Programmazione

Trattamento delle Informazioni

- Informatica = studio sistematico dei *processi* che servono al trattamento delle informazioni o più in generale della definizione della soluzione di problemi assegnati
  - analisi dettagliata di ciò che serve al trattamento dell'informazione
  - progetto di una soluzione applicabile alla generazione di informazioni prodotte da altre informazioni
  - correttezza e efficienza della soluzione pensata

Informatica e studio di Algoritmi

- “algoritmo”
  - la sequenza precisa di operazioni il cui svolgimento sequenziale è necessario per la soluzione di un problema assegnato

- Informatica → studio sistematico degli algoritmi
  - Il calcolatore è tra tutti gli esecutori di algoritmi (compreso l’uomo) quello che si mostra più potente degli altri e con una potenza tale da permettere di gestire quantità di informazioni altrimenti non trattabili

Algoritmo ed Esecutore

- Algoritmo
  - un testo che prescrive un insieme di operazioni o azioni eseguendo le quali è possibile risolvere il problema assegnato
  - Se si indica con istruzione la prescrizione di una singola operazione, allora l'algoritmo è un insieme di istruzioni da svolgere secondo un ordine prefissato
- Esecutore
  - l'uomo o la macchina in grado di risolvere il problema eseguendo l'algoritmo
  - Se un algoritmo è un insieme di istruzioni da eseguire secondo un ordine prefissato, allora l’esecutore non solo deve comprendere le singole istruzioni ma deve essere anche capace di eseguirle

Cos’è un computer

- Un computer è un apparecchio elettronico
  - strutturalmente non ha niente di diverso da un televisore, uno stereo, un telefono cellulare o una calcolatrice elettronica
- progettato per eseguire autonomamente e velocemente attività diverse
  - Come tutte le macchine, non ha nessuna capacità decisionale o discrezionale, ma si limita a compiere determinate azioni *secondo procedure prestabilite*
- Il computer è una macchina che in maniera automatica esegue operazioni “*elementari*” ad altissima velocità
  - L'altissima velocità di elaborazione (ad es. milioni di istruzioni per secondo) fa sì che operazioni complicate, espresse mediante un gran numero di operazioni semplici, siano eseguite in tempi ragionevoli per l'ambiente esterno

Algoritmo e Programma

- Informalmente, un *algoritmo* è la descrizione di un lavoro da far svolgere ad un *esecutore*
  - Se si vuole usare un computer bisogna non solo progettare preliminarmente un algoritmo, ma anche provvedere a comunicarglielo in modo che gli risulti comprensibile
- La descrizione di un algoritmo in un linguaggio comprensibile ad un calcolatore è detto Programma
- Il linguaggio nel quale viene scritto il programma è detto linguaggio di programmazione

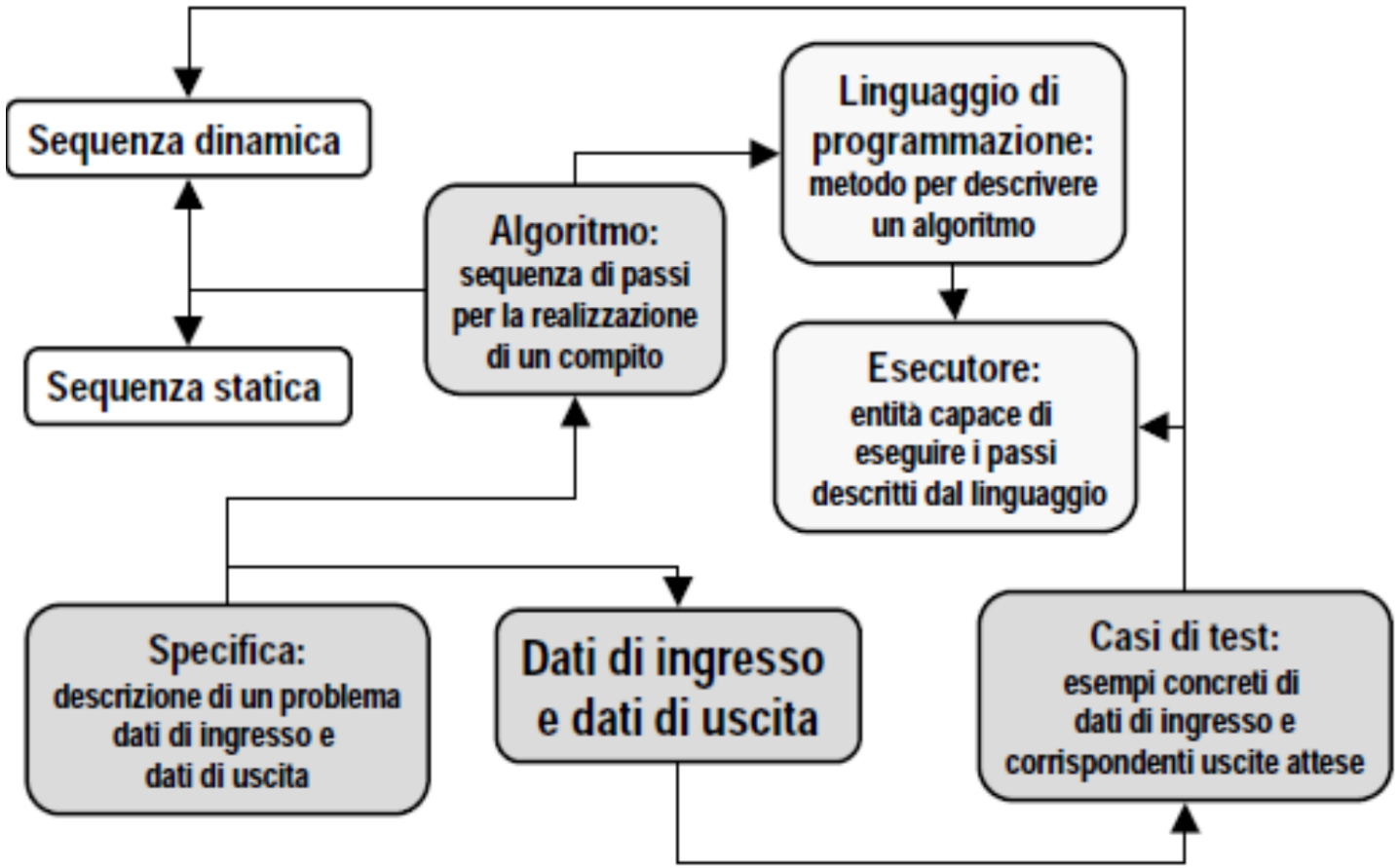
Programma

- La specifica è la descrizione in linguaggio naturale del problema da risolvere
- Informazioni di ingresso
- Informazioni di uscita
- Esempi:
  - Somma di due numeri naturali di una cifra
    - Input: i due numeri; output: la somma e il riporto
  - Somma di due numeri naturali in colonna
    - Input:i due numeri naturali; output: la somma

Processo e Processore

- Si definisce *processo* il lavoro svolto eseguendo l'algoritmo, e *processore* il suo esecutore
  - Il processo non è altro che l’elenco delle azioni effettivamente svolte come si susseguono nel tempo
  - Sequenza statica e sequenza dinamica

Riepilogo


Sequenza dinamica
Sequenza statica
Algoritmo:
sequenza di passi
per la realizzazione
di un compito
Linguaggio di
programmazione:
metodo per descrivere
un algoritmo
Esecutore:
entità capace di
eseguire i passi
descritti dal linguaggio
Specifica:
descrizione di un problema
dati di ingresso e
dati di uscita
Dati di ingresso
e dati di uscita
Casi di test:
esempi concreti di
dati di ingresso e
corrispondenti uscite attese

Il primo programma in C++

```
// Il primo programma in C++
#include <iostream>

using namespace std;

int main ()
{
    cout << "Hello, world!" << endl;
}
```

Dettagli

- Direttive per il precompilatore
  - `#include <iostream>`
- Simboli speciali
  - `Operatori e delimitatori`
  - `{ } ; <<`
- Parole chiave
  - `using return`
- Commenti
  - `// /* */`
- Identificatori
  - `main cout system`
- Stringhe Costanti
  - `“Hello, World!” “pause”`

Esercizio

- Modificare il programma per fare in modo che stampi a video quanto segue

Ciao sono io: il tuo Personal Computer.

