Introduzione alla Programmazione
a.a. 2020/21

Eserciziario

30 settembre 2020

Indice

Come usare l'eserciziario

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

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

Organizzazione dell'eserciziario

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

Ciascuna parte è introdotta da un *cheatsheet* riassuntivo delle regole e degli argomenti trattati nel capitolo, simile ai "foglietti" usati per copiare agli esami, e poi è strutturato in tre sezioni:

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

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

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

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

  Ci aspettiamo che voi produciate un programma simile a

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

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

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

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

Note utili

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

Elementi fondamentali di programmazione

Parte 1

Espressioni, lettura e assegnazione

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

Cheatsheet

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

1.1 Esercizi di riscaldamento

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

   (a) Seguire l'algoritmo proposto (che fissa una serie di dettagli ulteriori)

```
// 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)

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

Confrontando i due algoritmi, quali aspetti ritenete vincenti? provate a elaborare il vostro algoritmo ideale.

2. Come esercizio 1, ma prima di ogni input viene dato un messaggio di richiesta, del tipo "inserisci il valore di ..."

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.

   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)

```
// 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.

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.

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.
8. Scrivere un programma che chiede all'utente in che anno è nato e stampa quanti anni ha.
9. Scrivere un programma che calcola perimetro e area di un triangolo, dopo aver chiesto e letto i dati necessari.
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.
11. Scrivere un programma che calcola circonferenza e area di un cerchio, dopo aver chiesto e letto i dati necessari.
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.
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.
[**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*.
[**SUGGERIMENTO:** se alla media sottraggo la semidistanza, che valore ottengo?]
15. Scrivere un programma che scambia fra loro i valori di due variabili senza usare variabili di appoggio.
[**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. (errore comune dei principianti): `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`	
	`bool`	tipic. 1 byte
Tipi floating point (reali)	`float`	32 bit (4 byte)
	`double`	64 bit (8 byte)
	`long double`	80 bit (10 byte)

Costanti letterali: `10` ← costante di tipo intero, `1` ←costante di tipo intero (incluso `bool`), `10.` ←costante di tipo floating point.

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!"`

Altri tipi possono essere **definiti dal programmatore** oppure **definiti in librerie esterne** (richiedono un `#include` appropriato).

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.

```
// 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)

```
// 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;
// se a0 maggiore di a1 scambiare fra loro a0 e a1, cioè
// - assegnare ad aux il valore di a1, ad a1 il valore di a2...
//    ... e ad a0 il valore di aux
//    (a questo punto a0 <= a1 e a1 == aux)
// se a0 maggiore di a2
// - assegnare ad aux il valore di a0, ad a0 il valore di a2 e...
//    ... ad a2 il valore di aux
// se a1 maggiore di a2 scambiare fra loro a1 e a2, cioè
// - assegnare ad aux il valore di a2, ad a2 il valore di a1,...
//    ... ad a1 i valore di aux (a questo punto a0<=a1 e 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).

4. Scrivere un programma che legge tre numeri reali e li stampa in ordine decrescente

5. Scrivere un programma che legge un numero intero da input e stampa se è o no divisibile per 13.

6. Scrivere un programma che verifica se tre numeri reali dati in input possono essere i lati di un triangolo, cioè se nessuno di essi è maggiore della somma degli altri due o minore del valore assoluto della loro differenza.

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

   Quindi, in cascata (ovvero usando il risultato di ciascuna operazione come argomento per la successiva), lo divide per 4.9, per 3.53 e per 6.9998. Poi, sempre in cascata, moltiplica per 4.9, per 3.53 e per 6.9998.

   Infine confronta il risultato ottenuto con il numero iniziale e se rappresentano due numeri reali diversi stampa `"moltiplicare NON è l'inverso di dividere"`.

8. Scrivere un programma che verifica se un numero intero dato in input rappresenta o meno un anno bisestile.

9. Scrivere un programma che implementa un turno di gioco di forbice carta sasso, ovvero che legge in input la mossa dei due giocatori e restituisce in output chi ha vinto.

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

    - "Freddo dannato" se `Temp` è compreso fra -20 e 0
    - "Freddo" se `Temp` è compreso fra 1 e 15
    - "Normale" se `Temp` è compreso fra 16 e 23

- “Caldo” se `Temp` è compreso fra 24 e 30
- “Caldo da morire” se `Temp` è compreso fra 31 e 40
- “Non ci credo, il termometro deve essere rotto” se `Temp` è superiore a 40 o inferiore a -20

11. Scrivere un programma che legge un numero intero compreso fra 1 e 12 e stampa il nome del mese corrispondente (1==gennaio...). Se il numero non è compreso fra 1 e 12 stampa un messaggio di errore ed esce.

2.3 Esercizi più avanzati

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”.

13. Scrivere un programma che scrive in lettere i nomi italiani delle ore, approssimati per difetto a 15 minuti. Il programma deve prendere in input due valori interi, uno tra 1 e 12 (ore) e l’altro tra 0 e 59 (minuti) e se i valori dati in input non rispettano il vincolo stampa un messaggio di errore ed esce ritornando -1 come codice di errore. Se l’input è corretto, scrive “Sono le ore " seguito dal valore delle ore (p.es. se è 11 scrive “undici", ma se è 1 scrive “una") e dal valore dei minuti, approssimato al quarto d’ora (p.es. se è 18 scrive “ e un quarto", se è 39 scrive “ e mezza", se è 55 scrive “ e tre quarti"; se è 0 invece non scrive niente). Infine, se i minuti non sono divisibili esattamente per 15, scrive “ circa".

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.

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

Parte 3

Cicli

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

Cheatsheet

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

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

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

Ciclo `for` in generale:

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

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

Ordine esecuzione:
`istruzione-1` → `espressione-booleana` → `code-block` → `istruzione-2` → da capo

Ciclo while

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

Ciclo do. . . while

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

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

Quale costrutto devo usare? Regola di prima approssimazione:

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

Tecnica del look-ahead

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

3.1 Esercizi di riscaldamento

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

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

   Implementate questo esercizio in tre varianti: usando un ciclo for oppure un ciclo while oppure un ciclo do while.

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.
   [**SUGGERIMENTO:** Ricordate che i caratteri sono tipi numerici! Si possono confrontare e anche sommare o sottrarre. Le lettere maiuscole vengono rappresentate con numeri consecutivi, quindi per sapere se un carattere rappresenta una lettera maiuscola basta verificare che sia maggiore o uguale del carattere 'A' e minore o uguale al carattere 'Z'.]

```
// stampare la stringa "Inserisci una lettera maiuscola"
// dichiarare una variabile first di tipo char
/* ripetere
        - leggere first
   finché first minore di `A' o maggiore di `Z'
   // Hint: ovvero finché l'utente non inserisce una lettera maiuscola
*/
// Hint: a questo punto sappiamo che first è una lettera maiuscola!
// dichiarare una variabile c di tipo char inizializzata con `Z'
// Hint: a questo punto sappiamo che first <= c!
/* ripetere
        - se c è minore di first
                -- assegnazione di c a first
        - stampa della stringa "Inserisci una lettera maiuscola (o altro...
        ... carattere per terminare)"
        - lettura di c
   finché c è maggiore di `A' e minore di `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 $n! = n*(n-1)*(n-2)*...*1$.

