

LABORATORIO DI ALGORITMI

Martedì 1 Marzo 2022



LABORATORIO DI ALGORITMI - 17611

- Docente: Marinella Sciortino
- CFU: 6
- Lezioni:
 - Martedì: dalle 9:30 alle 12:30
 - Mercoledì: dalle 8.30 alle 11.30
- Esami: prova scritta + progetto/prova orale
- **Propedeuticità:**
 - 05880 - PROGRAMMAZIONE E LABORATORIO C.I.
 - 16670 - ALGORITMI E STRUTTURE DATI
 - 16784 - SISTEMI OPERATIVI
 - 16450 - ARCHITETTURE DEGLI ELABORATORI
 - 16671 - INFORMATICA TEORICA

TESTI DI RIFERIMENTO E CONSULTAZIONE: VISIONARE LA SCHEDA DI TRASPARENZA

- Data Structures and Algorithms in C++, Michael T. Goodrich, Roberto Tamassia, David M. Mount, Wiley, 2011.
- Algorithms in C++, Parts 1-4: Fundamentals, Data Structure, Sorting, Searching [3rd ed.], Robert Sedgewick. Addison-Wesley Professional, 1998.
- Algorithms in C++ Part 5: Graph Algorithms, 3rd Edition. Robert Sedgewick. Addison-Wesley Professional, 2002.
- The magic of Algorithms! Lectures on some algorithmic pearls. Paolo Ferragina, Università di Pisa. 2019.
- Introduzione agli algoritmi e strutture dati (3 ed). Cormen, Leiserson, Rivest, Stein. Mc Graw Hill, 2010 (Polinomi e FFT/Polynomials and the FFT)
- Algoritmi e Strutture Dati. Demetrescu, Finocchi, Italiano. Mc Graw Hill, 2008. C++. Guida essenziale per programmatori di Bjarne Stroustrup, Pearson, 2015

Nota: Si ringraziano R. Sedgewick, K. Wayne e U. Ferraro Petrillo per il materiale didattico fornito che sarà utilizzato durante il corso.

OBIETTIVI DEL CORSO

- Focalizzare l'attenzione sugli strumenti teorici e pratici per realizzare algoritmi efficienti.
- L'attività di laboratorio verrà effettuata tramite implementazioni in linguaggio C++.
- Si assumeranno solide conoscenze di base del linguaggio C e del paradigma di programmazione ad oggetti.
- Saranno ripresi alcuni argomenti trattati durante il corso di Algoritmi e Strutture Dati, studiandone in modo approfondito i dettagli sulle strutture dati utilizzate e le tecniche relative all'implementazione.

ALGORITMI E STRUTTURE DATI

In modo informale:

- ALGORITMI: “Come” risolvere un problema
- STRUTTURA DATI: “Come” memorizzare le informazioni

Durante questo corso prenderemo in considerazione alcuni topics importanti in Computer Science con notevoli ricadute applicative, come il sorting, il searching, i grafi, etc, concentrandoci sulle implementazioni di efficienti strutture dati e algoritmi utilizzati per affrontare tali problematiche