Che desidera imparare tante cose nuove.

Devi solo progettare cosa devo fare.

Inventando nuovi programmi.

Forza datti da fare. Ci divertiremo.

Soluzione

```
/* Secondo programma di saluto */

#include <iostream>

using namespace std;

int main ()
{
        cout << "Ciao sono io: il tuo Personal Computer." << endl;
        cout << "Che desidera imparare tante cose nuove." << endl;
        cout << "Devi solo progettare ed implementare cosa devo fare." << endl;
        cout << "Inventando nuovi programmi."  << endl;
        cout << "Forza datti da fare. Ci divertiremo." << endl;
}
```

Altro esercizio

- Modificare il programma per fare in modo che richieda un numero all'utente e poi stampi a video il testo ''Il numero inserito è: '' seguito dal numero inserito

Soluzione

```
/* Terzo programma di saluto che usa anche cin */

#include <iostream>

using namespace std;

int main ()
{
        int numero;
        cin >> numero;
        cout << "Numero inserito: " << numero << endl;
}
```

Struttura dei programmi

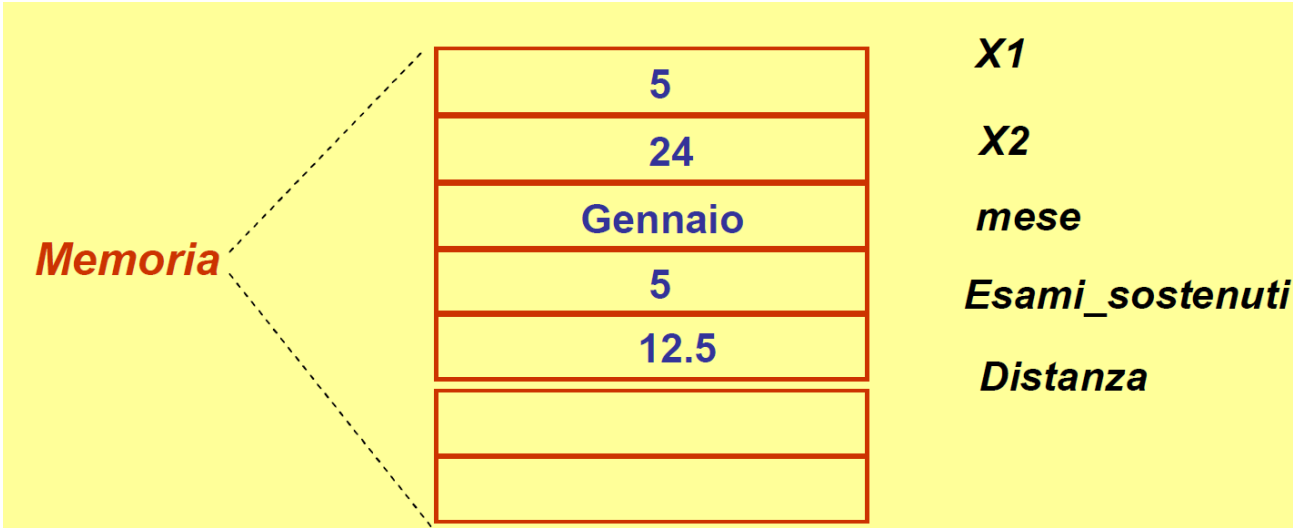
```
<definizioni>

int main ()
{
      prima_istruzione_da_eseguire;
      ......;
}
```

Commenti

- Hanno valore soltanto per il programmatore e vengono ignorati dal compilatore
- E’ possibile inserirli nel proprio codice in due modi diversi:
  - racchiudendoli tra i simboli `/*` e `*/`
  - facendoli precedere dal simbolo `//`
- Nel primo caso è considerato commento tutto quello che è compreso tra `/*` e `*/`
  - (il commento quindi si può estendere su più righe)
- Nel secondo caso il commento è ciò che va dal simbolo `//` alla fine della riga

Variabili: cosa sono

- Contenitori di valori (celle di memoria)
- Ogni variabile può contenere un singolo valore che può cambiare nel corso del processo elaborativo
- Il tipo di valore contenibile nella cella viene stabilito una volta per tutte e non può cambiare



Variabili: come si usano

- Definizione tramite identificatore e tipo del valore
  - int a , b;
- Scrittura valore (assegnazione)
  - a = 1;
- Lettura valore (uso)
  - b = a + 1;

- **Una variabile non può mai essere USATA prima di essere stata DEFINITA**

Identificatori: si usano per

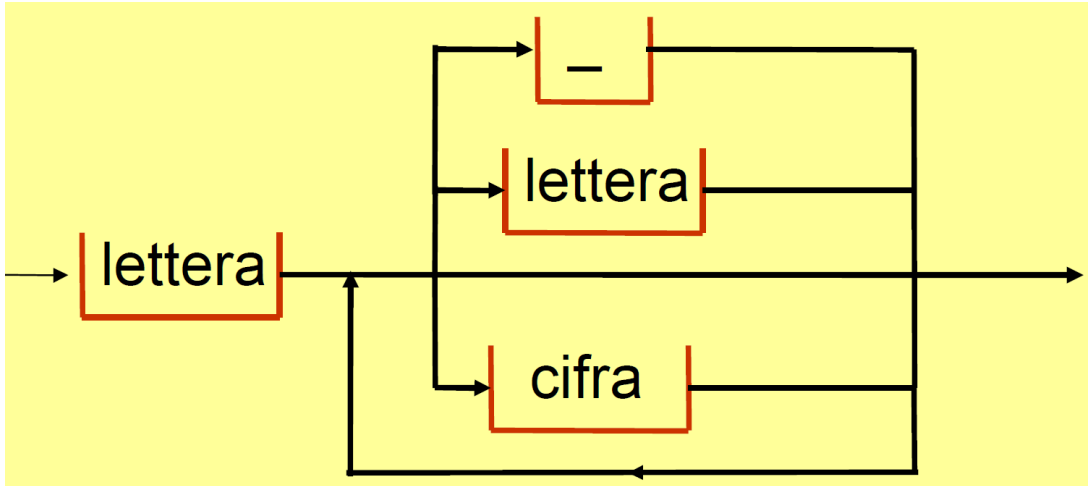
- I nomi di
  - Variabili
  - Costanti
  - Etichette
  - Tipi definiti dal programmatore
  - Funzioni
  - Parole riservate

Identificatori: formato

- Devono iniziare con un carattere alfabetico
- Utilizzabili lettere, cifre e carattere “_”
- Possono essere di lunghezza arbitraria
- Non utilizzabili nomi delle paroli chiavi
- Significativi caratteri maiuscoli e minuscoli (case sensitive)
- Non utilizzabile il carattere “spazio”

pi_greco alpha1 eq2grado

2alpha ax?xyeq 2grado



Tipi semplici

- Tipi semplici
  - bool
  - int
  - char
  - float
  - double
- Modificatori
  - short
  - unsigned
  - long