```
// stampare la stringa "Inserire un numero positivo: "
// dichiarare una variabile intera n
// leggere n
// se n è minore di zero
//      - stampare "Avevo detto positivo!"
//      - uscire dal programma ritornando codice di errore 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:

```
// stampare la stringa "Inserisci un numero maggiore di 0: "
// dichiarare una variabile len di tipo int
// leggere len
// se len non e` maggiore di zero
//      - stampare "Avevo detto positivo!"
//      - uscire dal programma ritornando codice di errore 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. 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:

```
/* 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
        - dichiarare una variabile intera answer
        - 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 chiede all'utente numeri interi positivi e stampa su una nuova riga tante `` `|' `` quanto è grande il numero (come le aste all'asilo), finché l'utente non vuole terminare:

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

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

```
// stampare la stringa "Inserire un numero positivo: "
// dichiarare una variabile intera k
// leggere k
// se k è minore di zero
//      - stampare "Valore non valido"
//      - uscire dal programma ritornando il codice di errore 666
// stampare su una nuova riga la stringa "Rovesciando " seguita da k
// dichiarare una variabile intera inv inizializzata a zero
/* finché k è maggiore di zero
        - dichiarare una variabile intera mod inizializzata con il resto della...
        ... divisione intera di k per 10
        - assegnare a k il quoziente di k per 10
        - assegnare a inv la moltiplicazione di inv per 10
        - assegnare a inv la somma di inv e mod
*/
// stampare la stringa " si ottiene " seguita da inv
```

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

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

(b) si stampi il numero ottenuto leggendo il numero dato da destra verso sinistra. Ad esempio su -56 si stampi -65.

3.2 Esercizi di base

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

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

11. Scrivere un programma che legge due numeri interi strettamente positivi e stampa il trapezio rettangolo fatto di `x' con le basi lunghe quanto i numeri letti, e l'altezza pari alla differenza fra le basi più uno. 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.

12. Scrivere un programma che chiede all'utente un numero intero positivo n e stampa un rombo di asterischi che sulla diagonale ha $2 * n + 1$ caratteri. Ad esempio su 8 stampa

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

    che sulla diagonale ha 17 caratteri.
    [**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.]

13. 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.

14. Scrivere un programma che chiede all'utente un numero intero e verifica se è *palindromo*, ovvero se le sue cifre (in base 10) lette da destra a sinistra corrispondono alle cifre lette da sinistra a destra (ad esempio 165373561 è palindromo). Dopo la verifica stampa il risultato all'utente.

15. Scrivere un programma che chiede all'utente un numero reale e lo legge. Poi chiede all'utente di provare a indovinarne la radice quadrata e se l'utente inserisce il valore giusto gli dice `"Bravo!"` ed esce, altrimenti gli propone di riprovare finché l'utente non riesce ad indovinare.
    Per provare questo programma usate come dati 25.3268614564 la cui radice quadrata è 5.03258 (se preferite altri valori, vi conviene partire da un numero con cifre decimali e farne il quadrato, in modo da evitare errori di approssimazione dovuti ai troncamenti).

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

3.3 Esercizi più avanzati

17. Scrivere un programma che verifica se un numero intero positivo dato in input è un *numero di Armstrong*. Un intero positivo 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 $1634 = 1^4+6^4+3^4+4^4 = 1+1296+81+256$.

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

19. Scrivere un programma che legge un intero positivo compreso fra 1 e 3000 e lo stampa in notazione romana. 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 addittiva (non posizionale come i numeri arabi), partendo dai simboli che rappresentano i numeri più grandi a sinistra e man mano scendendo con simboli che rappresentano numeri sempre più piccoli; ad esempio MMXVII rappresenta 2017 come $1000+1000+10+5+1+1$.
    - si possono incontrare dei simboli in ordine inverso, ma in questo caso i valori invece di andare sommati vanno sottratti; questo meccanismo può essere usato solo per i numeri 4, rappresentato da IV, 9, rappresentato da IX, 40, rappresentato da XL, 90, rappresentato da XC, 400, rappresentato da CD, e 900, rappresentato da CM. Quindi ad esempio 1984 si rappresenta con MCMLXXXIV e 999 si rappresenta con CMXCIX (e non con IM).

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

Parte 4

Array

Impariamo ad utilizzare gli array.

Cheatsheet

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

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

4.1 Esercizi di riscaldamento

1. **creaArrayInt, Crea un array di prova con elementi di tipo `int`.** Scrivere un programma che dichiara un array `v` di `N` interi e lo "popola" (assegna valori ai suoi elementi).

   ```
   // Dichiarare una costante N con valore 10
   // Dichiarare un array a di N interi
   // Iterare sulla variabile intera i a partire da 0 e fino a N escluso:
   //      Assegnare all'elemento i-esimo di a il valore N-i
   ```

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

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

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

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

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

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

3. **leggiArrayInt, Leggi un array di `int` da tastiera.** Scrivere un programma che dichiara un array `v` di `N` interi e lo "popola" leggendo valori da input.

```
// Dichiarare una costante N con valore 10
// Dichiarare un array a di N interi
// Iterare sulla variabile intera i a partire da 0 e fino a N escluso:
//      Dichiarare una variabile intera val
//      Stampare il messaggio composto da:
//         - la stringa "Valore n. "
/          - il valore di i
//         - il separatore ": "
//      Leggere da input un valore di val
//      Assegnare all'elemento i-esimo di a il valore di val
```

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

4. Scrivere un programma che legge `N` valori reali, li memorizza in un array di lunghezza `N`, e ne stampa la media.

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

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

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

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

4.2 Esercizi di base

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

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

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

   Quindi stampa `source` e `dest` (lasciando una riga vuota in mezzo per chiarezza).

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

```
CRIVELLO DI ERATOSTENE (n)

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

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

```
2) Inizialmente, sia p pari a 2, il numero primo più piccolo.

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

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

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

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

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

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

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

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

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

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

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

4.3 Esercizi più avanzati

10. Scrivere un programma `palyndrome` che legge un array `a` e calcola un valore di tipo `bool` che vale `true` se l'array è palindromo. Poi stampa un messaggio che comunica il risultato all'utente.

    [**SUGGERIMENTO:** usare il programma `reverse`.]

11. Scrivere un programma che legge un array di `int` e stampa la frequenza di ogni valore contenuto (il numero di volte che compare).

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

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

13. Scrivere un programma che legge un array di `int source` e scrive in un altro array `dest` il contenuto dell'array `source` ordinato in modo crescente. Poi stampa `dest`.

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

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

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

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

    Al ternine del procedimento, i valori che non sono stati eliminati devono essere contenuti in elementi consecutivi dello stesso array, e gli elementi rimanenti devono essere azzerati. Il programma infine stampa tutti gli elementi non zero.

    Esempi:

    Array iniziale:

1	1	1	2	3	3	4

    Risultato finale:

1	2	3	4	0	0	0

    Array iniziale:

2	2	1	2	3	3	4

    Risultato finale:

2	1	2	3	4	0	0

    Array iniziale:

2	2

    Risultato finale:

2	0

    Array iniziale:

5

    Risultato finale:

5

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

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

Parte 5

Struct

Impariamo ad utilizzare il tipo di dato struct, che ci permette di aggregare dati anche quando non sono 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";
```

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

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

Operano tutte su dati di tipi `float` o `double`.

5.1 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;
    std::string birthYear;
};
```

   Dichiarare due variabili di tipo `Person`:

```
Person me, you;
```

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

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

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

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

```
me = you;
```

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

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

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

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

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

```
// 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

```
// 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 di calcola come $D(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.]

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

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

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

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

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

5.2 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 verifica se uno dei due rettangoli sia contenuto nell'altro, e stampa un messaggio di output opportuno.

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

   Scrivere un programma che legge una data D1 e, dopo averne verificato la correttezza controlla se D1 sia una data passata o futura e stampa un messaggio di output opportuno.
   [**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 di 2018 allora la data è sicuramente passata, se è maggiore di 2018 allora la data è sicuramente futura. Se è esattamente 2018 allora occorre controllare il mese e, solo nel caso anch'esso non sia informativo (ossia uguale al mese corrente) passare al controllo del giorno.]

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

   - Scrivere un programma che legge le coordinate dei 3 vertici di un triangolo e le memorizza in una struct di tipo `Triangle`. Calcola poi area e perimetro del triangolo e memorizza le informazioni nei corrispondenti campi della struct. Infine verifica se il triangolo è o no un triangolo rettangolo e stampa un messaggio di output opportuno.
   - 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.

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

   Scrivere un programma che legge le informazioni relative a due variabili T1, T2 di tipo `Time`, ne verifica la correttezza e calcola il tempo trascorso tra i due orari, assumendo che si riferiscano allo stesso giorno.

