
        - assegnare a inv la moltiplicazione di inv per 10
        - assegnare a inv la somma di inv e mod
*/
// stampare la stringa " si ottiene " seguita da inv
```

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

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

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

4.2 Esercizi di base

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

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

using namespace std;

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

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

10. Scrivere un programma che chiede all'utente un numero intero K positivo (> 0) e ne stampa il numero di cifre (in base 10). Ad esempio su 27458 stampa 5.

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

    L'output è:

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

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

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

    [File `numdigits.cpp`]

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

12. Scrivere un programma che legge due numeri interi positivi (>0) (le 2 basi di un trapezio rettangolo: b e B con b<=B) e stampa il trapezio rettangolo fatto di `'x'` con le basi lunghe quanto i numeri letti, e l'altezza pari alla differenza fra le basi più uno.
    [File `stampatrapezio.cpp`]

    Ad esempio avendo in input 5 e 9 stamperà:

```
xxxxx
xxxxxx
xxxxxxx
xxxxxxxx
xxxxxxxxx
```

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

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

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

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

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

14. Scrivere un programma che chiede all'utente di scegliere il suo colore preferito presentandogli un menu con almeno 5 scelte di colori i cui nomi iniziano con una lettera diversa (ad esempio rosso, verde, blu, giallo e arancio). Per selezionare la scelta l'utente potrà usare l'iniziale del colore scelto, indifferentemente in maiuscolo o minuscolo. Se il carattere inserito non corrisponde a nessun colore proposto il programma ricomincerà da capo presentando il menu con le scelte. [File `menucolors.cpp`] [SUGGERIMENTO: Come base di partenza può essere usato l'esercizio 6]

15. Scrivere un programma che chiede all'utente un numero intero positivo (>0) e verifica se è *palindromo*, ovvero se le sue cifre (in base 10) lette da destra a sinistra corrispondono alle cifre lette da sinistra a destra (ad esempio 165373561 è palindromo). Dopo la verifica stampa il risultato all'utente. Per risolvere questo esercizio si può partire dalla soluzione dell'esercizio 8. [File `ispalindrome.cpp`]

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

4.3 Esercizi più avanzati

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

18. Scrivere un programma che legge un intero positivo 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 5

Cicli e Funzioni

5.1 Esercizi di riscaldamento

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

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

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

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

   [File `testf_average.cpp`]

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

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

```
char last_letter(){
// stampare la stringa "Inserisci una lettera minuscola "
// dichiarare una variabile last di tipo char
/* ripetere
        - leggere last
   fintanto che last minore di 'a' o maggiore di 'z'
*/
// dichiarare una variabile c di tipo char inizializzata con 'a'
/* ripetere
        - se c è maggiore di last
```

```
                -- assegnare il valore di c a last
        - stampare la stringa: "Inserisci una lettera minuscola (o altro carattere per terminare)"
        - leggere  c
   finché c è maggiore o uguale ad 'a' e minore o uguale a 'z'
*/
// restituire il carattere last
}
```

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

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

[File `testf_last_letter.cpp`]

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

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

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

Scrivere un programma per testare la funzione `factorial`:

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

[File `testf_factorial.cpp`]

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

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

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

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

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

[File `testf_replicate.cpp`]

5. 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`]

5.2 Esercizi di base

Per ciascun esercizio in questa sezione considerate quali valori sono accettabili come parametri per le funzioni e verificate la correttezza delle stesse creando main opportuni.

6. Scrivere una funzione `isPrime` con un argomento *n* di tipo intero che restituisce `true` se *n* è *positivo e primo* oppure `false` negli altri casi (*negativo o non primo*).

   [File `testf_prime.cpp`]

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

   [File `testf_ndigits.cpp`]

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

   [File `testf_online_average.cpp`]

9. 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`]

10. Scrivere un programma C++ che gestisca un registro voti per una classe usando varie funzioni (ma non i Vector!). Il programma chiede all'utente di inserire il numero di studenti. Per ogni studente, il programma deve chiedere di inserire il voto (da 0 a 100). Se il voto non è valido, deve richiedere nuovamente l'inserimento. Il programma deve essere composto almeno dalle seguenti funzioni:

    - int leggiVoto(int numeroStudente) → legge e ritorna un voto valido.
    - bool isPromosso(int voto) → ritorna true se il voto è >= 60, altrimenti false.
    - Funzioni per calcolare e aggiornare singolarmente:
      - la somma dei voti
      - il numero di studenti promossi
      - il numero di studenti bocciati
      - il voto massimo
      - il voto minimo

    In dettaglio:

```
// Funzione per leggere un voto valido da tastiera
int leggiVoto(int numeroStudente);

// Funzione che ritorna true se lo studente è promosso
bool isPromosso(int voto);

// Funzione che aggiorna la somma dei voti
int aggiornaSomma(int sommaAttuale, int voto);

// Funzione che aggiorna il conteggio dei promossi
int aggiornaPromossi(int promossiAttuali, int voto);

// Funzione che aggiorna il conteggio dei bocciati
int aggiornaBocciati(int bocciatiAttuali, int voto);

// Funzione che calcola il nuovo massimo
int calcolaMassimo(int votoMassimo, int voto);

// Funzione che calcola il nuovo minimo
int calcolaMinimo(int votoMinimo, int voto);
```

    Tutte queste funzioni devono ritornare un nuovo valore. Dopo aver inserito tutti i voti, il programma deve:

    - Calcolare e stampare la media della classe.
    - Stampare il numero di studenti promossi e bocciati.
    - Stampare il voto più alto e il voto più basso della classe.

    Esempio esecuzione

```
Inserisci il numero di studenti: 3
Inserisci il voto dello studente 1 (0-100): 123
Voto non valido. Riprova.
Inserisci il voto dello studente 1 (0-100): 87
```

```
Inserisci il voto dello studente 2 (0-100): 64
Inserisci il voto dello studente 3 (0-100): 57