Esercizio

- Scrivere un programma che effettui lo scambio di valore tra due variabili di tipo intero
  - Far assegnare all'utente il valore delle due variabili e poi assegnare alla prima il valore della seconda ed alla seconda il valore della prima

Soluzione in 05-swap.cpp

Dimensione in memoria

- Grandezza dello spazio in memoria occupato dalla variabile e relative limiti di rappresentazione
  - `sizeof()`

Type	Size	Range	Format specifier	Example
char	1 byte	-128 to +127	%c	'a' , 'A'
unsigned char	1 byte	0 to 255	%c	'a' , 'A'
int	2 byte	-32768 to +32767	%d	-25, 5, 0, -5
unsigned int	2 bytes	0 to 65535	%u	254, 36777
long int	4 bytes	-2147483648 to +2147483647	%ld	45l, -5l, 5000l
unsigned long	4 bytes	0 to 4294967295	%lu	1000l, 20000l
float	4 bytes	$\pm 3.4*10^{\pm 38}$	%f	-3.5f ,125.13f
double	8 bytes	$\pm 1.7*10^{\pm 308}$	%lf	-125.25 , 270.6
long double	10 bytes	$\pm 3.4*10^{\pm 4932}$	%Lf	-330.45L, -1.2L

Esercizio

- Scrivere un programma che consenta di verificare se il valore di una variabile è uguale a quello inserito dall'utente

Soluzione in 07-varSizeCheck.cpp

Conversioni di tipo

- Conversioni implicite
  - `int x = 3.64;`
- Casting
  - double m;
  - int n = (int) m;
- Funzioni di conversione

Esempio in 08-casting.cpp

Scope della variabile

- In funzione di dove viene definite la variabile, essa diventa visibile solo ad una parte del codice
  - Variabili globali
  - Variabili locali
    - Al main
    - Ad una funzione
    - Ad un blocco di codice

- Esempio in 09-scope.cpp

Costanti

- Identificano valori che non cambiano nel tempo
- Definizione tramite identificatore e tipo
  - `const int numero;`
  - `const char carattere;`
- Scrittura costante (assegnazione)
  - `numero = -103; //Costante numerica`
  - `Carattere = ‘!’; //Costante carattere`
- Possono essere anche espressioni composte
  - `A = 1023 + ( - 254);`

#define

- E’ un’istruzione per il precompilatore che sostituisce il testo inserito nella definizione all’interno del programma, tutte le volte che lo incontra
  - `#define PI 3.14`
  - Dirà al compilatore di sostituire l’identificatore “PI” con il valore 3.14 tutte le volte che lo incontrerà nel nostro codice
  - Si può usare per definire costanti

Esempio in 10-const.cpp

Capite bene perché funziona così!

Costrutti di selezione if-then e if-then-else

```
if (condizione)
{
     Blocco di istruzioni
}
```

```
if (condizione)
{
     Blocco di istruzioni
}
else
{
     Blocco di istruzioni
     alternativo
}
```

Operatori algebrici

Operatore	Descrizione
+	Somma
-	Sottrazione
*	Moltiplicazione
/	Divisione
%	Modulo che restituisce il resto della divisione tra due interi

Operatori logici

Operatore	Descrizione
&&	AND o anche prodotto logico
||	OR o anche somma logica
!	NOT o anche negazione logica

Operatori relazionali

Operatore	Descrizione
==	Uguaglianza, per determinare se due valori sono uguali
!=	Diversità, per determinare se due valori sono diversi
>	Per determinare se il valore a sinistra precede quello di destra (o è più grande)
<	Per determinare se il valore a sinistra segue quello di destra (o è più piccolo)
>=	Per determinare se il valore a sinistra è più grande o uguale a quello di destra
<=	Per determinare se il valore a sinistra è più piccolo o uguale a quello di destra

Esempio

- Dato un numero intero in input scrivere se è maggiore di 10

```
// Uso di if
#include <iostream>
using namespace std;

int main()
{

/*
Dato un numero intero in input scrivere se è maggiore di 10
*/

       int n;
       cout << "Inserisci numero : ";
       cin >> n;

// Controllo se > 10
    if ( n > 10)
       cout << "n e' maggiore di 10" << endl;
}
```

Selezione tra più scelte

```
if (cond 1°caso)
     {codice 1° caso}
else if (cond 2°caso)
     {codice 2° caso}
else if …
else //ultimo caso
     {codice ultimo caso}
```

Esercizio

- Dato un numero intero in input scrivere se è
  - positivo, negativo o nullo
  - pari o dispari

Soluzione in 12–if_v2.cpp

Esercizio 2

- Dato un numero intero scrivere se è positivo, negativo o nullo, se è pari e dispari e se è maggiore di -3

Soluzione in 13–if_v3.cpp

Esercizio 3

- Dati due numeri naturali in ingresso valutare se solo l'uno il multiplo dell'altro

Soluzione in 14–if_multipli.cpp

Operatori bitwise

Operatore	Descrizione
&	AND bit a bit
|	OR bit a bit
^	OR esclusivo bit a bit
~	Complementazione bit a bit o anche inversione dei bit (zero al posto di uno e viceversa)
<<	Shift left
>>	Shift right

Pre-incremento e post-incremento

++i	i++	i = i +1
--i	i--	i = i -1

```
B = 2;
A = ++B;
//B è uguale a 3
//A è uguale a 3
```

```
B = 2;
A = B++;
//B è uguale a 3
//A è uguale a 2
```

Operatore condizionale

```
condizione ? risulato1 : risultato2
```

- Esempio

```
max = a < b ? b : a;
```

- Che è equivalente a

```
If (a < b)
      max = b;
else
      max = a;
```

Selezione tra più scelte con switch

```
switch (var)
{
      case valore1:
            {codice1};
            break;
      case valore2:
            {codice2};
            break;
      …
      default:
            {codice altri casi}
}
```

```
if (cond 1°caso)
      {codice 1° caso}
else if (cond 2°caso)
      {codice 2° caso}
else if …
else //ultimo caso
      {codice ultimo caso}
```

Esercizio

- Dato un numero tra 1 e 12 scrivere a video il nome del mese corrispondente

Soluzione in 15-switch_mesi.cpp

Esercizio 2

- Dato un numero tra 1 e 12 scrivere a video il nome del mese corrispondente
  - Chiedendo in input anche se vogliamo il nome in inglese o in italiano

Soluzione in 16-switch_mesi_eng.cpp

Iterazione: while-do

```
while (condizione)
{
    Blocco di istruzioni
}
```

- Ripete il blocco di istruzioni 0 o più volte fin quando la condizione è verificata

Esercizio

- Siccome ogni dato in input deve essere **validato**
- Calcolare e stampare a video il valore della lunghezza della circonferenza dato il raggio, utilizzando una costante per π e chiedendo all'utente il valore del raggio, ripetendo la domanda fin quando il valore immesso non è positivo
  - $L = 2 \pi r$

Soluzione in 17-while_circonferenza.cpp

Iterazione: do-while

```
do
{
     Blocco di istruzioni
}
while (condizione);
```

- A differenza del while-do, il blocco di istruzioni in questo caso verrà ripetuto almeno una volta

Esercizio: minimo, massimo, media

- Scrivere un programma che calcoli il minimo, il massimo e la media dei voti di un esame
  - Dopo ogni inserimento il programma chiede se tutti i voti siano stati inseriti