5.3 Esercizi più avanzati

8. 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.

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

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

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

   [**SUGGERIMENTO:** Usate array di struct.]

Parte 6

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 *testare*, ovvero chiamare su argomenti noti per verificare che il risultato sia quello atteso. In questo programma avremo cura di fare solo chiamate passando argomenti corretti.
NOTA: Ritroverete in questa parte molti esercizi che avete già svolto senza usare funzioni. D'ora in avanti, tenete presente che alcune funzioni sviluppate in un esercizio possano essere utili allo svolgimento di un esercizio successivo. Cercate di riutilizzare (ossia di richiamare) le funzioni già scritte, quando è appropriato.

Cheatsheet

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

Dichiarazione:
Esempio

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

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

Definizione:
Esempio

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

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

Invocazione o chiamata:
Esempi:

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

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

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

Errori comuni:

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

Cheatsheet

Trattamento errori:

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

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

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

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

6.1 Esercizi di riscaldamento

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

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

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

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

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

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

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

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

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

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

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

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

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

```
// chiamare la funzione hello
// chiamare la funzione hellomany passandole come argomento il valore 5 (letterale)
// chiamare la funzione givemefive stampando su std::cout il valore restituito
// chiamare la funzione prossimo passandole come argomento il valore 4...
//                                        ... e stampando su std::cout il valore restituito
// chiamare la funzione somma passandole come argomenti i valori 2 e 3...
//                                        ... e stampando su std::cout il valore restituito
```

[**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.

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

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

   Scrivere un programma per testare la funzione secondo il seguente algoritmo

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

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

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

Scrivere un programma per testare la funzione `area`:

```
//  dichiarare due variabili b e h di tipo float
//  leggere b e h
//      dichiarare la variabile float a
//      chiamare la funzione area assegnando ad a il valore che restituisce
//      stampare l'area
```

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

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

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

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

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

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

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

```
*/
// restituire n
}
```

Scrivere un programma per testare la funzione `factorial`:

```
// stampare la stringa "Inserire un numero positivo: "
// dichiarare una variabile intera num
// leggere num
// stampare il risultato della chiamata di factorial su num seguito da " è il...
// ... fattoriale di " seguito da num
```

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

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

```
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
```

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

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

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

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

     Scrivere un programma per testare la funzione:

     ```
     // dichiarare una costante s1 di tipo string inizializzata con "Prima scelta"
     // dichiarare una costante s2 di tipo string inizializzata con "Seconda scelta"
     // dichiarare una costante s3 di tipo string inizializzata con "Terza scelta"
     ```

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

- 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
```

- 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
```

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
```

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

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

9. 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
```

6.2 Esercizi di base

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

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

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

    ```
    xxxxx
    xxxxxx
    xxxxxxx
    xxxxxxxx
    xxxxxxxxx
    ```

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

    [**SUGGERIMENTO:** usare la funzione `replicate`.]

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

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

    ```
    sssqqqqqqq
    ```

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

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

    che sulla diagonale ha 17 caratteri.
    [**SUGGERIMENTO:** 1)è più facile stampare il rombo con due cicli, il primo per le righe in cui il numero di asterischi cresce e il secondo per le righe in cui il numero di asterischi diminuisce.]

    [**SUGGERIMENTO:** 2) usate la funzione `replicate2_line`]

16. Scrivere una funzione con un argomento intero `n` che restituisce un booleano, `true` sse `n` è *palindromo*, ovvero se le sue cifre (in base 10) lette da destra a sinistra corrispondono alle cifre lette da sinistra a destra.
    [**SUGGERIMENTO:** usare la funzione `reverse`.]

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

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

18. 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.

6.3 Esercizi più avanzati

19. 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 $1634 = 1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256$.

20. Scrivere una funzione con un argomento intero `n` che restituisce il numero di zeri alla fine del fattoriale (in base 10) del suo argomento **senza calcolarne il fattoriale**. Ad esempio su 5 stampa 1 perché $5 = 12\mathbf{0}$, mentre su 11 stampa 2 perché $11! = 39\,916\,8\mathbf{00}$.

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

22. 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`.

    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.

23. 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.

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

    Scrivere una funzione che presi come argomenti un intero non negativo (la popolazione iniziale), la percentuale di nascite al netto delle morti come intero fra 0 e 100 e il numero di persone che si trasferiscono nella città al netto di quelle che l'abbandonano, restituisce un intero pari al numero di abitanti dopo un anno.

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

    [**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.]

25. Analogamente al punto precedente, scrivere una funzione che prende, oltre ai parametri della funzione al punto 24, anche un intero che rappresenta un numero di anni e restituisce la popolazione dopo quel numero di anni.

    [**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.]

26. 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).

27. 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`.

Argomenti di programmazione C++

Parte 7

Puntatori - senza allocazione dinamica di memoria

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

Cheatsheet

- Puntatore nullo `nullptr`
- Operatore di referenziazione `&`. Opera su una variabile (o altro valore sinistro), prendendone l'indirizzo e trasformandolo in un puntatore. Esempio: `&a;` = l'indirizzo di `a`.
- Definire una variabile di tipo "puntatore a un oggetto di tipo `T`":
  - senza inizializzazione: `T *p;`. Esempio: `int *p;`

    Meglio inizializzare le variabili, perché rischia di usarne il valore senza avergliene assegnato uno prima. Nel caso dei puntatori questo vuol dire accedere ad un'area di memoria a caso.
  - con inizializzazione: `T *p = espressione-di-tipo-puntatore>;`

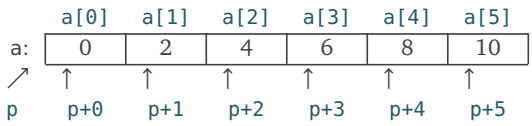
    Esempi: `char *p = nullptr; char *q = p;  float *p = &a;` dove `a` è una variabile di tipo `float`.

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

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

  Esempi: `*p = 2;` (scrive 2 nell'area puntata da `p`); `a = *p + 4` (legge il valore contenuto nella memoria puntata da `p`, le somma 4 e assegna il risultato ad `a`)
- Dimensione in memoria di qualcosa che ha tipo `T`: `sizeof T`
- Dimensione in memoria di una variabile `a`: `sizeof (a)`
- Aritmetica dei puntatori: se `T *p;` e `n` è un valore intero, allora `p+n` è l'indirizzo che si ottiene sommando `n` volte `sizeof (T)` all'indirizzo contenuto in `p`. Corrisponde a spostarsi, rispetto all'indirizzo puntato da `p`, di `n` elementi di tipo `T`.

  Esempio: `int a[6] = {0, 2, 4, 6, 8, 10}; int *p = a;` corrisponde a



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

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

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

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

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

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

7.1 Esercizi di riscaldamento

1. Scrivete un programma in cui usate i puntatori per accedere alle locazioni di variabili. Siete incoraggiati a migliorare i messaggi di stampa minimali proposti nell'esercizio in modo da capire più facilmente quali valori state stampando, ad esempio aggiungendo frasi come *indirizzo di s1 ==* oppure *valore di p ==*.

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

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

2. Riprendete la vostra implementazione della funzione `reverse` dell'esercizio 8 in sezione 4 e provate a modificare il programma di test utilizzando `source` per instanziare entrambi i parametri di `reverse`. Se avete implementato `reverse` nel modo ovvio, non otterrete il risultato atteso, perché copiando gli elementi dalla testa di `source` nella coda di `dest` in realtà state modificando la coda di `source` stesso. Per evitare questo problema, vogliamo premettere al corpo della funzione `reverse` il controllo che i due parametri corrispondano ad aree di memoria distinte e se invece coincidono sollevare un'eccezione.

   Nota. In questo specifico caso si potrebbe evitare il problema usando la stessa tecnica suggerita per l'esercizio 15 in sezione 4. Però, in situazioni più complesse potrebbe essere impossibile risolvere i problemi in caso di aliasing fra i parametri.