Media della classe: 69.3333
Numero di studenti promossi: 2
Numero di studenti bocciati: 1
Voto più alto: 87
Voto più basso: 57
```

[File `testf_mediaVoti.cpp`]

5.3 Esercizi più avanzati

11. Partendo da quanto fatto nell'Esercizio 17 della precedente Sezione, 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`]

12. Partendo da quanto fatto nell'Esercizio 18 della precedente Sezione, scrivere una funzione con un argomento intero `n` compreso fra 1 e 3000 e lo stampa in notazione romana.

    [File `testf_roman.cpp`]

Parte 6

Vector

In questa sezione ci concentriamo sull'utilizzo degli `vector`, imparando a maneggiarli e a definire funzioni che ne facciano uso.

Cheatsheet

`#include <vector>`	header per i Vector
`vector<T> V`	Dichiara la variabile `V` di tipo `vector` con elementi di tipo `T`
`vector<T> W(V)`	Crea `W` e lo inizializza con il contenuto di un altro `vector<T> V` ("costruttore di copia")
`vector<T> V = {...}`	Crea un `vector V` di `T` e lo inizializza con gli elementi del vector specificati (che devono essere di tipo `T`)
`vector<T> V(n)`	Crea un `vector` di `n` elementi inizializzati a un valore "nullo" predefinito che dipende dal tipo T (es. 0 se `T=int`, 0.0 se `T=float`, stinga vuota `""` se `T=string`)
`vector<T> V(n, val)`	Crea un `vector V` di `n` elementi inizializzati ad un valore `val`
`vector<vector<int>>`	`a(M, vector<int>(N, 0));` Dichiarazione di un vector 2D di dimensione MxN
`V.size()`	Ritorna la lunghezza (size) di `V` (cioè il numero di elementi che contiene). Il tipo di ritorno è `size_t` `size_t` è un tipo intero senza segno (unsigned) usato per rappresentare dimensioni e indici di array, vector, stringhe, ecc. È garantito abbastanza grande da contenere la dimensione massima della memoria della piattaforma (32 bit o 64 bit a seconda del sistema). È il tipo restituito da funzioni come `size()`, `sizeof()`, e serve per evitare warning da confronto tra tipi con segno e senza segno. In pratica: quando cicli sugli elementi di un container usando la sua dimensione, è buona pratica usare `size_t`.
`V.capacity()`	Ritorna lo spazio occupato da `V`, include spazio non ancora utilizzato
`V.max_size()`	Ritorna la dimensione massima possibile
`V.empty()`	Ritorna `true` se il vector è vuoto e `false` altrimenti
`V[i]`	Accede all'elemento `i`-esimo (segmentation fault se `i>=V.size()` o `i<0`)
`V[i][j]`	Accede all'elemento nella riga `i` e colonna `j` di una matrice (vector di vector o array 2D). Attenzione: se `i >= V.size()` o `i < 0`, oppure `j >= V[i].size()` o `j < 0`, si verifica comportamento indefinito (spesso segmentation fault).
`V.at(i)`	Accede all'elemento `i`-esimo (ma con eccezione gestibile se `i>=V.size()` o `i<0`)
`V.front()`	Accede al primo elemento (il vector `V` NON deve essere empty)
`V.back()`	Accede all'ultimo elemento (il vector `V` NON deve essere empty)
`W = V`	Copia il contenuto da `V` a `W` (che deve esistere)
`V.push_back(val)`	Aggiunge `val` in coda (ultima posizione) incrementando automaticamente la dimensione
`V.pop_back()`	Elimina l'ultimo elemento decrementando automaticamente la dimensione
`V.resize(n)`	Ridimensiona a nuova size a `n`. Se `n` è più piccolo della size corrente, il vector viene ridotto ai suoi primi `n` elementi, rimuovendo quelli oltre. Se `n` è maggiore della dimensione corrente, il contenuto viene espanso inserendo alla fine quanti più elementi sono necessari per raggiungere una dimensione di `n`.
`V.clear()`	Svuota (porta la size 0)
`using namespace std;`	in testa al file per poter omettere il prefisso `std::` prima di ogni vector (come fatto per rendere il testo più conciso nel resto della sezione)

6.1 Esercizi di riscaldamento

1. **Stampa un vector di `int`.** Scrivere un programma che stampa un vector `a` di `int`. [File `stampaVectorInt.cpp`]

```
// Creare e popolare un vector di lunghezza 7
vector<int> a = {2, 4, 34, 78, 4, 3, 876};
// Iterare sulla variabile intera i da 0 fino alla dimensione del vector (escluso):
// - Stampare il valore i-esimo del vector in posizione i e a capo
```

Output atteso:

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

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

```
// Creare e popolare un vector di float con 5 elementi
vector<float> a = {2.4, 5.67, 34, 28.456, 846.42};
// Iterare sulla variabile intera i a partire dall'ultimo indice e fino a 0 incluso:
// - Stampare il valore in posizione i e andare a capo
```

Output atteso:

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

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

```
// Dichiarare una costante N con valore 10
// Dichiarare un vector di N interi inizializzato con zeri
// Iterare sulla variabile intera i a partire da 0 e  fino alla dimensione del vector (escluso):
// - Assegnare all'elemento i-esimo del vector il valore N - i
// Stampare il vector da a[0] a a[N-1]
```

Output atteso:

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

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

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

// Iterare sulla variabile intera i a partire da 0 e fino alla dimensione del vector (escluso):
// - Assegnare all'elemento i-esimo di a il valore i*v;

// Stampare il vector in ordine crescente da a[0] a a[N-1]
```

Esempio di Output atteso

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

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

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

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

// Stampare il vector in ordine crescente da a[0] a a[N-1]
```

Esempio di Output atteso

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

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

   Esempio di Output atteso

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

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

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

   Esempio di Output atteso

   ```
   Inserisci un valore float per a[0] = 67.7
   Inserisci un valore float per a[1] = 56.7
   Inserisci un valore float per a[2] = 73.1
   Inserisci un valore float per a[3] = 89.5
   Inserisci un valore float per a[4] = 56.0
   I valori contenuti nel vector a sono: { 67.7, 56.7, 73.1, 89.5, 56 }
   La media dei valori contenuti nel vector a è:  68.6
   ```

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

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

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

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

   // Iterare sulla variabile intera x a partire da 0 e fino alla dimensione del vector (escluso):
   // - Assegnare all'x-esimo elemento del vector il valore ax^2+bx+c

   // Stampare il vector in ordine crescente da arr[0] a arr[N-1]
   ```

   Esempio di Output atteso

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

```
Valore di arr[7] = 144.05
Valore di arr[8] = 184.6
Valore di arr[9] = 229.83
```

6.2 Esercizi di base

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

   Suggerimenti:

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

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

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

   L'output, su due righe, è:

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

   Esempio di esecuzione:

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

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

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

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

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

    [File `crivello.cpp`]

    ```
    CRIVELLO DI ERATOSTENE (n)

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

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

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

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

    4) Partire da p=p+1 e scorrere in avanti il vector isprime finché non si trova il primo numero
       NON marcato (isprime[p] è true), oppure finché non è finito il vector

    5) Se il vector è finito, stop. Altrimenti p diventa il numero trovato e si ricomincia dal punto 3)
       (la prima volta sarà per p=3)
    ```

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

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

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

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

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

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

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

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

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

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

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

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

```
const int M = 5;
const int N = 8;

// Dichiarazione di un vector 2D di dimensione MxN inizializzato a tutti 0
vector<vector<int>> a(M, vector<int>(N, 0));
```

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

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

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

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

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

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

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

    [SUGGERIMENTO: conviene avere un vector di contatori (`int`) lungo tanto quanto il vector di ingresso.]

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

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

6.3 Esercizi più avanzati

19. Scrivere un programma che legge un vector di `int source` e scrive in un altro vector `dest` il contenuto del vector `source` ordinato in modo crescente. Poi stampa `dest`. [File `sortInt.cpp`]

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

    [SUGGERIMENTO: Basta fare swap fra le celle poste alla stessa distanza dagli estremi del vector.]

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

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

    Esempi:

    Vector iniziale:

1	1	1	2	3	3	4

    Risultato finale:

1	2	3	4	0	0	0

    Vector iniziale:

2	2	1	2	3	3	4

    Risultato finale:

2	1	2	3	4	0	0

    Vector iniziale:

2	2

    Risultato finale:

2	0

    Vector iniziale:

5

    Risultato finale:

5

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

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

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

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

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

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

Parte 7

Vector avanzati

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

7.1 Esercizi di riscaldamento

1. Scrivere un programma che esegua la ricerca di un elemento `item` (un valore intero) all'interno del vector `v` (un vector di 15 interi letti da input). Se l'elemento viene trovato, il programma deve restituire la posizione dell'elemento nel vector. Se l'elemento è presente più volte, deve restituire la prima posizione in cui compare.