Soluzione in 18-minMaxMed_voti.cpp

Esercizio 2: Gioco

- Implementare un piccolo gioco nel quale bisogna indovinare un numero intero tra 1 e 100
- Ogni volta che il giocatore non indovina il numero, il programma dice se il numero da indovinare è più alto o più basso di quello provato
  - Per fare le prove è opportuno fissare un numero costante da indovinare

Esercizio 2: Gioco con numero random

- Modificare il gioco per fare in modo che il numero da indovinare sia scelto a caso dal computer

```
#include <cstdlib>
#include <ctime>
[…]
int seed = time(NULL);
srand(seed);
int numeroGrande = rand();
int numero = 1 + (numeroGrande % 100);
```

Soluzione in 19-gioco.cpp

Esercizio 2: Gioco con massimo di tentativi

- Modificare il gioco imponendo che il numero debba essere indovinato al massimo in 7 tentativi

Soluzione in 20-gioco_v2.cpp

Esercizio 3: Morra cinese

- Si realizzi un programma che implementa il gioco della Morra Cinese
- Il gioco della Morra Cinese è per due giocatori. Ogni giocatore sceglie un elemento tra i tre possibili (Forbici, Carta, Sasso). Il vincitore viene deciso nel seguente modo
  - Le Forbici vincono sulla Carta
  - La Carta vince sul Sasso
  - Il Sasso vince sulle Forbici
- In caso di elementi uguali, il gioco è in pareggio

Esercizio 3: Morra cinese (cont.)

- Realizzare il gioco in una modalità contro il computer, nella quale si suppone che il computer scelga sempre il Sasso
- Il programma deve chiedere prima di tutto a quale punteggio deve terminare la partita
  - ogni vittoria vale 1 punto, il pareggio non attribuisce punti
- Successivamente, devono essere svolte le giocate fino a che non si possa determinare il vincitore, che verrà dichiarato con un messaggio a video

- Soluzione in 21-morraCinese.cpp

Esercizio 3: Morra cinese con elemento random

- Modificare il programma per fare in modo che il computer scelga ogni volta un elemento in maniera casuale

Esercizio 3: Altre varianti

- Prevedere anche una modalità a due giocatori, nella quale viene prima chiesto il simbolo del primo giocatore, poi quello del secondo (la scelta della modalità viene fatta all'inizio del programma)
- Prevedere anche la possibilità di terminare la partita dopo un certo numero di giocate, anziché al raggiungimento di un valore finale

Iterazione: for

```
for (prologo; condizione; continuazione)
{
     Blocco di istruzioni;
}
```

- Equivalente a

```
prologo;
while (condizione)
{
     Blocco di istruzioni;
     continuazione;
}
```

Esercizio 1: Minimo, massimo, e media con il for

- Scrivere un programma che calcoli il minimo, il massimo e la media di un numero prefissato di interi
  - All'inizio del programma viene chiesto il numero dei valori da inserire
  - Si usi il ciclo for per l'inserimento dei valori

Soluzione in 22-minMaxMed_for.cpp

Esercizio 2: Alfabeto inverso

- Scrivere a video tutte le lettere minuscole dell'alfabeto in ordine inverso, dalla z alla a
  - Si usi un ciclo for che opera su un char
  - Si ricordi che il char è un byte e se gli assegno un valore in questo modo
  - `char lettera = 'z';`
  - Allora al byte verrà assegnato il valore corrispondente alla lettera z dalla tabella ascii

Soluzione in 23-alfabetoInverso.cpp

Esercizio 3: elevazione a potenza

- Scrivere un programma che riceva in ingresso un numero intero x ed un numero intero y e calcoli $z = x^y$ utilizzando un ciclo for

Soluzione in 24-potenza_for.cpp

Esercizio 4: elevazione a potenza con while

- Scrivere un programma che riceva in ingresso un numero intero x ed un numero intero y e calcoli $z = x^y$ utilizzando un ciclo while

Soluzione in 25-potenza_while.cpp

Debugging con Dev Cpp

- Inserimento di breakpoint
- Valutazione di valori di variabili o di espressioni mentre l’esecuzione è interrotta (watch)
- Esecuzione passo per passo del programma (step by step)

Dati strutturati

- Sono composti da un insieme ordinato di dati semplici
  - Ogni dato semplice è caratterizzato da tipo, valore, attributo
- l più semplice esempio di dato strutturato è dato dagli array

Array

- Un array è una collezione ordinata di dati omogenei
  - Tipo dei dati dell'array
  - Meccanismo di memorizzazione degli array
  - Dichiarazione/definizione di un array
    - `int nomeArray[10] = {int1, int2, …};`
  - Accesso ai singoli elementi dell'array
    - `nomeArray[0]`
    - `nomeArray[1]`
    - …
    - `nomeArray[9]`
- Aritmetica degli indici

Operazioni sugli array

- Dato un vettore di numeri interi ordinati in ordine crescente (oppure non ordinati)
  - Visualizzare il contenuto del vettore
  - Trovare in che posizione si trova un dato valore
  - Trovare un elemento in un vettore ordinato
  - Cancellare un elemento dal vettore
  - Inserire un valore nel vettore in una determinata posizione
  - Inserire un elemento in un vettore ordinato rispettando l’ordine

Visualizzare il contenuto

```
int v[10] = {1, 2, 5, 8, 12, 15, 20};
int riemp = 7;
cout << "Vettore :";
for (int i = 0; i < riemp; i++)
    cout << v[i] << ", ";
cout << endl;
```

Ricerca di un elemento

```
int num; //Valore da cercare
cout << "Valore da cercare? ";
cin >> num;
bool trovato = false;
bool superato = false;
int i = 0;
while ((!trovato) && (i < riemp) && (!superato))
{
        if (v[i] == num)
                trovato = true;
        else if (v[i] > num)
                superato = true;
        else
                i++;
}
if (trovato)
        cout << "Trovato in posizione " << i << endl;
else
        cout << "Non trovato" << endl;
```

Eliminazione di un elemento

```
cout << "Numero da eliminare? ";
cin >> num;
trovato = false;
// Utilizziamo l'algoritmo di ricerca per trovare la posizione del valore da eliminare
while ((!trovato) && (i < riemp) && (!superato))
{
        if (v[i] == num)
                trovato = true;
        else if (v[i] > num)
                superato = true;
        else
                i++;
}
// E lo eliminiamo se lo troviamo, spostando verso sinistra tutti i successivi
if (trovato)
{
        riemp--;
        for (int j = i; j < riemp; j++)
                v[j] = v[j+1];
}
else
        cout << "Non trovato" << endl;
```

Inserimento di un elemento alla volta in un vettore ordinato

```
do
{
        int i = 0, j, k;
        cout << "Numero da inserire? ";
        cin >> num;
        // Qui utilizziamo l'algoritmo di ricerca
        // per trovare i, indice del primo elemento uguale o superiore
        // a quello da inserire
        // Una volta trovato, spostiamo a destra tutti gli elementi
        // successivi per fare posto per il nuovo elemento da inserire
        for (j = riemp; j > i; j--)
                v[j] = v[j-1];
        // Poi lo inseriamo
        v[i] = num;
        riemp++;
        cout << "Vettore : ";
        for (int k = 0; k < riemp; k++)
        cout << v[k] << ", ";
        cout << endl;
} while (riemp < MAXV);
```

Esempio

- Calcolare la media di un insieme di 5 numeri utilizzando un array

- Soluzione in 27-mediaArray.cpp

Stringhe

- Una parola può essere dichiarata come un array di char (unica soluzione prevista in linguaggio C)

```
char parola[10];
```

- Particolarità
  - Può essere assegnata tutta la stringa con un solo comando, ma solo nell'istruzione di dichiarazione

```
char parola[]="parola";
char parola[7]="parola";
```

- Il settimo carattere di "parola" è un valore ASCII zero ('\0'), che indica la fine della stringa

Operazioni su stringhe

- Accesso a una lettera

```
parola[3]  // accede alla quarta lettera della parola
```

- Lunghezza della stringa

```
int l = strlen(parola);
```

- Confronto tra stringhe

```
cfr = strcmp(parola1, parola2);
```

  - Vale 0 se sono uguali, un numero negativo se parola1 precede parola2 in ordine alfabetico, un numero positivo altrimenti

- Assegnazione

```
strcpy(parola2, parola1);
```

  - Equivale a `parola2 = parola1` (ma questa sintassi sarebbe scorretta!)