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

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

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

```
char* selectVar(char& a, char& b, char& c) {
// definire un puntatore di tipo char p inizializzato al puntatore nullo
// stampare i messaggi "Scegli fra queste variabili" e
// "potrai cambiare idea in seguito e sceglierne una diversa che preferisci"
// stampare il messaggio "Vuoi la prima (y/n)? contiene "
// stampare a;
// dichiarare una variabile answer di tipo char
// leggere answer
// se la risposta è 'y' o 'Y'
// assegnare a p l'indirizzo di a
// stampare il messaggio "Preferisci la seconda (y/n)? contiene "
```

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

Scrivere un programma di test

```
// dichiarare tre variabili di tipo char ch1, ch2, ch3...
// ...inizializzate con lettere fra loro diverse
// definire un puntatore di tipo char selected inizializzato con ...
// ... la chiamata di selectVar su ch1, ch2 e ch3
// confrontare selected con l'indirizzo di ch1 e se sono uguali
// stampare il messaggio "hai scelto ch1"
// confrontare selected con l'indirizzo di ch2 e se sono uguali
// stampare il messaggio "hai scelto ch2"
// confrontare selected con l'indirizzo di ch3 e se sono uguali
// stampare il messaggio "hai scelto ch3"
// stampare il messaggio "Inizialmente ch1==" seguito da ch1
// stampare il messaggio ", ch2==" seguito da ch2 e ", ch3==" seguito da ch3
// stampare un'andata a capo
// stampare il messaggio "ora cancello la variabile che hai scelto"
// assegnare uno spazio all'area di memoria puntata da selected
// stampare il messaggio "Ora ch1==" seguito da ch1, da ", ch2==" seguito da ch2 e...
// ...da ", ch3==" seguito da ch3 e un'andata a capo
```

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

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

Nel `main`

```
// dichiarare un array v di N interi
// dichiarare un puntatore a interi p inizializzato con v
// stampare il messaggio "Dimensione di v == " seguito ...
// ...dalla dimensione di v e andare a capo
// stampare il messaggio "v ha dimensione " seguito dalla dimensione di v diviso...
// ...la dimensione del suo primo elemento e andare a capo
// stampare il messaggio "Dimensione di p == " seguito dalla dimensione di p ...
// ... e andare a capo
// chiamare f su v
```

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

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

```
// dichiarare una costante N di tipo int inizializzata ad un valore...
// ... strettamente maggiore di 0
// dichiarare un array v di N interi
// dichiarare un puntatore a interi p inizializzato con v
/* iterare su i a partire da 0 e fino a N-1
        - leggere un valore intero memorizzandolo nella cella puntata da p
        - incrementate p
*/
// dichiarare una variabile sum di tipo float e inizializzarla a zero
// assegnare v a p (per ricominciare da capo a scorrere v)
/* iterare su i a partire da 0 e fino a N-1
        - sommare il contenuto della cella puntata da p a sum
*/
// stampare la divisione di sum per N
```

7.2 Esercizi di base

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

6. Modificare l'esercizio 2 introducendo funzioni per i frammenti di codice ripetuti:

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

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

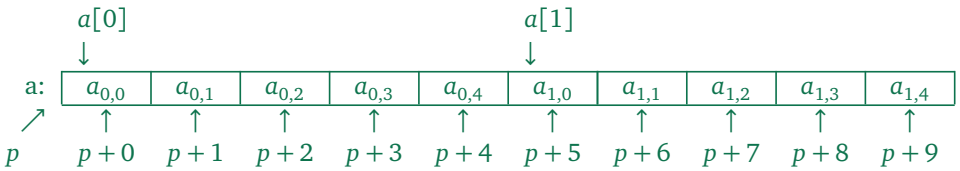
7. Scrivere una funzione che preso un `array v` di `N` elementi (dove `N` è una costante positiva a vostra scelta) di tipo `char` restituisce il numero di cifre, cioè caratteri nell'intervallo `['0','9']`, in `v`, usando l'aritmetica dei puntatori per scorrere `v`.

8. Migliorare le funzioni relative a ordinamenti e ricerche degli esercizi in sezione 4 usando l'aritmetica dei puntatori per scorrere gli `array`.

7.3 Esercizi più avanzati

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

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



   Quindi per scorrere tutti gli elementi di un vettore bidimensionale basta un unico puntatore incrementato sempre di uno anche per passare da una *riga* alla successiva] [**SUGGERIMENTO:** Per inizializzare un puntatore all'indirizzo del primo elemento di un `array` basta assegnargli l'`array` stesso. Nel caso bidimensionale, però questo vuol dire avere un puntatore al tipo `array` interno e non all'elemento della matrice. Per ottenere un puntore al tipo della singola cella, quindi, bisogna assegnare al puntatore il primo elemento dell'`array`.]

10. Scrivere una funzione `diagonal` che, ricevuta come argomento una matrice quadrata di caratteri, stampa gli elementi sulla diagonale usando l'aritmetica dei puntatori per visitare la matrice.

    [**SUGGERIMENTO:** Gli elementi sulla diagonale distano fra loro la lunghezza di una riga +1.]]

Parte 8

Puntatori - allocazione dinamica di memoria

In questa sezione vedremo l'uso dei puntatori per *allocare dinamicamente memoria* e lavorarci. Qui ai puntatori, anziché l'indirizzo in memoria di oggetti già esistenti, viene assegnato l'indirizzo di una nuova area di memoria ottenuta dall'area di sistema chiamata heap (*allocazione*), che al termine dell'uso andrà restituita al sistema stesso (*deallocazione*). È un modo per richiedere memoria solo nel momento in cui serve e nella quantità che serve.

Vedremo anche gli elementi minimi per lavorare con i file.

Cheatsheet

`int *a;`	dichiara una variabile di tipo puntatore a `int` (come nella parte precedente)
`a = new int;`	alloca la memoria necessaria a memorizzare un `int` **e allo stesso tempo** assegna l'indirizzo di tale memoria al puntatore `a` **Attenzione:** memoria allocata senza fare assegnazione = memoria perduta!
`float *fpunt = new float;`	Dichiarazione, allocazione e assegnazione in unica istruzione
`float *fpunt = new float[10];`	alloca la memoria necessaria a memorizzare 10 `float` (come un array di 10 elementi) Il puntatore `fpunt` punta al primo elemento del blocco di memoria allocato
`*fpunt`	accede alla memoria il cui indirizzo è contenuto in `fpunt` ("puntata da `fpunt` ") **RICORDARE:** Se `fpunt` è dichiarato come sopra, `*fpunt` è una espressione che ha tipo `float`
`*(fpunt+2)`	accede alla memoria il cui indirizzo è quello contenuto in `fpunt` incrementato di 2 × la dimensione di un `float`, ovvero `2*sizeof (float)`. Equivale a `fpunt[2]`
`delete p`	elimina puntatore allocato con `new`
`delete[] p`	elimina puntatore allocato con `new []` **ATTENZIONE! DOPO `delete`, IL PUNTATORE NON CAMBIA VALORE, MA L'INDIRIZZO CHE CONTIENE NON È PIÙ VALIDO!** Se deve essere azzerato, fatelo con `p = nullptr;` subito dopo

Se il tipo struttura `TS` è definito così...

```
struct {
  int a;
  int b; } TS;
```

⟶ ...allora:

`TS * ps`	Dichiara `ps` come puntatore a un oggetto di tipo `TS` (NON INIZIALIZZATO, valore imprevedibile)
`* ps`	un oggetto di tipo `TS`
`(* ps).a`	un oggetto di tipo `int`, il membro `a` dell'oggetto puntato da `ps`
`ps->a`	sinonimo di `(*ps).a` , più rapido da usare

Errori:

- Usare `*p` con `p` non allocato
- Usare `*(p+n)` con `p` allocato con meno di `n` elementi (stesso errore di usare array con indice eccessivo!)
- `delete p` con `p` già deallocato
- La sequenza di istruzioni `q = p; delete p; *q;` — Se `p` e `q` sono uguali, deallocando la memoria di uno anche l'altro diventa non più valido
- Fare il test `if (p == nullptr) ...` senza che nessuno abbia fatto esplicitamente `p=nullptr;`.
- Pensare che un puntatore dichiarato come `int p;` abbia valore iniziale `nullptr` (il valore è invece normalmente imprevedibile)