   [File `sequentialSearch.cpp`]

```
// dichiarare una variabile const int N inizializzata a 15
// dichiarare una variabile int item
// leggere item da input
// dichiarare un vector 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 il vector
   - se il valore alla pos corrente (i) e' uguale a item
   -- assegnare true a found e il valore di i a loc
*/
// se trovato, scrivere su output: item " è stato trovato in posizione " loc
// altrimenti scrivere: item " non è stato trovato"
```

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

   [File `selectionSort.cpp`]

```
// Dichiarare una variabile int minIndex
/* Iterare sul vector dalla prima all'ultima posizione (indice i)
    - Memorizzare in minIndex la posizione corrente (cioè i)
    - Iterare sul vector dalla posizione successiva alla corrente (i+1) fino all'ultimo elemento (indice j)
        -- Se il valore alla posizione j è < del valore alla posizione minIndex
            --- Memorizzare j in minIndex
    - Scambiare il valore alla posizione i con quello alla posizione minIndex
*/
```

3. Scrivere un programma che effettui la ricerca dell'elemento `item` (un intero) nel vector `v` (vector 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!]

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 vector v di interi
// leggere v da input (leggiVectorInt)
// visualizzare su output i valori inseriti (stampaVectorInt)
// usare SelectionSort per ordinare v in ordine decrescente
// usare codice di stampaVectorInt (visto in sezioni precedenti) per stampare i valori
```

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

   [File `readMatrix.cpp`]

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

6. Scrivere un programma che stampi su output il vector bidimensionale di interi `A`, di dimensioni `M` × `N`.

   [File `printMatrix.cpp`]

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

7.2 Esercizi di base

7. Confrontare l'efficienza della sequentialSearch vs binarySearch quando si cerca se un certo valore (item) è presente nel vector. Modificare i precedenti esercizi 1 e 3 nel seguente modo:

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

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

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

   (d) Alla fine, stampare il valore di count (numero di accessi effettuati al vector)

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

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

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

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

   [File `verifySymmetry.cpp`]

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

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

]

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

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

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

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

    Alla fine il programma deve stampare `s1` e `s2` ordinati e il vector `d`.

    [File `mergeVectors.cpp`]

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

    Esempio di output:

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

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

    [File `dec2bin.cpp`]

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

    Esempi di esecuzione (coppie input e output):

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

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

    Ad esempio: [1 10 15 18] $\longrightarrow$ [10 15 18 0]

    [File `shiftLeft.cpp`]

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

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

    [File `rotateRight.cpp`]

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

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

    NOTA. Se |N| ≥ lunghezza vector, shift creerà un vettore contenente 0 in ogni cella. Se |N| > lunghezza vector, rotate si comporterà come se |N| fosse sostituito da |N % lunghezza vector|. In particolare, se |N| == k×lunghezza vector (k intero), rotate creerà un vettore uguale a quello dato, ovvero nessuna modifica.

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

    [File `tris.cpp`]

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

7.3 Esercizi più avanzati

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

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

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

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

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

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

Parte 8

Funzioni avanzate ed Eccezioni

Cheatsheet

Passaggio parametri

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

Trattamento errori

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

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

Uso `try`- `catch`:

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

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

8.1 Esercizi di riscaldamento

In questa sezione estenderemo alcuni esercizi già visti nelle sezioni precedenti con: uso delle eccezioni e struttura del codice basata su funzioni.

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

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

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

   [File `testf_swapDouble.cpp`]

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

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

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

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

   [File `testf_average_exception.cpp`]

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

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

```
}
```

Scrivere un programma per testare la funzione `area`:

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

[File `testf_area_exception.cpp`]

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

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

   Usare le eccezioni per gestire i casi di errore.

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

Scrivere un programma per testare la funzione `factorial`:

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

[File `testf_factorial_exception.cpp`]

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

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

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

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

Scrivere un programma per testare la funzione:

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

[File `testf_print_menu.cpp`]

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

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

Scrivere un programma per testare la funzione:

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

[File `testf_print_choice.cpp`]

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

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

Scrivere un programma per testare la funzione:

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

[File `testf_get_choice.cpp`]

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

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

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

Scrivere un programma per testare la funzione secondo il seguente algoritmo

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

[File `testf_use_menu.cpp`]

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

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

[File `testf_menu.cpp`]

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

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

```
*/
// restituire inv moltiplicato per sign
}
```

Scrivere un programma per testare la funzione:

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

[File `testf_reverse.cpp`]

8.2 Esercizi di base

In questa sezione estenderemo alcuni esercizi già visti nelle sezioni precedenti con: uso delle eccezioni e struttura del codice basata su funzioni. Quindi per ciascun esercizio in questa sezione considerate quali valori sono accettabili come parametri, verificatene la correttezza nel corpo della funzione prima di usarli e se necessario segnalate eventuali errori usando il meccanismo delle eccezioni.

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

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

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

   [File `testf_square.cpp`]

9. 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`.]

10. 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`]

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

```
sssqqqqqqq
```

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

[File `testf_replicate2_line.cpp`]

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

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

che sulla diagonale ha 17 caratteri.

[File `testf_rhombus.cpp`]

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

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

13. 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`.]

14. 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`]

15. 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`]

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

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

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

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

[File `testf_population_sim.cpp`]

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

17. Analogamente al punto precedente, scrivere una funzione che prende, oltre ai parametri della funzione al punto 16, 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.]

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

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

    [File `testf_RectInfo.cpp`]

19. Scrivere un programma che ordina un `vector<int>` (già inizializzato con 5 valori nel main) usando la funzione selectionSort che potete implementare partendo dagli esercizi svolti nel precedente laboratorio. Implementare due versioni della funzione: (1) una che riceve il vector per valore; (2) una che riceve il vector per riferimento. Mostrare nel main che solo nel secondo caso l'ordinamento è effettivamente propagato al vector originale.

    ```
    // Prototipi delle funzioni Selection Sort
    void selectionSortByValue(vector<int> v);
    void selectionSortByReference(vector<int>& v);
    ```

    Esempio esecuzione:

    ```
    Vector iniziale: 7 3 9 1 5

    [Funzione by VALUE] Vector ordinato all'interno della funzione: 1 3 5 7 9
    Dopo chiamata per VALORE (nel main): 7 3 9 1 5

    [Funzione by REFERENCE] Vector ordinato all'interno della funzione: 1 3 5 7 9
    Dopo chiamata per RIFERIMENTO (nel main): 1 3 5 7 9
    ```

    [File `testf_selectionSortParamPassing.cpp`]

8.3 Esercizi più avanzati

20. 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`]

21. 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.

22. 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`]

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

    [File `testf_sum_divisible.cpp`]

Parte 9

String

In questa sezione ci concentriamo sull'utilizzo degli string tramite il tipo `std::string` (cf https://cplusplus.com/reference/string/string/).

Cheatsheet	
`#include <string>`	header per i `std::string`
`using namespace std;`	per evitare di ripetere sempre `std`
`string s`	Dichiara la variabile `s` di tipo `string`
`string s="Hello"`	Crea una `string s` e la inizializza con `"Hello"`
`s.length()`	Ritorna la lunghezza della string `s` (cioé il numero di caratteri che contiene)
`s=s+"Hello"`	Concatena (aggiunge) `Hello` alla fine della string `s`
`s=s+'A'`	Concatena (aggiunge il carattere) `A` alla fine della string in `s`
`string t="Hello"+"World"`	**NON FUNZIONA!**
`string t=string("Hello")+string("World")`	Funziona
`s[i]`	Accede all'elemento i-esimo della string (con`0<=i<s.length()`)
`getline (cin,s)`	Legge una string finita con A capo da input e la memorizza nella variabile `s`
`s.substr(i,k)`	Estrae da `s` la string con al massimo `k` caratteri iniziando dalla posizione `i`
`s.erase(i,k)`	Cancella da `s` al massimo `k` caratteri iniziando dalla posizione `i`

9.1 Esercizi di riscaldamento

1. Scrivere un programma che chiede all'utente di inserire una string e poi stampa il numero di spazi nella string.

   [File `numberSpaces.cpp`]