- Concatenamento

```
strcat(parola1,parola2)
```

  - Equivale a `parola1 = parola1 + parola2` (ma anche questa sintassi sarebbe scorretta)

- Input e output di stringhe
- Attenzione all'uso di cin !

Esempio sulle stringhe

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
	char s[50] = "parola1";
	char t[50] = "parola2";
	cout << strlen(s) << endl;
	cout << strcat(s,t) << endl;
	cout << strcmp(s,t) << endl;
}
```

Esempio in 28-esempioStringhe.cpp

Esercizio sulle stringhe

- Data una stringa in input visualizzare una stringa contenente tutte le sue vocali e un’altra stringa contenente tutte le sue consonanti

Soluzione in 29-stringhe_vocaliConsonanti.cpp

Altro esercizio sulle stringhe

- Modificare la morra cinese per supportare l'utilizzo delle stringhe ("Sasso", "Forbici", "Carta") e chiedere se fare un'altra partita ("Si", "No")

- Soluzione in 30-stringhe_morraCinese.cpp

Esercitazione: Algoritmi con le stringhe

- Implementare funzioni equivalenti a quelle appena viste:
  - strlen
  - strcmp
  - strcpy
  - strcat

- Nota: si ricordi che stringa[i] = 0 (inteso come numero 0) è equivalente a stringa[i] = ‘\0’ ma non è equivalente a stringa[i] = ‘0’ (carattere 0 della tastiera, che ha codice ASCII 48)

strlen()

```
char s[] = “parola1”;
int i = 0;
while (s[i] != 0)
     i++;
cout << i << endl;
```

Oppure, in una sola riga

```
for (i = 0; s[i] != 0; i++);
```

Oppure

```
for (i = 0; s[i]; i++);
```

strcpy()

```
char s[10];
char t[] = “parola”;
int i = 0;
while (t[i] != 0){
      s[i] = t[i];
      i++;
}
s[i]=0;
cout << s << endl;
```

Oppure

```
for (int i = 0; t[i] != 0; s[i] = t[i], i++);
s[i] = '\0';
```

strcat()

```
int i, j;
char s[] = "parola1";
char t[] = "parola2";
char u[100]; // oppure char u[strlen(s)+strlen(t)-1]
for (i = 0; s[i] != 0; i++)
     u[i] = s[i];
for (j = 0; t[j] != 0; j++)
     u[i + j] = t[j];
u[i + j] = 0;
cout << u << endl;
```

strcmp()

```
char s[50];
char t[50];
int res;
cout << "Prima parola : ";
cin >> s;
cout << "Seconda parola : ";
cin >> t;
int cont = 0;
while (s[cont] == t[cont] && s[cont] != 0)
        cont++;
if (s[cont] == 0 && t[cont] == 0)
        cout << "Sono uguali« << endl;
else if (s[cont] < t[cont])
        cout << s << " precede " << t << endl;
else
        cout << t << " precede " << s << endl;
```

Matrici

- Con il termine matrice si intende, genericamente, un vettore a due o più dimensioni
- Dichiarazione di una matrice quadrata

```
int a[10][10];
```

- Dichiarazione di una matrice rettangolare con 2 righe e 4 colonne

```
int a[2][4];
```

- Dichiarazione di una matrice tridimensionale

```
int a[3][4][3];
```

Operazioni di base sulle matrici

- Accesso ad un elemento di una matrice

```
cout << a[1][3]; // In lettura
cin >> a[1][3]; // In scrittura
```

- Memorizzazione per righe di una matrice rettangolare

```
int a[4][3];
for (int i = 0; i < 4; i++)
     for (int j = 0; j < 3; j++)
          cin >> a[i][j];
```

Vedi file: 31-matrici_opBase.cpp

Esercizio 1

- Data una matrice con 4 righe e 3 colonne, calcolare la somma ed il massimo dei valori per ogni riga

- Soluzione in: 32-matrici_somma_e_max.cpp

Esercizio 2

- Data una matrice, scambiare (swap) due sue righe

- Soluzione in: 33-matrici_scambia_riga.cpp

struct

- Una struct (o record) può tenere insieme un insieme limitato di dati eventualmente non omogenei
- Una struct è formata di campi, ognuno dei quali può assumere un qualsiasi tipo
  - E’ possibile che un campo di una struct sia a sua volta una struct
  - La struct è, quindi, un nuovo tipo strutturato definito all’interno di un programma
    - E’ possibile quindi dichiarare anche array o matrici di struct

struct

- Dichiarazione della struct

```
struct nome_struct {
        Tipo1 nome_campo_1;
        ..
        tipoN nome_campo_N;
}
```

- Dichiarazione di una variabile di tipo struct:

```
struct nome_struct nome_variabile;
```

- Definizione di una variabile di tipo struct:

```
struct nome_struct nome_variabile = {valore1, …, valoreN}
```

- Accesso ad un campo in lettura o scrittura

```
nome_variabile.nome_campo = valore;
valore = nome_variabile.nome_campo;
```

struct

- Assegnazione

```
nome_variabile_struct1 = nome_variabile_struct2;
```

  - Attenzione: quest'assegnazione dipende dalle assegnazioni dei tipi che compongono la struct. Se ci sono stringhe o array il comportamento potrebbe non essere quello atteso!
  - E' più corretto assegnare un campo alla volta
- Input, output
  - Ad un campo alla volta
- Confronto
  - L'operatore == Ha lo stesso comportamento del caso array
  - Non esistono operatori come < o >

Esempi

```
struct data
{
	int anno;
	int giorno;
	char mese[20];
} oggi = {2012, 7, "novembre"};

struct data domani = {2016, 16, ”novembre”};
oggi.giorno = 21;
strcpy(oggi.mese, "dicembre");
oggi.anno = 2012;
```

```
struct persona
{
	char nome [20];
	char cognome [20];
	int peso;
};
struct persona p;
strcpy(p.nome, ”Pinco”);
strcpy(p.cognome, ”Pallino”);
p.peso = 75;
cout << p.nome << p.cognome << p.peso;
```

Altro esempio

```
struct persona
{
	char [20] nome;
	char [20] cognome;
	int peso;
	struct data nascita;
};
struct persona p;
strcpy(p.nome, ”Pinco”);
strcpy(p.cognome, ”Pallino”);
p.peso = 75
p.nascita.giorno = 10;
strcpy(p.nascita.mese, ”novembre”);
p.nascita.anno = 2009;
```

Esercizio

- Realizzare un programma che memorizza le seguenti informazioni in una struct chiamata data:
  - Giorno, mese, anno, un'array di 2 stringhe per il nome dei due più importanti santi del giorno, una struct che contiene nome, cognome e peso della persona che festeggia il compleanno in quel giorno
  - Il programma chiede all'utente tutti i valori per il giorno corrente e li stampa a video dopo averli memorizzati nella struct

- Soluzione in: 34-esempiStruct_v2.cpp

typedef

Un modo più generale per creare nuovi tipi è il typedef