Cheatsheet	
File	`<ifstream>` (solo lettura) – `<ofstream>` (solo scrittura) – `<fstream>` (entrambi)
`std::ifstream f;`	dichiara una variabile di tipo file stream di input (lettura)
`std::ofstream f;`	dichiara una variabile di tipo file stream di output (scrittura)
`f.open("nomefilesudisco");`	apre file "nomefilesudisco" (in lettura o scrittura) e lo associa a `f`; d'ora in poi nel programma uso `f` per leggere/scrivere `nomefilesudisco`
`std::ofstream f("nomefilesudisco");`	dichiara e apre allo stesso tempo (inizializzazione)
`if( f.good() ) ...`	verifica se l'ultima lettura è andata a buon fine (no errors)
`if( f.eof() ) ...`	verifica se l'ultima lettura ha raggiunto la fine del file
`<<  >>`	Operatori per scrivere e leggere, rispettivamente
`f.close()`	Quando ho finito di usare il file

8.1 Esercizi di riscaldamento

1. Scrivere un programma che implementa il seguente algoritmo:

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

   Nota. Fate attenzione a non perdere il riferimento a `v` nell'inizializzarne il contenuto o nella stampa.

2. Implementare le funzioni necessarie a leggere e stampare un `array` dinamico di interi, ovvero un `array` la cui dimensione viene stabilita al momento dell'esecuzione, *run time* (invece che al momento della compilazione, *compile time*, sulla base di un valore costante e immutabile). Per rappresentare gli `array` dinamici di interi utilizzeremo il seguente tipo:

   `dynamic_array`:

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

   I dati sono immagazzinati in un blocco di memoria allocato dinamicamente ed assegnato a un puntatore. Ricordate che, come visto nella sezione 7, quando si ha un puntatore che punta a una zona di memoria che può contenere `N` elementi lo si può utilizzare con la stessa sintassi di un `array` per accedere ai suoi elementi, mediante l'indice fra parentesi quadre. Pertanto, l'accesso in lettura e in scrittura (assegnazione) è uguale a quanto visto in precedenza per gli `array`.

   La differenza sta nel fatto che l'array è **immutabile**, mentre il puntatore può essere cambiato (associato ad altra area di memoria usando `new` e l'operatore di assegnazione, o manipolato usando l'aritmetica dei puntatori: `p++`, o `p = p + 4`, ecc).

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

```
dynamic_array new_d_array(int size) {
// se size <=0, porre size a 0
// dichiarare una variabile a di tipo dynamic_array
// assegnare size al campo size di a
// allocare size interi, assegnando la memoria al campo store di a
// restituire a
```

```
}
```

(b) Scrivere una funzione per la lettura interattiva (da tastiera) di un `array` dinamico di interi

```
void read_d_array(const dynamic_array& a)
{
/* iterare a.size volte (usando un indice i)...
        // stampare "inserisci un valore"
        // leggere un valore nell'i-esimo elemento del campo store di a...
        // ...usando la notazione con l'operatore ``parentesi quadre'' per accedervi
*/
}
```

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

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

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

8.2 Esercizi di base

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

3. Scrivere la funzione `delete_d_array` che preso il riferimento ad un array dinamico `a` lo *svuota*, ovvero se `a.size` è positivo rilascia lo spazio allocato per `a.store` e assegna zero ad `a.size`.

4. Riscrivete la funzione precedente aggiungendo un controllo: se `a.size` è zero, solleva un'eccezione.

   Nota. Nel programma di test potete facilmente verificare se due chiamate di `delete_d_array` successive su uno stesso `array` dinamico `d` (inizializzato con `read_d_array`) sollevano un'eccezione. Non avete invece modo di verificare da programma che lo spazio sia effettivamente stato rilasciato. Per farlo potete però usare programmi che controllano che non ci siano *memory leak*, come ad esempio valgrind (http://www.valgrind.org/) che trovate già installato sulle macchine di laboratorio.

5. Scrivere la funzione `set` che, ricevuti come argomenti un array dinamico `a`, un indice (intero) `index` ed un valore `value`, assegna `value` all'elemento con indice `index` di `a.store`.

   La funzione `set` restituisce `value`.

   Solleva una eccezione se `index` non è un valore corretto rispetto a `a.size` (l'indice è *out of range* se è minore di zero o maggiore o uguale alla dimensione).

6. Scrivere la funzione `get` che, ricevuti come argomenti un array dinamico `a` e un indice (intero) `index`, restituisce il valore dell'elemento con indice `index` di `a.store`.

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

   NOTA: L'argomento `a` va dichiarato con il tipo `const dynamic_array a` per rendere chiaro che non verrà modificato dalla funzione.

7. Ispirandosi alla funzione `read_d_array()`, scrivere una funzione `fread_d_array()` per la lettura da file di un `array` dinamico di interi. La funzione deve ricevere in aggiunta un argomento di tipo "riferimento a `ifstream`" (è obbligatorio specificare il *riferimento* al tipo quando gli argomenti sono di un tipo stream). Deve poi comportarsi come `read_d_array()`.

   Il prototipo deve quindi essere: `void read_d_array(const dynamic_array& a, ifstream& f)`

8. Scrivere una funzione `reverse_file()` che trascrive un file in ordine inverso. La funzione deve ricevere due argomenti di tipo `std::string&`, che rappresentano i nomi di due file. Deve aprire il primo file in lettura (quindi deve esistere, sollevare eccezione se non esiste) e il secondo in scrittura (se non esiste viene creato, quindi non occorre controllare). Poi legge il contenuto del primo file, che è una sequenza di numeri interi, e scrive lo stesso contenuto, in ordine inverso, sul secondo file.

   Il prototipo deve quindi essere: `void reverse_file(const std::string& nomefilein, const std::string& nomefileout)`.

8.3 Esercizi più avanzati

In questa sezione creeremo un tipo di oggetto simile ad un array ma flessibile, chiamato `myvector`.

Un oggetto di tipo `myvector` è una struct che contiene tre membri:

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

Le funzioni richieste sul tipo `myvector` sono le seguenti.

NOTA: Nella specifica delle funzioni richieste, spesso viene indicato che un parametro di tipo `myvector` è già inizializzato, o viceversa non lo è ancora.
La correttezza dei myvector, quindi, è stabilita solo da un *contratto* fra voi che implementate le funzioni e chi le chiamerà. Questa indicazione è qualcosa di cui ci dobbiamo fidare. Non possiamo verificare che venga rispettato, perché non potete controllare se l'area puntata da `store` sia stata allocata, né che sia di dimensione pari a `capacity`. Possiamo solo fare le seguenti verifiche: (1) che `size` sia minore o uguale a `capacity`; (2) che `size` sia non negativa; (3) che `capacity` sia positiva. Se chiamiamo funzioni passando come argomenti `myvector` che non sono stati inizializzati correttamente, quindi, potranno verificarsi errori impredicibili, fra cui *segmentation fault* se la funzione cerca di accedere ad un areea di memoria che non è stata riservata per il `myvector`. Nel caso peggiore, gli errori non si verificheranno fino al momento in cui verrà collaudato il nostro software (per esempio per dare il voto all'esame...)

9. Scrivere la funzione `void create(myvector& v, int capacity);` che, ricevendo come argomento il riferimento a un `myvector v` non ancora inizializzato, fa l'allocazione di `v.store` con lunghezza `capacity`, assegna `v.capacity = capacity`, e assegna `v.size = 0`.

   Sui `myvector` si possono definire due funzioni che operano alla coda (*back*), o estremo, del vettore: la funzione push aggiunge un elemento e la funzione pop lo legge e rimuove. Negli esercizi seguenti creiamo le funzioni `push_back` e `pop_back`.