```
// dichiarare una variabile string st
// leggere st da input con getline
// dichiarare una variabile int count e inizializzarla a 0
/* iterare su i a partire da 0 e fino a st.length()
   - se il carattere alla posizione corrente (i) e' uguale a ' '
   -- aumentare count di 1
*/
// scrivere su output: st " ha " count " spazi!"
```

2. Scrivere un programma che chiede all'utente di inserire una string e verifica se questa string rappresenta un numero positivo (deve contenere solo cifre e non iniziare con 0).

   [File `isNumber.cpp`]

```
// dichiarare una variabile string st
// leggere st da input con getline
// dichiarare una variabile bool isNumber e inizializzarla a true
// se st contiene almeno un carattere
//     se il primo carattere è diverso di 0
```

```
//        iterare su i a partire da 0 e fino a st.length()
//        - se il carattere alla posizione corrente (i) non e' un cifra (è <'0' oppure e' >'9')
//        -- mettere isNumber a false
//     altrimenti
//        mettere isNumber a false
// altrimenti
//     mettere isNumber a false
// se isNumber e' true
//     scrivere su output: st " e' un numero positivo."
// altrimenti
//     scrivere su output: st " e' un numero positivo."
```

3. Scrivere un programma che chiede all'utente di inserire un numero positivo, crea la string con la string `"AH"` ripetuta quel numero di volte e la stampa.

   [File `ahVolte.cpp`]

```
// dichiarare una variabile int num
// leggere num da input
// dichiarare una variabile string st e inizializzarla a ""
// iterare su i a partire da 0 e fino a num
// - aggiungere "AH" alla fine di st
// scrivere su output: st
```

4. Scrivere un programma che chiede all'utente di inserire una string, crea una string corrispondente con gli spazi cancellati e stampa la string ottenuta.

   [File `removeSpaces.cpp`]

```
// dichiarare una variabile string st
// leggere st da input con getline
// dichiarare una variabile string st2 e inizializzarla a ""
// iterare su i a partire da 0 e fino a st.length()
// - se il carattere alla posizione corrente (i) di st non e' uno spazio
// -- aggiungere questo carattere alla fine di st2
// scrivere su output:  "La string senza spazi e' " st2
```

9.2 Esercizi di base

Esercizi sulle stringhe. Non usare altre funzioni di libreria se non quelle indicate nel cheatsheet.

5. Scrivere un programma che chiede all'utente di inserire una string e stampa se la string è un palindroma o no. [File `isStringPalindrome.cpp`]

```
// Implementare una funzione che verifica se una stringa è palindroma
// Restituisce true se la stringa è palindroma, false altrimenti
bool isStringPalindrome(const string& str);
```

6. Scrivere un programma che chiede all'utente di inserire un numero positivo (da salvare in una variabile `int`) e poi converte questo numero in una stringa utilizzando un ciclo per enumerare ogni cifra. Alla fine il programma stampa la string ottenuta. [File `numberToString.cpp`]

```
// Implementare una funzione che converte un numero intero in stringa
string numberToString(int number);
```

7. Scrivere un programma che chiede all'utente di inserire una stringa che rappresenti un numero positivo, verificando l'input finché non viene fornito correttamente (vedi esercizi di riscaldamento). Una volta che la stringa è valida, il programma converte il valore in un numero intero e lo stampa. [File `stringToNumber.cpp`]

```cpp
// Funzione che converte una stringa in un numero intero
// Presuppone che la stringa sia già stata validata
int stringToNumber(const string& str);

// Funzione ausiliaria che verifica se una stringa rappresenta un numero positivo valido
// Restituisce true se la stringa contiene solo cifre, false altrimenti
// vedi soluzione ad esercizio di riscaldamento
bool isValidPositiveNumber(const string& str);
```

8. Scrivere un programma che chiede all'utente di inserire una stringa e la inverte direttamente all'interno della stessa variabile di tipo string, senza usare un'altra variabile string. [File `reverseString.cpp`]

```cpp
// Implementare una funzione che inverte una stringa in-place (direttamente nella stessa variabile)
// senza usare un'altra variabile string
void reverseString(string& str);
```

9. Scrivere un programma che chiede all'utente due string e stampa se la seconda string è contenuta nella prima. [File `containsSubstring.cpp`]

```cpp
// Implementare una funzione che verifica se una stringa (substring) è contenuta in un'altra stringa (str)
// Restituisce true se substring è contenuta in str, false altrimenti
bool containsSubstring(const string& str, const string& substring);
```

10. Scrivere un programma che chiede all'utente una string e restituisce una nuova string senza le vocali. [File `removeVowels.cpp`]

```cpp
// Implementare una funzione che rimuove tutte le vocali da una stringa
// Restituisce una nuova stringa senza vocali
string removeVowels(const string& str);
```

Parte 10

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 trigonometriche
`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`.

10.1 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;   // oppure   std::string name, surname;
    int birthYear;
};
```

   Dichiarare due variabili di tipo `Person`:

```
Person me, you;
```

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

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

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

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

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

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

```
me = you;
```

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

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

[File `person.cpp`]

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

```
struct Point {
        double x;
        double y;     // oppure   double x, y;
};
```

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

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

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

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

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

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

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

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

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

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

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

[File `retta.cpp`]

10.2 Esercizi di base

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

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

   [File `rectanglesIn.cpp`]

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

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

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

   [File `whenDate.cpp`]

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

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

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

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

   - calcola e stampa la lunghezza totale della spezzata (vedi Esercizio 2, dist.cpp)
   - verifica se i lati hanno tutti la stessa lunghezza
   - verifica se la spezzata è chiusa (cioè se il primo e ultimo punto coincidono), e se lo è scrive un messaggio: "La linea è chiusa e quindi definisce un poligono" `https://it.wikipedia.org/wiki/Poligono`. Se i lati hanno tutti la stessa lunghezza, aggiungere: "regolare" `https://en.wikipedia.org/wiki/Regular_polygon`.

   [File `poly1.cpp`]

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

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

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

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

10.3 Esercizi più avanzati

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

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

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

    [SUGGERIMENTO: Usate vector di struct.]

    [File `students.cpp`]

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

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

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

    tutte variabili-membro di tipo `double`.