```
typedef tipo_esistente nuovo_tipo
```

Ad esempio:

```
typedef float reale;
reale x = 3.5;
typedef struct persona individuo;
individuo x; x.peso = 90;
typedef char[35] parola;
parola p = “parola”;
```

Esercizio 2

• Progettazione e realizzare un programma che è in grado di memorizzare le seguenti informazioni per un insieme di massimo 20 giorni:

- Giorno, mese, anno, il nome dei due più importanti santi del giorno (utilizzando un’array di 2 stringhe), il nome, cognome e peso della persona che festeggia il compleanno in quel giorno (utilizzando un'altra struct);
- Il programma chiede all’utente tutti i valori per il giorno corrente e per tutti gli altri giorni che vuole (massimo 19) e li stampa a video dopo averli memorizzati nella struttura dati.

- Soluzione in: 34-esempiStruct_v3.cpp

Puntatori

Data una qualsiasi variabile, l'operatore & ci restituisce l'indirizzo di memoria nel quale la variabile viene memorizzata

```
&x // è indirizzo di x (puntatore ad x)
```

L'operatore inverso è l'operatore di puntamento * che, dato un puntatore restituisce il valore della variabile

Utilizzo dei puntatori

```
int y; // data una variabile intera
int* x;
x = &y; // definisco il puntatore a questa variabile
```

Conseguenze

```
cout << x // stampa il valore dell'indirizzo di y
cout << *x // stampa il valore puntato da x, ovvero il valore di y
*x = 10; // modifica il valore di y;
```

In pratica ci sono due modi equivalenti per accedere al valore di y

```
y
*x
```

NOTA: *&y == y; Cioè * e & sono operazioni inverse tra loro

Per azzerare un puntatore:

```
x=0;
x=NULL;
```

Puntatori a struct

- Nel caso di una struttura:

```
struct elemento {
        int riga;
        int colonna;
        float valore;
} e;
```

- Un puntatore può essere definito come

```
struct elemento* ptr; ptr = &e;
```

- Per accedere alla riga dovremmo scrivere

```
(*ptr).riga
```

- Dove le parentesi servono per precedenza. Ma si può abbreviare con

```
ptr->riga
```

- Se **ptr1** e **ptr2** sono due puntatori dello stesso tipo, scrivendo

```
ptr1 = ptr2;
```

- Abbiamo due variabili che puntano alla stessa variabile (due alias)

Puntatori ad array

- Gli array rappresentano un caso particolare di puntatore

```
int v[3];
```

- v (senza indici) è esso stesso un puntatore
- v è anche l'indirizzo del primo elemento del vettore stesso
- Esempio:

```
int v[3] = {25, 28, 34};
int *pzero = &v[0];
int *puno = &v[1];
cout << pzero << endl << puno << endl << &v <<endl;
```

Allocazione statica e dinamica

- Per allocazione statica si intende una allocazione di memoria a variabili che avviene a tempo di compilazione e/o al più un'unica volta durante un'esecuzione di un programma
- Tutte le dichiarazioni di variabili viste finora vengono risolte con allocazione statica
- L'allocazione statica non ci consente di risolvere problemi come quello di gestire una lista di elementi di lunghezza variabile, allocando, istante per istante, solo lo spazio di memoria minimo indispensabile

Allocazione dinamica

- Per allocare dinamicamente una variabile con un'istruzione

```
int* p = new int;
```

  - `p` è un puntatore, ed è allocato staticamente (è dichiarato come ogni altra variabile vista finora)
  - `*p`, valore puntato da `p` è un `int` allocato dinamicamente all'atto dell'esecuzione dell'istruzione `new`
- Per distruggere la variabile dinamica puntata da p

```
delete p;
```

- Con questa istruzione viene distrutta `*p`, non il puntatore `p`, che viceversa scompare quando termina naturalmente il proprio scope
- Per allocare un array di n elementi

```
int* p = new int[n];
```

- Per distruggere un array

```
delete [] p;
```

Esercizio

- Scrivere un programma che effettui lo scambio di valore tra due variabili di tipo intero create dinamicamente
  - Far assegnare all'utente il valore delle due variabili e poi assegnare alla prima il valore della seconda ed alla seconda il valore della prima

Soluzione in 35a-swapDimanico.cpp

Liste

- Una Lista, a differenza di un array, ha un numero variabile di elementi ed è tale da allocare, in ogni istante, un quantitativo di memoria pari a quella minima necessaria

```
struct Elemento {
        int valore;
        struct Elemento* prossimo
};
typedef struct Elemento Lista;
```

- La definizione di lista coincide con quella di un elemento (il primo elemento della lista stessa, collegato al secondo, che è collegato al terzo e così via)
- `prossimo` è un puntatore all'elemento successivo della lista
- L'ultimo elemento della lista punterà ad un elemento inesistente, rappresentato dalla costante NULL (solitamente pari a 0, valore sicuramente non utilizzabile per un indirizzo)

Esempio: Riempimento

```
struct Elemento {
     int valore;
     struct Elemento* prossimo;
     };
typedef struct Elemento Lista;
Lista* l=new Lista;
l->valore=4;
l->prossimo = new struct Elemento;
l->prossimo->valore = 5;
l->prossimo->prossimo = new struct Elemento;
l->prossimo->prossimo->valore = 7;
l->prossimo->prossimo->prossimo = NULL;
```

Riempimento di una lista da un vettore

```
int v[3] = {4,5,7};
int riemp = 3;
cout << "Riempimento della lista da un vettore" << endl;
Lista* lista = new Lista;
{
	Lista* p = lista;
	int i = 0;
	while (i < riemp){
		p->valore = v[i];
		i++;
		if (i < riemp)
			p->prossimo = new Lista;
		else
			p->prossimo = NULL;
		p = p->prossimo;
	}
}
```

Visualizzazione della lista

```
Lista* p = lista;
while (p != NULL)
{
     cout << p->valore << endl;
     p = p->prossimo;
}
```

Inserimento del valore 6 dopo il 5

```
p = lista;
while (p->valore != 5 && p->prossimo != NULL)
     p = p->prossimo;
if (p->valore == 5){
     Elemento* e = new Elemento;
     e->valore = 6;
     e->prossimo = p->prossimo;
     p->prossimo = e;
}
```

Eliminazione del valore 5

```
p = lista;
struct Elemento* precedente = NULL;
while (p->valore != 5 && p->prossimo!=NULL){
     precedente = p;
     p = p->prossimo;
}
if (p->valore == 5){
     precedente->prossimo = p->prossimo;
     delete p;
}
```

Deallocazione della lista

```
p = lista;
while (p != NULL){
     Lista* q = p;
     p = p->prossimo;
     delete q;
}
lista = NULL;
```

Esercizio

Creare un programma che riempia una lista con valori forniti dall'utente tramite tastiera. Al termine dell'inserimento, il programma deve stampare a video tutti gli elementi della lista.

Funzioni

Per risolvere un problema complesso, è opportuno scomporlo in sottoproblemi

- Un sottoproblema può essere risolto da un sottoprogramma
- Analogamente ad un programma, un sottoprogramma ha un insieme di dati in input e restituisce un insieme di dati in output
- Un sottoprogramma viene chiamato da un altro sottoprogramma, il quale gli fornisce I valori di input e al quale vengono restituiti i valori di output
  - main è un caso particolare di sottoprogramma, che deve essere presente in ogni programma C++
  - In C++ i sottoprogrammi sono sempre codificati come funzioni

Dichiarazione e definizione di una funzione