10. Scrivere la funzione `push_back` che, ricevendo come argomento il riferimento a un `myvector v` già inizializzato, e un valore `x`, opera come segue: se `v.size` è minore di `v.capacity` inserisce `x` in `v.store` all'indice `v.size` e incrementa di uno `v.size`. Se `v` è già completamente pieno (cioè `v.size == v.capacity`), la funzione solleva un'eccezione.

11. Scrivere la funzione `pop_back` che dato per riferimento un `myvector v` inizializzato se `v.size` è positivo restituisce il valore memorizzato in `v.store` all'indice `v.size-1` e decrementa di uno `v.size`. Se `v` è vuoto (cioè `0 == v.size`), la funzione solleva un'eccezione.

    Lettura e scrittura di `myvector` si possono fare operando sul membro `store` con l'operatore di indirizzamento `[ ]`. In sostituzione, negli esercizi seguenti creiamo delle funzioni `set` e `get` che fanno le stesse operazioni, aggiungendo però dei controlli di correttezza sugli indici.

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

13. Scrivere la funzione `int get(const myvector& v, int index);` che restituisce il valore presente in `v.store` all'indice `index`. Solleva una eccezione se `index` non è un valore corretto, ovvero se non è compreso fra `0` e `v.size`.

    Negli esercizi seguenti sfruttiamo il fatto che `myvector` usa l'allocazione dinamica per distruggere (`destroy`) oppure ridimensionare (`resize`) i myvector.

14. Scrivere la funzione `void destroy(myvector& v);` che dealloca `v.store` e pone a zero sia `v.size` che `v.capacity`.

15. Scrivere la funzione `resize` che dato per riferimento un `myvector v` già inizializzato e un intero `new_capacity` strettamente positivo modifica la capacità del vettore preservando i valori contenuti per quanto possibile.

    La funzione dovrà quindi

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

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

    Negli esercizi seguenti aggiungiamo ulteriori controlli di correttezza.

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

    [**SUGGERIMENTO:** utilizzate la funzione `push_back` in un blocco try, catturate l'eccezione che sollevate in caso di mancanza di spazio e nel corrispondente catch chiamate in ordine la funzione `resize` per raddoppiare la capacità e poi nuovamente la funzione `push_back` per inserire l'elemento che a questo punto vi troverà posto.]

17. Scrivere la funzione `bool looks_consistent(const myvector& v)` che restituisce vero se `v.store` non è nullo, `0<=v.size`, `v.size<=v.capacity` e `0<v.capacity`.

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

18. Modificare tutte le funzioni precedenti **ad eccezione di `create`** in modo che come prima cosa verifichino se il loro argomento di tipo `myvector` è evidentemente scorretto (cioè se `looks_consistent(v)` è falso) e in tal caso sollevino un'eccezione.

Parte 9

Vector

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

Cheatsheet

`std::vector<T> V`	Dichiara la variabile `V` di tipo `std::vector` con elementi di tipo `T` **qualsiasi** (anche non elementare)
`std::vector<T> V(A)`	Crea un `std::vector V` di `T` ed inizializza i suoi elementi con un array `A` (di `T`) pre-esistente
`std::vector<T> W(V)`	Crea `W` e lo inizializza con il contenuto di un altro `std::vector<T> V` ("costruttore di copia")
`std::vector<T>v(N)`	Crea un `std::vector` di `N` elementi inizializzati a un valore "nullo" predefinito che dipende dal tipo T (es. 0 se `T=int`, 0.0 se `T=float`, stinga vuota se `T=std::string`)
`std::vector<T> V(N, val)`	Crea un `std::vector V` di `N` elementi inizializzati ad un valore `val`
`V.size()`	Lunghezza di `V` (numero elementi)
`V.capacity()`	Spazio occupato da `V`, include spazio non ancora utilizzato
`V.max_size()`	Dimensione massima possibile
`V[i]`	Accede all'elemento `i`-esimo (segmentation fault se `i>=V.size()` o `i<0`)
`V.at(i)`	Accede all'elemento `i`-esimo (errore trattabile se `i>=V.size()` o `i<0`)
`V.back()`	Accede all'ultimo elemento (possibile errore se `V` è vuoto)
`W=V`	Copia il contenuto da `V` a `W` (che deve esistere)
	Casi in cui il tipo-base `T` non è un tipo elementare
Se `T` tipo struct	Esempio: `struct T {int a, float b};` → `V` è un vector di `T`; `V[i]` è un `T`; `V[i].a` è un `int`; `V[i].b` è un `float`
Se tipo `T` = `vector` di...	Esempio: `vector< vector<int> > V` → `V` è un `vector` di `vector` di `int`; `V[i]` è un `vector` di `int`; `V[i][j]` è un `int`
`V.push_back(val)`	Aggiunge `val` in coda (ultima posizione) incrementando automaticamente la dimensione
`V.pop_back()`	Elimina l'ultimo elemento decrementando automaticamente la dimensione
`V.resize(N)`	Ridimensiona a nuova lunghezza `N`
`V.clear()`	Svuota (porta a lunghezza 0)
`using namespace std;`	in testa al file per poter omettere il prefisso `std::`
	Per abbreviare le dichiarazioni con tipi complicati
`typedef tipo nuovonome`	definisce un nuovo nome per un tipo Esempio: `typedef vector< int > vettoreint` Esempio: `typedef vector< vector<int> > matriceint` `nuovonome` diventa un nome di tipo. Esempio di dichiarazione: `matriceint M;`

9.1 Esercizi di riscaldamento

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

```
void readVector(std::vector<int>& v) {
// Stampare "Inserisci la dimensione della sequenza: "
// Dichiarare una variabile intera N
// Leggere N
```

```
/* iterare N volte
        - leggere un valore intero
        - memorizzare il valore letto in v
*/
}
```

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

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

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

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

```
12
5
10
8
y
```

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

```
void SelectionSort_vector(std::vector<int>& v) {
// dichiarare una variabile int greatestIndex
/* iterare sul std::vector dalla prima all'ultima posizione
        - memorizzare in greatestIndex la posizione corrente (sia i)
        - *//* iterare sul std::vector dalla posizione successiva alla corrente ...
                ... (i+1) fino all'ultimo elemento
                -- se il valore alla pos corrente (j) e' < del valore...
                ... alla pos greatestIndex
                        --- memorizzare j in greatestIndex
          *//*
        - scambiare il valore alla posizione i con quello alla pos greatestIndex
*/
}
```

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

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

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

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

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

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

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

Quali sono le differenze fra le due implementazioni?

9.2 Esercizi di base

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

5. Scrivere una funzione `std::vector<int> cat(std::vector<int> v1, std::vector<int> v2)` che restituisce un `std::vector<int>` contenente il contenuto di `v1` seguito dal contenuto di `v2`.
   Scrivere un programma di test che legge interi strettamente positivi in due `std::vector<int> first` e `second`, Quindi chiama `cat` per concatenare i due vettori e memorizza il risultato in `std::vector<int> total`, stampando infine i tre elenchi contenuti in `first`, `second` e `total` .