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

    - Somma
    - Differenza
    - Prodotto
    - Rapporto

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

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

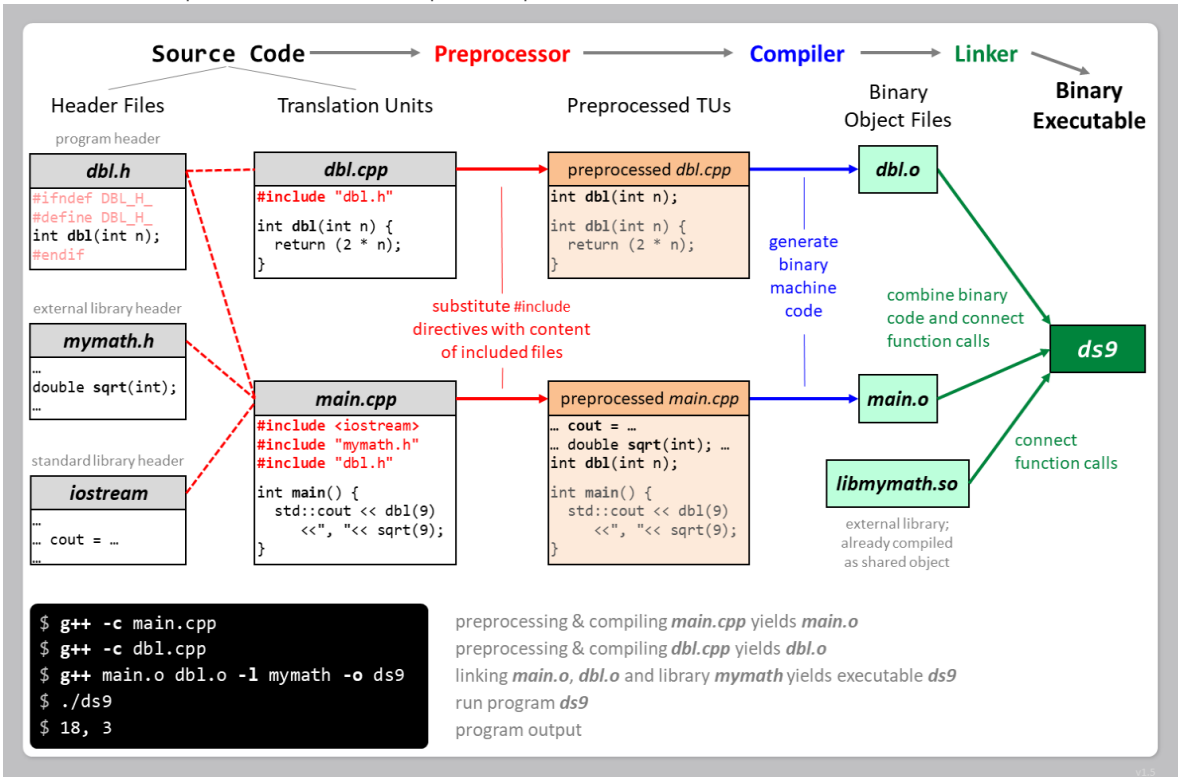
    [File `complexCalc.cpp`]

Parte 11

Compilazione separata

Cheatsheet

Organizzare il codice di un programma in *file separati* rende lo stesso più facile da navigare e mantenere, e inoltre promuove la riutilizzabilità del codice. Le funzioni definite in diversi file sorgente (`.cpp`) possono essere *riutilizzate* in un altro programma includendo il relativo file di intestazione (`.h`). Questo favorisce la *creazione di componenti modulari e riutilizzabili*, che è fondamentale per ottenere un buon design del software.
Suddividere un programma in diversi file sorgente consente anche di effettuare la **compilazione separata**. La compilazione separata ottimizza il processo di compilazione. Infatti, quando si modifica un file sorgente e si ricompila, è sufficiente ricompilare solo quel file, e non l'intero programma. Questa strategia di compilazione è particolarmente vantaggiosa per progetti di grandi dimensioni, dove ricompilare tutto per una piccola modifica è molto dispendioso.
Creazione di File di Intestazione: In C++, i file di intestazione sono un componente chiave della compilazione separata. Contengono ad esempio i prototipi delle funzioni, le dichiarazioni delle struct etc, che informano il compilatore sulla struttura del codice senza rivelarne i dettagli implementativi. I file di intestazione solitamente hanno un'estensione `.h` e vengono inclusi nei file sorgente con la direttiva `#include` (vedi esempio sotto). Le Include Guard `#ifndef X #define X ... #endif` sono una parte fondamentale della creazione di file di intestazione. Impediscono che lo stesso file di intestazione venga incluso più di una volta all'interno di una singola unità di compilazione. Questo è cruciale perché includere un file più volte può causare errori di ridefinizione.



Per dettagli aggiuntivi analizzare la figura dove è mostrato anche come compilare separatamente i vari files. Notare che la direttiva `#include "dbl.h"` è presente anche nel file `dbl.cpp` che contiene l'implementazione della funzione. In questo caso non sarebbe strettamente necessaria in quanto il `.h` contiene solo il prototipo della funzione (mentre è necessaria quando il `.h` contiene ad esempio la

definizione di struct usate poi nelle funzioni). Aggiungere la direttiva `#include` del file `.h` nel file delle funzioni `.cpp` è considerata una buona pratica per evitare alcuni tipi di errori (ad esempio: disallineamento tra nomi/parametri delle funzioni tra il `.h` e il `.cpp`: con l'include vengono già rilevati durante la compilazione separata). Inoltre, se le funzioni nel `.cpp` si richiamano fra loro è necessario che i prototipi siano noti prima delle rispettive chiamate, quindi metterli tutti all'inizio includendo il `.h` lo garantisce. Un prototipo può comparire più volte (in quanto è una dichiarazione senza definizione) a differenza della definizione che deve comparire una sola volta.

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

11.1 Esercizi di riscaldamento

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

   [File `countDigitsInVector_funct.h`] e [File `countDigitsInVector_funct.cpp`] per dichiarazione e definizione della funzione e [File `countDigitsInVector_main.cpp`] per i test.

```
/******************   countDigitsInVector_funct.h   ***************** */

// Header guard: previene inclusioni multiple dello stesso file header
#ifndef COUNTDIGITSINVECTOR_FUNCT_H
#define COUNTDIGITSINVECTOR_FUNCT_H

// Inclusione della libreria vector necessaria per usare il tipo vector
#include <vector>

// Dichiarazione della funzione countDigitsInVector
// Parametri:
//   - v: vector di caratteri passato per riferimento costante (const&)
//        const: la funzione non modifica il vector
//        &: passaggio per riferimento per evitare copie
// Ritorna: int rappresentante il numero di cifre trovate nel vector
int countDigitsInVector(const std::vector<char>& v);

// Fine dell'header guard
#endif

/******************    countDigitsInVector_funct.cpp   ***************** */

// Inclusione del file header corrispondente
#include "countDigitsInVector_funct.h"

// Definizione (implementazione) della funzione countDigitsInVector
int countDigitsInVector(const std::vector<char>& v) {
    // Variabile contatore inizializzata a 0 (conterrà il numero totale di cifre trovate)

    // Ciclo che scorre tutti gli elementi del vector
    {
        // Controlla se il carattere corrente è una cifra
        // Una cifra ha valore ASCII tra '0' (48) e '9' (57)
        // Se è una cifra, incrementa il contatore
    }
    // Restituisce il numero totale di cifre trovate
}

/******************    countDigitsInVector_main.cpp   ***************** */
```

```cpp
// File principale per testare la funzione countDigitsInVector

// Inclusione delle librerie necessarie
#include <iostream>  // Per cout, cin
#include <vector>    // Per usare vector
#include "countDigitsInVector_funct.h"  // Header con la dichiarazione della funzione
using namespace std;

int main() {
    // Test 1: vector con 2 cifre
    // Crea un vector di char con inizializzazione diretta
    // Stampa il contenuto del vector test1
    // Chiama la funzione e stampa il risultato

    // ... implementare altri test
```

11.2 Esercizi di base

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

2. Scrivere una funzione di nome `isPresentInVector` che preso un `vector t` di interi e un intero `x` restituisce `true` se `x` è presente in `t` e `false` altrimenti.

   [File `isPresentInVector_funct.h`] e [File `isPresentInVector_funct.cpp`] per dichiarazione e definizione della funzione e [File `isPresentInVector_main.cpp`] per i test.

3. Scrivere una funzione di nome `maxVector` che preso un `vector t` di interi restituisce il valore del suo elemento più grande. Se `size` del vector vale `0`, la funzione solleverà un eccezione di tipo `string`.

   [File `maxVector_funct.h`] e [File `maxVector_funct.cpp`] per dichiarazione e definizione della funzione e [File `maxVector_main.`] per i test.