```
tipo_restituito nomeFunzione (lista di parametri)
{
     corpo della funzione
     return valore; // Di tipo tipo_restituito
}
```

- Il nome della funzione deve essere univoco
- Ogni funzione restituisce un valore di un tipo scalare (anche un puntatore) e `valore` deve appartenere a questo tipo
  - Es.: main ha come tipo restituito int (e nei nostri esempi aveva valore di ritorno 0)
- Così come le variabili, è possibile dichiarare una funzione senza definirla subito
  - La definizione deve essere però presente in un punto successivo del programma

Esempio

```
int strlen (char parola[50]){
      int i = 0;
      while (parola[i] != 0)
            i++;
      return i;
}
int main () {
      char testo[50] = ‘cheneso’;
      char testoNuovo[50];
      int lunghezza;
      lunghezza = strlen (testo);
      cout << lunghezza;
      strcpy (testoNuovo, testo);
}
```

Chiamata di funzione

```
variabile = nomeFunzione (lista di parametri)
```

- Una funzione può essere chiamata da una qualsiasi altra funzione dello stesso programma
- Variabile deve essere una variabile dello stesso tipo del valore ritornato dalla funzione
  - Variabile è opzionale: si può anche scrivere

```
nomeFunzione (lista di parametri)
```

- In questo caso il valore ritornato viene perduto
- Es.:

```
int l = strlen(s);
cout << strlen(“parola”);
```

Passaggio dei parametri

- Concettualmente, si può distinguere tra
  - Parametri di ingresso (in)
  - Parametri di uscita (out)
  - Parametri di ingresso e uscita (inout)
- Tecnicamente, esistono tre modalità di passaggio dei parametri, in C++
  - Per valore
  - Per puntatore
  - Per riferimento

Passaggio per valore

```
int somma (int addendo1, int addendo2)
      { return addendo1 + addendo2; }
int main(){
      int x, y, z;
      x = 100;
      y = 3;
      z = somma(x, y);
}
```

Passaggio per valore: considerazioni

- Non è possibile (per le nostre attuali conoscenze) passare per valore un dato strutturato (al più si può passare un puntatore)
- Le variabili addendo1 e addendo2 nell'esempio precedente devono essere considerate come se fossero delle variabili locali
  - il cui scope si limita al corpo della definizione della funzione
  - Il cui valore iniziale è quello che gli giunge dalla chiamata
    - Nell'esempio, le variabili x e y, il cui valore veniva trasferito ad addendo1 e addendo2 non possono venire in nessun modo toccate dall'esecuzione della funzione
- Il passaggio per valore è adatto al trasferimento delle sole variabili di input

Passaggio per puntatore

- Come trasmettere dei parametri di output o di input e output ad una funzione?
  - E’ necessario che la funzione abbia la possibilità di manipolare I valori delle variabili trasmesse
- L’unico meccanismo di scambio tecnicamente possibile è quello per valore
- Soluzione:
  - Trasmettiamo per valore il puntatore alla variabile

Passaggio per puntatore: come si fa

- Dichiarazione e definizione

```
void dimezza (float* x)
{
      *x = *x / 2.0;
      return;
}
```

- Chiamata:

```
float a = 3.14;
float* p = &a;
dimezza(p);
```

Passaggio per puntatore: considerazioni

- Nella chiamata viene effettivamente passato per valore l'indirizzo (&a)
- Nella dichiarazione si dice che la funzione attende una variabile x di tipo puntatore a float
- Nella definizione si sfrutta la conoscenza del puntatore x per andare a modificare la variabile *x da esso puntato
  - Ma *x coincide proprio con la variabile a definita nella funzione chiamante
- Al momento della chiamata della funzione è come se avvenisse

```
float *x = &a;
```

  - cioè parametro della funzione = valore inviato dal programma

Passaggio di una struct

Una struct può essere passata soltanto per puntatore

```
void visualizza (struct elemento *e)
{
      cout << e->valore;
      return;
}
```

- Chiamata

```
struct elemento s;
…
visualizza (&s);
```

- Nella chiamata è come se avvenisse

```
struct elemento *e = &s;
```

Passaggio di un array

- Anche un array può essere passato solo per puntatore
  - Ma il nome dell'array è anche un puntatore ad esso

```
void visualizza (int e[]) //la dimensione non è necessaria
{
        cout << e[0] << e[1] << e[2];
        return;
}
```

- Chiamata

```
int a[3];
…
visualizza (a);
```

- La dimensione dell'array può essere omessa dalla dichiarazione della funzione
  - Ma bisogna essere attenti a non andare oltre la sua dimensione, nella funzione
  - Nella chiamata è come se avvenisse

```
int e[] = a;
```

Passaggio per riferimento

- Il passaggio per riferimento permette, così come il passaggio per puntatore, di trasmettere variabili di output o input/output all'interno di una funzione
- A differenza del passaggio per puntatore, bisogna tenerne conto solo nella dichiarazione della funzione, non nella chiamata, nè nella definizione
- Dichiarazione

```
tipoRestituito nomeFunzione (tipoVariabile &nomeVariabile)
```

- Definizione

```
{ … nomeVariabile = … }
```

- Chiamata

```
nomeFunzione (variabile)
```

Passaggio per riferimento: esempi

- Dichiarazione e definizione

```
void dimezza (float &x) {
        x = x / 2.0;
        return;
}
```

- Chiamata

```
float a = 3.14;
dimezza(a);
cout << a;
```

Esercizio 1

- Calcolare il cubo dei primi n numeri naturali, con n fornito dall'utente
- Il programma deve prevedere una funzione che calcola il cubo di un numero
- Il main deve richiedere il valore di n e poi chiamare la funzione per il calcolo del cubo n volte, con valore della variabile di input che va da 1 a n

- Risultato in 39-funzioni_cubo.cpp

Esercizio 2

- Calcolare la somma ed il massimo dei valori di un vettore di interi
- Il programma deve prevedere una funzione che calcola il massimo ed un'altra che calcola la somma
- Il main deve richiedere la lunghezza del vettore, i valori da inserire nel vettore, poi chiamare le due funzioni e riportare a video il valore restituito da queste

- Risultato in 40-funzioni_vettori.cpp

Librerie di funzioni

- Una funzione può risultare utile in più di un programma
- E’ utile raccogliere un insieme di funzioni che hanno qualcosa in comune all’interno di una stessa libreria
- Una libreria può essere vista come un file con la dichiarazione e la definizione di un insieme di funzioni, tra le quali non c’è, però, la funzione main

Esempio di progetto

- Obiettivo: separare le funzioni di Insiemi in modo da poterle utilizzare in un programma qualsiasi
  - In un file header insiemi.h sistemiamo tutte le dichiarazioni delle funzioni
  - In un file insiemi.cpp sistemiamo le definizioni di tutte le funzioni sugli insiemi
  - In un programma qualsiasi usaInsiemi.cpp definiamo una funzione main che utilizza liberamente tutte le funzioni sugli insiemi
- usaInsiemi.cpp deve avere anche una riga #include “insiemi.h”
- Per la compilazione in DevCpp, tutti i file creati (insiemi.h, insiemi.cpp, usaInsiemi.cpp) devo fare parte dello stesso progetto (aperti nello stesso DevCpp)

```
int stampaArray (int array[50]);

int main (){
…
stampaArray(vettoreMio);

…
}

int stampaArray (int array[50]){
     corpo della funzione;
}
```

Compilazione separata

- Molto spesso è utile separare la dichiarazione delle funzioni dalla loro definizione
  - E’ possibile distribuire, in una libreria, direttamente la versione compilata della definizione delle funzioni (il file oggetto)
  - Consentendo agli utenti di utilizzarla, ma non di modificarla
  - Risparmiando la fatica di doverla ricompilare ad ogni programma che la utilizza
- I file contenenti tutte le dichiarazioni sono detti header file e, di solito, hanno estensione .h

Definizione ed uso di libreria

- Nel file funzioniInsiemi.cpp che definisce le funzioni della libreria (che sono solo dichiarate in Insiemi.h) dovrà esserci