6. Scrivere una funzione `std::vector<int> insert(std::vector v, int i, int val)` che aggiunge a `v` in posizione `i` il valore di `val`. La lunghezza di `v` deve essere incrementata di 1 e tutto l'eventuale contenuto di `v`, dalla posizione `i` (compresa) fino alla fine, deve essere spostato in avanti di una posizione. **NOTA:** siccome è previsto che il vettore si incrementi di dimensione, le posizioni di inserimento valide sono tutte quelle esistenti (da 0 a `v.size()-1`) e anche una oltre l'ultima (la posizione `v.size()`), quindi `i`∈ [`0`,`v.size()`]. Se `i` non è compreso in questo intervallo sollevare una eccezione `int`.
   [**SUGGERIMENTO:** Inserire in coda a `v` il suo ultimo elemento (se esiste), spostare in avanti di una posizione gli elementi di v a partire dalla posizione i in avanti (copiandoli a partire dal fondo in modo da non sovrascriverli) e assegnare il valore da inserire nella posizione i.]
   Scrivere un programma di test che dichiara un `std::vector<int>` e fa il test della funzione `insert` nei seguenti casi:

   (a) Inserimento in `v` vuoto

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

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

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

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

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

   [**SUGGERIMENTO:** Per scrivere i programmi intermedi per testare le funzioni prodotte fino ad un certo punto, potete modificare sempre lo stesso main, commentando i pezzi che non vi servono più perché avete testato in maniera soddisfacente la funzione a cui fanno riferimento; però non cancellate, perché se poi dovete fare modifiche alle funzioni precedenti o vi accorgete di un caso che non avete provato, così vi trovate ancora il codice di test pronto, basta togliere i commenti.
   Per quanto riguarda l'input dei dati su cui provare le chiamate di funzione, potete fare lettura da input (più facile, ma ci mettete più tempo a far girare il programma) oppure da file (richiede più sforzo la prima volta, ma vi permette di ripetere i test molto rapidamente e senza ulteriore fatica).]

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

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

   (c) Scrivere la funzione `void add(PhoneBook& B, string surname, string name, int phoneNumber)` per creare un nuovo contatto `C` e aggiungerlo in coda alla rubrica `B`. (Definire `C` nella funzione e aggiungerlo con `push_back`).

   (d) Scrivere la funzione `void print(const PhoneBook& B)` per stampare il contenuto della rubrica `B`. (Per ogni contatto, bisogna stampare ogni membro separatamente.)

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

   (f) Scrivere una funzione `int FindPos(const PhoneBook& r, string S)` che, **utilizzando la ricerca binaria**, abbia il seguente comportamento:

   - Se nella rubrica **esiste** un contatto `C` il cui campo `C.Surname` è uguale all'argomento `S`, allora restituisca l'indice di tale contatto nella rubrica (ossia nel vettore)
   - Se nella rubrica **non esiste** un contatto `C` il cui campo `C.Surname` è uguale all'argomento `S`, allora restituisca l'indice del contatto che sarebbe quello immediatamente precedente in ordine alfabetico.

   [**SUGGERIMENTO:** Ricordate che la ricerca binaria assume che il vettore sia ordinato] [**SUGGERIMENTO:** Per ordinare un `vector` di `struct`, occorre confrontare non gli elementi `i` e `j` del `vector` ma il campo `Surnaname` e poi eventualmente il campo `Name` degli elementi `i` e `j`. Vedi cheatsheet per come riferirsi a un campo di un elemento di un vector di struct.]

   (g) Scrivere una funzione `void Shift_PhoneBook(PhoneBook& B, int pos)` che incrementa di un elemento la dimensione del vettore `B` e poi sposta a destra di un elemento tutti gli elementi a partire dalla posizione pos+1.

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

8. **Tavola pitagorica**: Creare una tabella di dimensioni 10×10, utilizzando vector di vector di interi, e riempirla con una tavola pitagorica, ovvero l'elemento di coordinate `i` e `j` deve contenere il valore `i * j`.

   Stampare la tabella in modo ordinato usando la funzione `setw()` (includere il modulo di libreria `iomanip`).

   [**SUGGERIMENTO:** Occorre stampare per prima cosa `setw(n)` su `cout` per fare in modo che lo spazio occupato da ciascuna stampa sia almeno di `n` caratteri (`n` = non meno di 3, nel nostro caso, per numeri che possono arrivare fino a 100).]

9.3 Esercizi più avanzati

9. Ancora sul **Crivello di Eratostene**: scrivere la funzione `std::vector<int> primes(int n)` che dato un intero positivo `n` restituisce un `std::vector<int>` che contiene tutti e soli i numeri primi compresi tra `2` e `n`, che quindi deve essere un vettore vuoto se `n<2`. Partire dal codice della funzione `isprime` (Parte 4, esercizio 9).

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

Parte 10

Liste collegate

In questa parte ci eserciteremo a usare liste collegate (linked lists). Si tratta di un modo per realizzare sequenze di oggetti, ed è una alternativa ai vector. La scelta tra le opzioni possibili si fa valutando quali operazioni prevedo di effettuare più di frequente: accesso immediato a un elemento qualunque → vector; inserimento, spostamento o eliminazione di elementi in mezzo → linked list. Per ogni funzione, anche dove non indicato, scrivete un test nel programma principale, chiamando la funzione con valori opportuni e stampando il risultato.

Esiste una classe `std::forward_list` in C++ che rappresenta una lista semplice (con collegamento solo in avanti = forward, al prossimo elemento). **Non** la useremo, per poter imparare la tecnica di programmazione necessaria a realizzare questa struttura dati.

Cheatsheet

Lista collegata semplice:

```
typedef qualchetipo TIPO; // tipo di info
struct cell {             // tipo elemento di lista
  TIPO info;
  cell * next;
};
typedef cell * list;     // tipo lista

list l = nullptr;        // variabile di tipo lista
```

- Ogni elemento contiene puntatore al prossimo elemento.
- Una lista è rappresentata da un puntatore al primo elemento (testa)
- Una lista vuota (puntatore di testa nullo) è una lista valida.
- Per segnalare che la lista è finita, il membro `next` dell'ultimo elemento vale `nullptr`.
- Promemoria: se `item` è un puntatore a struct (`cell *`), scrivere `item->cont` equivale a scrivere `(*item).cont`.

Avvertenze:

- Inizializzare sempre il puntatore di testa a `nullptr`
- Ogni volta che si crea un nuovo elemento, inizializzare sempre il membro `next` a `nullptr`; l'operatore `new` da solo non lo garantisce
- Evitare di sovrascrivere il puntatore a inizio lista
- Evitare che il puntatore a inizio lista "muoia" perché è terminato il blocco di codice in cui è definito (corpo di una funzione, corpo di un ciclo)
- Per aggiungere un nodo, prima crearlo assegnandolo a variabile ausiliaria `cell * temp = new cell;`
- Usare un **cursore** = puntatore a elemento corrente, per scorrere la lista senza perdere il valore del puntatore di testa
- Considerare i casi-limite negli algoritmi (lista vuota, primo elemento, ultimo elemento) oltre al caso generale (generico elemento interno)

Cheatsheet

File header `hello.h`

```
void hello();
// dichiarazione di tipi e struct
// #define
```

File libreria `hello.cpp`

```
#include <iostream>
#include "hello.h"
void hello()
{
  std::cout <<''Hello, world!\n";
}
```

File principale `main.cpp`

```
#include "hello.h"
int main()
{
  hello();
}
```

10.1 Esercizi di riscaldamento

Scrivere un programma `liste.cpp` in cui si definisce un tipo "lista di double" (vedi cheatsheet, con `typedef double TIPO;`). Quindi scrivere le funzioni seguenti, e un programma principale che ne verifichi il funzionamento, stampando opportuni messaggi.

1. `cell * newcell(const TIPO cont)`

   Creazione di un puntatore a una nuova cella con contenuto `cont` e corretta inizializzazione di `next`:

   ```
   // Dichiarare una variabile di tipo puntatore a cell, allocandola immediatamente con l'operatore new
   // Assegnare cont al membro info della variabile
   // Assegnare nullptr al membro next della variabile
   // restituire la variabile
   ```

2. `list push_front(list & mylist, const TIPO cont)`

   Aggiunta di un elemento in testa alla lista, con contenuto `cont`:

   ```
   // Dichiarare una variabile temp di tipo puntatore a cell, allocandola immediatamente con la funzione newcell(cont)
   // assegnare mylist al membro next di temp
   // assegnare temp a mylist
   // restituire mylist
   ```

3. `list push_back(list & mylist, const TIPO cont)`

   Aggiunta di un elemento in coda alla lista, con contenuto `cont`:

   ```
   // dichiarare una variabile cur di tipo puntatore a cella e inizializzarla a mylist
   // dichiarare una variabile temp di tipo puntatore a cell, allocandola immediatamente con la funzione newcell(cont)
   // se mylist e' nullptr, assegnare temp a mylist
   // altrimenti:
   //    finche' il membro next di cur non e' nullptr
   //       assegnare a cur il valore del suo membro next
   //    assegnare temp all'elemento next di cur
   // restituire mylist
   ```

4. `list insert_after(list mylist, TIPO cont, int n)`

   Aggiunta di un elemento con contenuto `cont` alla lista in posizione immediatamente successiva all'elemento numero `n`, oppure in coda se `n` maggiore della lunghezza della lista:

   ```
   // dichiarare una variabile cur di tipo puntatore a cella e inizializzarla a mylist
   // dichiarare una variabile temp di tipo puntatore a cell, allocandola immediatamente con la funzione newcell(cont)
   // se cur e' nullptr, assegnare temp a mylist
   // altrimenti:
   //     finche' il membro next di cur non e' nullptr e n non e' zero
   //         assegnare a cur il valore del suo membro next
   //         decrementare n
   //     assegnare l'elemento next di cur all'elemento next di temp
   //     assegnare temp all'elemento next di cur
   // restituire mylist
   ```

5. `list clear(list mylist)`

   Eliminazione di tutti gli elementi da una lista.