4. Scrivere una funzione di nome `allDiffVectorElements` che preso un `vector t` di interi restituisce `true` se `t` non contiene due elementi uguali e `false` altrimenti.

   [File `allDiffVectorElements_funct.h`] e [File `allDiffVectorElements_funct.cpp`] per dichiarazione e definizione della funzione e [File `allDiffVectorElements_main.cpp`] per i test.

5. Scrivere una funzione di nome `orderedStringVector` che preso un `vector t` di `string` restituisce `true` se gli elementi di `t` sono ordinati nel ordine crescente (secondo l'ordine del dizionario) e `false` altrimenti.

   [File `orderedStringVector_funct.h`] e [File `orderedStringVector_funct.cpp`] per dichiarazione e definizione della funzione e [File `orderedStringVector_main.cpp`] per i test.

6. Scrivere una funzione di nome `vectorIncludedInVector` che, preso un `vector<int> t1` e un `vector<int> t2`, restituisce `true` se ogni elemento di `t1` è presente almeno una volta in `t2`, e `false` altrimenti. La funzione deve restituire `true` se `t1` è vuoto. L'ordine degli elementi e le eventuali ripetizioni non influiscono sul risultato.

   [File `vectorIncludedInVector_funct.h`] e [File `vectorIncludedInVector_funct.cpp`] per dichiarazione e definizione della funzione e [File `vectorIncludedInVector_main.cpp`] per i test.

Appendice A

Cheatsheet per lavorare su Linux

Convenzioni grafiche:

`comando`	nome comando
`comando` ⟨`argomento`⟩	argomento obbligatorio
`comando [opzione]`	argomento opzionale
`comando {opz1|opz2}`	argomenti in alternativa

A.1 Comandi bash

Cheatsheet

Abbreviazioni nomi file e cartelle (Si usano come argomenti dei comandi, non sono comandi!

`?`	qualunque stringa di lunghezza 1 Es: `file1 file2 file3` si abbrevia con ⟶ `file?`
`*`	qualunque stringa di qualunque lunghezza Es: `main.cpp aux.cpp library.cpp` si abbrevia con ⟶ `*.cpp`
`.`	cartella corrente
`..`	cartella che contiene quella corrente

Comandi

`ls`	visualizza contenuto della directory corrente
`cd` ⟨`cartella`⟩	cambia directory (cartella) di lavoro
`cd ..`	per spostarsi nella cartella superiore
`pwd`	print working directory, mostra cartella corrente
`rm [-i]` ⟨`file`⟩ `[file2 file3 ...]`	cancella file (con `-i`: chiedi conferma)
`mkdir` ⟨`cartella`⟩	crea cartella
`rmdir` ⟨`cartella`⟩	cancella cartella se vuota
`rm -r` ⟨`cartella`⟩	cancella cartella e il suo contenuto
`rm -r /*`	cancella tutto il filesystem (e non c'è undelete!)
`edit` ⟨`file`⟩	chiama text editor predefinito e apre ⟨`file`⟩
`zip` ⟨`archiveName`⟩ ⟨`files`⟩	crea uno archivio zip contenente i file specificati
`zip CognomeN.zip ese1.cpp ese2.cpp`	(esempio)
`unzip` ⟨`archiveName`⟩	decomprime il contenuto dell'archivio nella cartella corrente

N.B. Un file di testo non deve necessariamente chiamarsi `qualcosa.txt`. Esempi:

- i file sorgenti (estensione `.cpp`)
- i file header (estensione `.h`)
- i file di configurazione di programmi, es. per bash il nome completo è `.bashrc`

Cheatsheet

Controllo I/O ed esecuzione

`prog <` ⟨`file`⟩	`prog` legge standard input (`cin`) da ⟨`file`⟩ anziché da tastiera – *redirection*
`prog >` ⟨`file`⟩	`prog` scrive standard output (`cout`) su ⟨`file`⟩ anziché su schermo
`prog 2>` ⟨`file`⟩	`prog` scrive standard error (`cerr`) su ⟨`file`⟩ anziché su schermo
	Nota: in scrittura ⟨`file`⟩ viene creato o sovrascritto senza chiedere conferma
`prog1 | prog2`	scrive standard output di `prog1` su standard input di `prog2` – *pipeline*
`prog &`	avvia `prog` in background e torna a bash
`prog1; prog2`	avvia `prog1` e al termine avvia `prog2`
Tasto `control-C`	arresta programma in esecuzione
Tasto `control-Z`	mette in pausa programma in esecuzione
`fg`	il programma in pausa riprende esecuzione in foreground
`bg`	il programma in pausa riprende esecuzione in background

A.2 Editing dei file sorgente

Cheatsheet

Qualunque text editor va bene

Visualizzare numeri di linea:

gedit, pluma	menu *Preferences* → linguetta *View* → *Display line numbers*
jedit	menu *Global Options* → linguetta *Gutter option* → *Line Numbering*
vim	`(esc):set numbers` o `:se nu`
emacs (in finestra grafica)	menu *Options* → sottomenu *Show/Hide* → *Line Numbers*
emacs (in terminale)	(esc)x linum-mode

A.3 Compilazione

Cheatsheet

Usi comuni del comando `g++`

`g++` ⟨`filesorgente.cpp`⟩	compila ed esegue linking, crea eseguibile `a.out`
`g++` ⟨`filesorg1.cpp`⟩ ⟨`filesorg2.cpp`⟩	compila tutti ed esegue linking insieme, crea eseguibile `a.out`
`g++` ⟨`filesorgente.o`⟩	esegue linking di file già compilato
`g++` ⟨`fileheader.h`⟩	**NO!**
`g++ ...argomenti, opzioni... 2> file`	scrive errori e warning su ⟨`file`⟩
`g++` ⟨`...argomenti...`⟩ `2> file; head file`	scrive errori e warning su ⟨`file`⟩ e subito dopo mostra le prime 10 righe

Opzioni utili di `g++`

`-std=c++14`	Segue le regole del linguaggio C++ standard 2014
`-Wall`	Stampa tutti i warning, inclusi quelli poco importanti
`-o` ⟨`nome`⟩	l'eseguibile si chiamerà ⟨`nome`⟩ anziché `a.out`
`-g`	Crea eseguibile contenente simboli leggibili per debugging

A.4 Debugging

Cheatsheet

Procedura di debugging

1) leggi i messaggi (suggerimenti)
2) stampe per isolare il punto
3) gdb: `g++ -g`
4) valgrind

Programmi utili per il debugging

`gdb` ⟨`...argomenti...`⟩	linea gdb
Comandi gdb	
`valgrind`	memoria (dettagli ed esempi forniti in CheatSheet della Parte **??**)

Appendice B

Riepilogo introduzione a C++

B.1 Struttura del programma

- Un blocco di istruzioni è racchiuso fra parentesi graffe, aperta e chiusa

```
{
        // istruzione
        // istruzione
        // ...
}
```

- Un blocco di istruzioni può contenere altri blocchi al suo interno

```
{
        // istruzione
        {
                // istruzione
                // istruzione
        }
        // ...
}
```

- Ogni istruzione o blocco contenuti in un altro blocco va graficamente rientrato (*indentato*) di un numero fisso di caratteri per renderlo visivamente evidente.

  N.B. Nel linguaggio C++ questa non è una regola di sintassi ma solo di stile, però è così importante che altri linguaggi (Python) è invece obbligatoria.

- Un programma deve sempre contenere una funzione chiamata `main` che restituisce un valore di tipo `int` (che normalmente non ci interessa).