```
#include ".\insiemi.h"
```

- Che fa includere fisicamente il contenuto di insiemi.h (che si trova nella cartella corrente) all'interno di funzioniInsiemi.cpp stesso al momento della compilazione
- funzioniInsiemi.cpp ha bisogno di conoscere le strutture dati dichiarate in insiemi.h
- Allo stesso modo, anche main.cpp deve includere insiemi.h
- main.cpp ha bisogno di conoscere le strutture dati e le funzioni dichiarate in insiemi.h (e definite in funzioniInsiemi.cpp)

Esempio insiemi.h

```
#define DIM 10
void input (int v[DIM], int &riempv);
void visualizza (int v[], int riemp);
void unione(int v1[DIM],int v2[DIM], int u[DIM*2], int riempv1, int
riempv2, int &riempu);
void intersezione (int v1[DIM],int v2[DIM], int inters[DIM], int riempv1,
int riempv2, int &riempinters);
void differenza(int v1[DIM], int v2[DIM], int diff[DIM], int riempv1, int
riempv2, int &riempdiff);
```

Esercizio

- Calcolare la somma ed il massimo dei valori di un vettore di interi
- Il programma deve prevedere una funzione che calcola il massimo ed un'altra che calcola la somma. Il main deve richiedere la lunghezza del vettore, i valori da inserire nel vettore, poi chiamare le due funzioni e riportare a video il valore restituito da queste
- *Il programma deve essere strutturato in 3 file: uno per il main, uno per le intestazioni delle due funzioni ed uno per le definizioni delle due funzioni*

Ricorsione

- Può una funzione richiamare sé stessa?
  - Tecnica della ricorsione
- La ricorsione risolve un problema in termini di sé stesso, fino a ricondursi ad un caso base di cui è nota la soluzione
  - Analogo al principio di induzione utilizzato per dimostrare teoremi
- Il caso «base» deve avere una soluzione banale
- I casi «generali» devono ricondurre la soluzione ad una esecuzione del problema stesso
- I casi «generali» devono portare, prima o poi, al caso «base»

Ricorsione: esempio

- Calcolo della potenza con base e esponenti interi
  - $b^e = b * b^{e-1}$
  - $b^0 = 1$

```
int potenza(int base, int esp){
     if (esp == 0)
          return 1;
     else
          return base * potenza(base, esp-1);
}
```

Ricorsione: altri esempi

- Calcolo del fattoriale
  - Caso base: 1!=1
  - Caso generale: n! = n* (n-1)!

- Ricerca di un valore in un vettore ordinato (crescente)
  - Partiamo dal centro del vettore
  - Caso base: abbiamo trovato il valore
  - Casi generali
    - Abbiamo trovato un valore più grande: ripetiamo la ricerca su un vettore che va dall'inizio all'elemento appena provato
    - Abbiamo trovato un valore più piccolo: ripetiamo la ricerca su un vettore che va dall'elemento appena provato alla fine

FILE

- Sono utilizzati per memorizzare dati in maniera persistente su disco
- I file sono sequenze di elementi omogenei (come gli array o le stringhe)
- Un file termina con un apposito carattere EOF (end of file, come le stringhe)
- Un file può essere letto o scritto come uno stream (come cin e cout), tramite gli operatori << e >>
- Per poter operare su di un file è necessario prima aprirlo. Per rendere definitive tutte le operazioni fatte sul file, è necessario chiuderlo

Operazioni sui file

- Dichiarazione tipi ifstream e of ofstream

```
#include <fstream.h>
ifstream in_file;
ofstream out_file;
```

- Apertura dei file

```
in_file.open("c:\Documents\Alessio\testo.txt")
out_file.open("testo.txt")
```

- Chiusura dei file

```
file.close()
```

- Lettura

```
in_file>>x;
```

- Scrittura

```
out_file<<x;
```

- Controlli
  - Avvenuta apertura: `file.is_open()`
  - Fallimento di un'operazione `file.fail()`
  - Raggiunta terminazione `in_file.eof()`

Esempio

- Scrivere un programma che legge degli interi da un file di testo e scrive gli interi letti, sommandogli 1, in un file di output

- Soluzione in 44-file e su apogeo
  - L’input e l’output -> [Botta] esempio file leggi e scrivi + 1

Esercizio con i file

- Dato in input un qualsiasi file di testo, ad esempio lo stesso file sorgente che si sta scrivendo
- Fare una statistica della frequenza con la quale occorrono i caratteri del codice ASCII e scrivere in output la classifica dei dieci caratteri più comuni
- L'output deve essere scritto sia a video che in un file "classifica.txt"
- Suggerimenti:
  - Si ricordi che un char è rappresentato sia dal carattere che dal suo codice ASCII, a seconda del contesto
  - Conviene utilizzare un vettore per memorizzare la frequenza dei caratteri
  - Si può sviluppare una funzione

```
void cercamax(int n[256],int &max,int &posmax)
```

Algoritmi di ordinamento

- Dato un array di elementi con valori ordinabili, riordinarlo in ordine crescente (o decrescente)
- Tra le soluzioni possibili, vedremo:
  - Select Sort: Ordinamento per selezione iterativa dei minimi (massimi)
  - Bubble Sort: Ordinamento per “bolle”

Selection sort

- Detto anche “selezione per minimi (massimi) successivi”
  - Al primo passo si cerca il minimo (massimo) tra tutti gli elementi e lo si mette in prima posizione
  - Ad ogni passo i successivo si cerca il minimo (massimo) degli elementi non ancora ordinati (dall’i-esimo all’ultimo) e lo si mette in posizione i, fino al completamento del vettore ordinato

Selection sort

```
void SelectSort(int a[], int n){
	int i, j, posmin;
	for (i = 0; i < n-1; i++){ //dal primo al penultimo
		// trova il minimo tra i e l'ultimo
		posmin = i;
		for (j = i+1; j < n; j++)
			if (a[j] < a[posmin])
				posmin = j;
		// se non e' già al posto giusto mettilo in posizione i
		if (posmin != i){
			int temp = a[i];
			a[i] = a[posmin];
			a[posmin] = temp;
		}
	}
}
```

Bubble sort

- Detto anche ordinamento “a bolle” perchè si cerca di scambiare elementi più “leggeri” con elementi più “pesanti” al fine di metterli complessivamente in ordine

- Si innestano due cicli, uno esterno su tutti gli elementi, uno più interno sugli ultimi elementi
- Ogni volta che si trova una coppia di elementi “in disordine” li si scambia nel caso di ordinamento crescente, tali che il più grande preceda il più piccolo

Bubble sort

```
void bubbleSort (int a[], int n) {
     int i, j;
     for (i = 0; i < n-1; i++)
          for (j = 0; j < n-1-i; j++)
               if (a[j] > a[j+1]){
                    int temp = a[j+1];
                    a[j+1] = a[j];
                    a[j] = temp;
               }
}
```

Bubble sort veloce

```
void BubbleSortVeloce (int a[], int n){
      bool bubble=true;
      int i=0;
      while(i<n-1 && bubble) {
            bubble = false;
            for (j=0;j<n-1-i;j++)
                  if (a[j] > a[j+1]) {
                        scambia(a,j,j+1);
                        bubble = true;
                  }
            i++;
      }
}
```