   ```
   // se mylist non e' nullptr:
   //    assegnare al membro next di mylist il risultato della chiamata clear(mylist->next)
   //    deallocare (delete) mylist
   //    assegnare nullptr a mylist
   // restituire mylist
   ```

10.2 Esercizi di base

6. Aggiungere al programma creato precedentemente la funzione `void print(list mylist)` che stampa una lista (stampare uno spazio dopo ogni elemento).

   [**SUGGERIMENTO:** Stampare una parentesi quadra aperta prima e una chiusa dopo, per poter visualizzare anche una lista vuota. Stampare un a capo alla fine.]

7. Scrivere un programma diviso in tre file:

   - un file `hello.cpp` contenente la funzione `void hello()` che stampa `Hello, world!`
   - un file `hello.h` contenente il prototipo della funzione
   - un file `main.cpp` contenente un programma principale che chiama la funzione `hello()`

8. Dividere il programma `liste.cpp` scritto sopra in:

   - un file `list.cpp` contenente le definizioni delle funzioni;
   - un file `list.h` contenente le dichiarazioni dei tipi e i prototipi delle funzioni;
   - un file `testlist.cpp` contenente il programma principale di prova

   I file `list.h` e `list.cpp` costituiscono la "libreria list".

9. Aggiungere alla libreria la funzione `int length(list mylist)` che restituisce il numero di elementi di `mylist`, zero se vuota.

   NOTA: La funzione deve scorrere la lista e contare gli elementi, quindi impiega un tempo proporzionale al loro numero. Quale modifica sarà necessaria alla struct `cell` e alle altre funzioni se vogliamo ottenere la stessa informazione in un tempo indipendente dal numero di elementi?

10. Aggiungere alla libreria la funzione `list remove(list & mylist, int n)` che elimina l'elemento in posizione `n` se presente (ovvero se la lista ha almeno `n+1` elementi), e restituisce la lista aggiornata.

    NOTA: Usiamo la convenzione degli array e dei vector per cui si inizia a contare gli elementi da zero, quindi l'ultimo elemento di una lista lunga `n+1` elementi è l'elemento numero `n`

11. Aggiungere alla libreria la funzione `list remove_cont(list & mylist, TIPO cont)` che elimina tutti gli elementi il cui membro `info` contiene `cont`, se ce ne sono, e restituisce la lista aggiornata.

12. Aggiungere alla libreria la funzione `bool is_present(list mylist, TIPO cont)` che restituisce `true` se il valore `cont` è presente nella lista `mylist`, `false` altrimenti.

13. Aggiungere alla libreria la funzione `cell * find_first(list & mylist, TIPO cont)` che cerca il primo elemento il cui membro `info` contiene `cont` e restituisce il puntatore a tale elemento; `nullptr` se non trovato.

14. Aggiungere alla libreria la funzione `cell * at(list mylist, int n)` che restituisce l'elemento numero `n` della lista `mylist`, oppure `nullptr` se `n` è maggiore del numero di elementi presenti o la lista è vuota.

15. Aggiungere alla libreria la funzione `list ordered_insert(list & mylist, TIPO cont)` che inserisce un nuovo elemento, con contenuto `cont`, in posizione ordinata e restituisce la lista aggiornata.

10.3 Esercizi più avanzati

16. Aggiungere alla libreria la funzione `list reverse(list & mylist)` che restituisce la lista `mylist` in ordine invertito.

17. Aggiungere alla libreria la funzione `list unique(list & mylist)` che elimina dalla lista tutti gli elementi contigui ripetuti (es `[ a b b c a a ]` ⟶ `[a b c a ]`), e restituisce la lista aggiornata.

18. Aggiungere alla libreria la funzione `list merge(list mylist1, list mylist2)` che assume che le liste `mylist1` e `mylist2` siano ordinate, e crea una terza lista contenente il contenuto delle due liste, sempre ordinato. Restituisce la terza lista.

    [**SUGGERIMENTO:** "Assume che siano ordinate" significa che **non va fatto alcun controllo**.]

    NOTA: La soluzione più semplice è quella che usa `ordered_insert()`. È anche quella che, a parità di numero totale di elementi, fa meno operazioni? Scrivere un algoritmo che fa tante iterazioni quanto la somma delle lunghezze di `mylist1` e `mylist2`.

19. Aggiungere alla libreria la funzione `cell * nth_from_last(list mylist, n)` che restituisce un puntatore all'elemento che dista `n` posizioni dall'ultimo, oppure `nullptr` se `n` è maggiore del numero di elementi presenti o la lista è vuota.

    [**SUGGERIMENTO:** Dopo avere scorso `n` elementi, inizializzare un cursore che punta alla testa, e da quel momento in poi fare scorrere in avanti sia il cursore principale che questo secondo cursore.]

20. Aggiungere alla libreria la funzione `list sort(list & mylist)` che restituisce la lista `mylist` ordinata.

Appendice

Appendice A

Cheatsheet per lavorare su Linux

Convenzioni grafiche:

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

A.1 Comandi bash

Cheatsheet

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

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

Comandi

`cd` ⟨`cartella`⟩	cambia directory (cartella) di lavoro
`pwd`	print working directory, mostra cartella corrente
`rm [-i]` ⟨`file`⟩ `[file2 file3 ...]`	cancella file (con `-i`: chiedi conferma)
`mkdir` ⟨`cartella`⟩	crea cartella
`rmdir` ⟨`cartella`⟩	cancella cartella se vuota
`rm -r` ⟨`cartella`⟩	cancella cartella e il suo contenuto
`rm -r /*`	cancella tutto il filesystem (e non c'è undelete!)
`edit` ⟨`file`⟩	chiama text editor predefinito e apre ⟨`file`⟩

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

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

Controllo I/O ed esecuzione

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

A.2 Editing dei file sorgente

Cheatsheet

Qualunque text editor va bene

Visualizzare numeri di linea:

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

A.3 Compilazione

Cheatsheet

Usi comuni del comando `g++`

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

Opzioni utili di `g++`

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

A.4 Caccia agli errori

Cheatsheet

Procedure da seguire per il debugging

1 Errore di sintassi indicato dal compilatore

- Leggi i messaggi del compilatore su errori e suggerimenti.
- Correggi il primo errore indicato prima di procedere con il secondo
- Vai alle righe indicate nel messaggio d'errore
- Leggi attentamente il messaggio d'errore per capire la natura del problema

2 Il programma compila, ma comportamento errato

- Applicare indentazione corretta e verificare il codice
- Parentesi nel posto giusto?
- Il ciclo fa tutte e sole le iterazioni che deve fare?
- Per `if`, `for`, `while`: se il corpo è un blocco di codice (più istruzioni), ho ricordato di mettere le graffe? Ho per caso messo un punto e virgola dopo la parentesi tonda chiusa?
- Mettere istruzioni di stampa su `std::cerr` in punti appropriati per vedere come evolve il valore delle variabili