- L'intestazione di una funzione è seguita da un blocco di codice che costituisce il *corpo* della funzione

```
int main()
{
        // istruzione
        // istruzione
        // ...
}
```

B.2 Regole grammaticali

- I caratteri whitespace (spazio, tabulazione e a-capo) si possono usare a piacere. Ogni istruzione è quindi terminata da un punto e virgola `;` .

- Nel codice si possono inserire commenti mono-riga (da `//` a fine riga) o multi-riga (da `/*` a `*/`).

- Nella sintassi C++ le funzioni sono le componenti “attive” (agiscono). Le componenti “passive” su cui le funzioni agiscono sono:
  - Literal (costanti letterali) come `1`, `3.14159`, `'A'`, `"Hello, world"`.
  - Variabili, come `i`, `x`, `cateto`, `num_elementi`
  - Costanti con nome, come `PI`, `OREINUNGIORNO`, `MAXMEM`
- Sia le funzioni che le variabili e costanti sono identificate da un nome (o, appunto, *identificatore*)
- Le regole per comporre gli identificatori sono:
  - Lunghezza praticamente illimitata (dipende dal compilatore)
  - Solo caratteri alfabetici, numerici e underscore `_`
  - I caratteri alfabetici maiuscoli e minuscoli sono diversi
  - Il primo carattere non può essere numerico

  N.B. L’uso di identificatori tutti maiuscoli per le costanti è solo una possibile convenzione, non è una regola del linguaggio.

B.3 Dichiarazioni

- Gli oggetti con nome, prima di essere usati, vanno definiti attraverso una *dichiarazione*. Una dichiarazione si può mettere in qualunque punto del codice.
- In una dichiarazione deve essere indicato il tipo di un oggetto (o il tipo del valore restituito, per una funzione – vedi `main` sopra)
- Alcune dichiarazioni-tipo:
  - `int i; //variabile`
  - `const float PI = 3.14159; //costante`
  - `int main() //funzione`

B.4 Variabili, inizializzazioni, assegnazioni

- Una funzione che restituisce un valore di tipo `int` deve sempre terminare la sua esecuzione con l’istruzione “restituisci un valore di tipo intero” (per esempio `return 0`). Lo stesso vale per qualunque altro tipo. Lo standard ci concede un’eccezione (possiamo non fare alcun `return`) solo per la funzione `main`.
- Una dichiarazione di variabile può contenere anche una *inizializzazione* che stabilisce il valore iniziale della variabile: `float angolo = 90;`. Vedi caso costante.
- Una variabile non inizializzata contiene un valore (non esiste una variabile che contiene “niente”), ma questo valore non è prevedibile e non ha alcun senso.
- Successivamente all’inizializzazione, il valore di una variabile viene impostato con una istruzione di assegnazione, usando l’operatore di assegnazione `=` . P.es. `x = 10;` .
- L’assegnazione prevede una variabile sul lato sinistro dell’operatore `=`, un valore o una espressione più complessa sul lato destro. I ruoli non sono interscambiabili. L’espressione viene valutata, e al termine dell’esecuzione la variabile contiene il valore risultante.
- Inizializzazione ed assegnazione hanno sintassi simili ma non sono la stessa operazione. Con i tipi semplici questa differenza non si nota, ma occorre saperlo.

B.5 Scope e visibilità

- Una dichiarazione, e quindi l'oggetto che questa definisce, esiste (scope) ed è accessibile (visibility) nel blocco in cui è definita e in tutti gli eventuali blocchi contenuti in esso, a partire dalla riga che contiene la definizione.

  (Nota: Questo perché il compilatore legge il codice sorgente una volta sola, dall'inizio alla fine, e non può usare prima una definizione che incontrerà dopo)

- Le funzioni possono essere dichiarate solo nello "scope globale", cioè al di fuori di qualunque altra funzione.

- Se in un blocco viene dichiarata una variabile con lo stesso nome di un'altra definita nello **stesso blocco**, è errore di sintassi e la compilazione non ha successo.

```
{
        int i;
        int i; // ERRORE
}
```

- Se in un blocco viene dichiarata una variabile con lo stesso nome di un'altra definita in un blocco **più esterno** (che lo contiene), non è errore; la variabile più esterna però non è *visibile* nel blocco più interno

```
{
        int i;
        {
                int i; // non e' errore, ma in questo blocco esiste solo questa i
        }
}
```

B.6 Espressioni e operatori

- Una espressione è un costrutto linguistico del quale si può calcolare il valore.

- Una espressione è:

  – Una costante literal (es. `1`)

  – Una costante con nome (es. `PI`)

  – Una variabile (es. `x`)

  – Una chiamata di funzione (es. `sqrt(2)`, "square root")

  – Più elementi tra quelli elencati, combinati attraverso operatori

  Il valore dell'espressione è uno, indipendentemente dal numero di elementi e operatori, e ha un suo tipo. P.es. `2.1 + 3.9` ha il valore `6.0` che è un unico valore numerico floating point.

- Esistono molti operatori, ciascuno dei quali agisce su determinati tipi; quelli più intuitivi sono gli operatori aritmetici `+ - * /`, che operano su interi o su floating point. L'operatore modulo `%` opera solo su interi.

- Ogni operatore ha un determinato livello di precedenza. P.es. somma e sottrazione hanno un livello di precedenza più basso di moltiplicazione e divisione. Le parentesi tonde `( )` si possono usare per cambiare la precedenza in una espressione.

- In una espressione che include elementi di tipi compatibili ma misti, per esempio float+double o int—float, il tipo "meno capace" viene convertito (*promosso*) al tipo "più capace".

- Una espressione che include elementi di tipi non compatibili, per esempio intero*literal stringa, genera errore. Ogni operatore accetta certe combinazioni di tipi e non altre.

Dal codice sorgente al programma eseguibile

- Fasi del processo:
  1. Preprocessore: riceve il mio sorgente (uno o più) contenente direttive e codice C++, esegue direttive, produce un sorgente in solo codice C++
  2. Compilatore: riceve sorgente in solo codice C++ (uno o più), traduce in linguagio macchina, produce file "oggetto"
  3. Linker/loader: riceve file oggetto, traduce riferimenti simbolici in indirizzi di memoria, unisce moduli di libreria standard (o anche altre se da me indicate); produce un unico file eseguibile
- Comando da usare con il compilatore "GCC – Gnu Compiler Collection":
  - minimale (produce a.out): `g++ miosorgente.cpp`
  - voglio dare un nome al file eseguibile: `g++ miosorgente.cpp -o mioeseguibile`
  - voglio compilare più file: `g++ miosrg1.cpp miosrg2.cpp -o mioeseg`

  N.B. Il comando `g++` esegue automaticamente tutti i tre passi indicati sopra.
- Per eseguire il mio eseguibile: `./mioeseguibile`
- Per usare una funzione o altro, che sia definito in un modulo della libreria, occorre che il sorgente includa le relative definizioni – esattamente come per gli oggetti creati da me. Le definizioni si trovano in *file header* forniti con la libreria.
- Direttiva del preprocessore per includere uno header di libreria: `#include`
- Sintassi per libreria installata nel sistema: `#include <iostream>`
- Sintassi per header mio contenuto nella mia cartella dei file sorgenti: `#include "mioheader"`

Input/output

- Le funzionalità sono fornite dal modulo "iostream" della libreria standard, quindi occorre `#include <iostream>`
- Ingresso dall'"oggetto" cin: `cin >> var1 >> var2;`
- Uscita sull'"oggetto" cout: `cout << var1 << var2;`
- Gli identificatori nella libreria standard sono "racchiusi" nel namespace `std`, quindi
  - o uso ogni volta l'operatore `::` per riferirmi al namespace corretto: `std::cin`, `std::cout`, `std::endl`
  - oppure una volta per tutte indico `using namespace std;` a inizio programma

B.7 Errori comuni

- Usare variabile non dichiarata
- Usare variabile prima di dichiararla
- Usare variabile con un nome simile, ma non uguale, a quello nella dichiarazione
- Dimenticare il punto e virgola
- Dimenticare il qualificatore di namespace, o l'istruzione `using namespace ...`
- Il literal "stringa vuota" `""` esiste. Il literal "carattere vuoto" `''` non esiste.

Appendice C

Linguaggio C(++)

Useremo un sottoinsieme del linguaggio C++, che chiameremo **C(++)**. Il C++ è un linguaggio molto esteso e molto complesso e contiene costrutti sofisticati, che non utilizzeremo in questo corso. Il C(++) coincide in buona parte col linguaggio ANSI C (predecessore del C++) esteso con alcuni costrutti del C++. Nello svolgimento degli esercizi e agli esami **non sarà consentito usare costrutti C++ o C diversi da quelli del C(++) visti a lezione e in laboratorio**. Ad esempio, non sarà consentito usare: tutti i costrutti object-oriented, i tipi template (ad eccezione dei vector), la sintassi avanzata per i cicli (range for), ecc.

I costrutti del C(++) sono riassunti nel seguente *cheatsheet*. La colonna a sinistra esemplifica i costrutti, la colonna destra ne spiega il significato. Le parti di codice tra parentesi acute (ad esempio `<body>` si riferiscono a pezzi di codice generico che può contenere espressioni o blocchi di istruzioni.

Cheatsheet

Commenti	
`//<commento>`	commento a fine riga
`/*<commento> */`	commento anche su più righe
Direttive	*solo per l'inclusione di moduli esterni*
`#include <nomemodulo>`	inclusione moduli della libreria standard
`#include "nomemodulo"`	inclusione moduli locali
Namespace	*esclusivamente per l'uso della libreria standard*
`using namespace std;`	qualificatore di default
`std::`	qualificatore per il singolo identificatore della libreria
Tipi base	*predefiniti dal linguaggio*
`bool`	Booleani
`char`	caratteri alfanumerici
`int, long int`	interi con segno
`unsigned int`	interi senza segno
`float, double`	reali (razionali, numeri con la virgola)
Tipi di libreria	*definiti dalla libreria standard - richiedono i moduli appositi*
`string`	stringhe di caratteri
`vector<T>`	array dinamici di tipo base noto `T`
`ifstream, ofstream`	tipi stream (per usare i file)
Costruttori di tipi	*per definire nuovi tipi*
`typedef T <nuovotipo>;`	definizione di nuovo tipo in base a tipo noto `T`
`typedef T * <nuovotipo>;`	definizione di nuovo tipo puntatore a tipo noto `T`
`struct <nometipo> {<campi>};`	dichiarazione di tipo struct
Dichiarazione di variabili	*per definire nome e tipo delle variabili*
`T <nomevar>;`	dichiarazione variabile di tipo noto `T`
`T <nomevar>[<cost>];`	dichiarazione di variabile array di tipo noto `T`
`struct <nomestruct> {<campi>} <nomevar>;`	dichiarazione di variabile struct
`T * <nomevar>;`	dichiarazione di variabile di tipo puntatore a tipo noto `T`
Dichiarazione di costanti	*per definire costanti di programma*
`const T <nomeconst> = <constexp>;`	definizione di costante di tipo noto `T` mediante espressione costante
Operatori di riferimento e dereferenziazione	*da usarsi con i puntatori*
`& <var>`	restituisce il puntatore alla variabile `<var>`
`* <varp>`	restituisce il contenuto della locazione (variabile) puntata dal puntatore `<varp>`

Cheatsheet

Assegnazione	*per assegnare valori alle variabili*
`<var> = <exp>;`	valuta `<exp>` e ne assegna il valore a `<var>`
`<var>++; <var>--;`	incremento e decremento di variabile
`<var> += <exp>; <var> *= <exp>;` ecc.	abbreviazioni di `<var> = <var> + <exp>; <var> = <var> * <exp>;` ecc.
Controllo di flusso	*condizionali e cicli*
`if (<expbool>) <body>`	condizionale semplice
`if (<expbool>) <bodyt> else <bodyf>`	condizionale a due rami
`switch(<exp>) { <cases> }`	condizionale a più alternative
`while (<expbool>) <body>`	ciclo while
`do <body> while(<expbool>);`	ciclo do-while
`for(<init>;<expbool>;<update>) <body>`	ciclo for **solo con sintassi standard**
Salti incondizionati	
`break;`	per uscire da cicli o `switch`
`continue;`	per continuare un ciclo saltando le restanti istruzioni del ciclo corrente
Funzioni	*costrutti necessari a dichiarazione e chiamata di funzione*
`int main() <body>`	funzione `main`
`void`	tipo vuoto per funzioni che non restituiscono valori
`T <nomefunc>(<listaparametriformali>) <body>`	dichiarazione di funzione a valori in tipo noto `T` (può essere `void`)
`T <nomeparametro>`	parametro di tipo `T` passato per valore (nella lista parametri della funzione)
`T & <nomeparametro>`	parametro di tipo `T` passato per riferimento
`T <nomeparametro>[]`	parametro array di tipo `T`
`return <exp>;`	uscita da funzione (`<exp>` è vuota se funzione `void`)
`<nomefunc>(<listaparametriattuali>);`	chiamata di funzione
Input/Output standard	*costrutti per input da tastiera e output a video (standard I/O)*
`std::cout << <exp>;`	inserimento: scrive in standard output il valore di `<exp>`
`std::cin >> <var>;`	estrazione: legge da standard input un valore e lo assegna alla variabile `<var>`
`... << <expa> << <expb> ...`	concatenazione di output
`... >> <vara> >> <varb> ...`	concatenazione di intput
`std::endl`	simbolo di fine linea (costante di tipo `char`)
Input/Output su file	*costrutti per I/O su file ASCII*
`ifstream <vari>; ofstream <varo>;`	dichiarazione di variabili stream in input e in output
`ifstream <vari>(<nomefilei>);`	dichiarazione di variabile stream con apertura del file in input
`iofstream <varo>(<nomefileo>);`	dichiarazione di variabile stream con apertura del file in output
`<vars>.open(<nomefile>);`	apertura del file e associazione alla variabile stream `<vars>`
`<varo> << <exp>;`	scrittura su file (anche concatenata)
`<vari> >> <var>;`	lettura da file (anche concatenata)
`<vars>.close();`	chiusura del file associato alla variabile stream `<vars>`
`<vars>.eof()`	test di fine file
Operatori su vector	*operazioni sui tipi `vector` **(solo questo sottoinsieme)***
`<v>[<expint>]`	accesso diretto su vector `<v>` (anche su array)
`<v>.at(<expint>)`	accesso diretto su vector `<v>` (no su array)
`<v>.size()`	numero di locazioni
`<v>.resize(<expint>);`	ridimensionamento
`<v>.push_back(<exp>);`	aggiunta di un elemento in coda
Operatori su string	*operazioni tipo `string` **(solo questo sottoinsieme)***
`<s>[<expint>]`	accesso diretto su stringa `<s>`
`<s>.size()`	numero di caratteri nella stringa
`<s>.resize(<expint>);`	ridimensionamento
`<s>.push_back(<exp>);`	aggiunta di un elemento in coda
`<s>.clear();`	cancellazione contenuto
`<s1> + <s2>`	operatore di concatenazione
`c_str(<s>);`	funzione che restituisce un array di caratteri equivalente alla string `<s>`