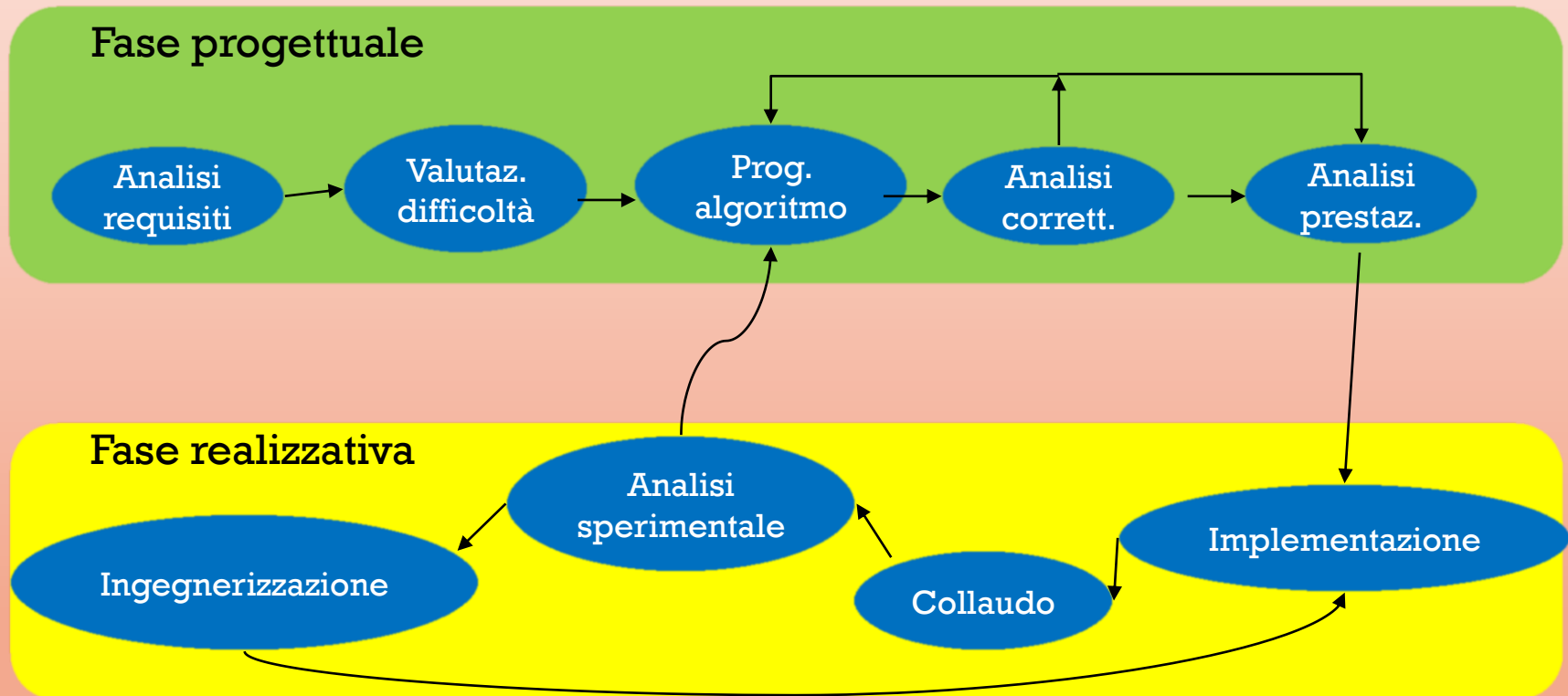
PERCHÉ STUDIARE ALGORITMI?

- Il loro impatto è molto esteso e può portare lontano. Esistono numerosi ambiti (Internet, Biologia, Computer Graphics, Sicurezza, Social Network, etc) in cui l'uso di efficienti algoritmi e appropriate strutture dati consentono di fare la differenza.
- Per diventare buoni programmatori
«I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships» - Linus Torvalds (creatore di Linux)

CICLO DI SVILUPPO DI UN ALGORITMO

- Lo sviluppo di un software robusto ed efficiente per la risoluzione di un problema è un compito complesso che richiede
 - creatività,
 - capacità di astrazione,
 - familiarità con gli strumenti matematici,
 - padronanza di un linguaggio di programmazione,
 - buone conoscenze della piattaforma di calcolo che si intende utilizzare
- Coinvolge varie fasi che si avvicendano in modo ciclico.
Esemplificando:
 - Fase progettuale
 - Fase realizzativa

FASE PROGETTUALE E FASE REALIZZATIVA



FASE PROGETTUALE

- **Analisi dei requisiti:**
 - definire in modo preciso e non ambiguo il problema di calcolo che si intende risolvere
 - Identificazione dei dati di ingresso e uscita
 - Decomposizione del problema in sottoproblemi
- **Valutazione difficoltà intrinseca del problema**
 - Si esaminano le risorse di calcolo usate dall'algoritmo: tempo di esecuzione e spazio di memoria
 - Si analizzano i limiti inferiori per tali risorse (per esempio si consideri il problema dell'ordinamento di n elementi)

FASE PROGETTUALE

- Progetto di un algoritmo risolutivo: tra le varie strategie risolutive si cerca quella che usa il minor numero di risorse di calcolo
- Analisi della correttezza: si prova che qualunque sia l'input l'algoritmo fornisce l'output corretto.
- Analisi delle prestazioni in un modello di costo teorico: si analizzano con strumenti matematici le risorse di calcolo da utilizzare. Ci aiuta a scegliere tra le varie soluzioni del problema
 - Si definisce in termini di cosa si esprime il tempo di esecuzione
 - Si distinguono vari tipi di analisi: caso peggiore, caso migliore, caso medio
 - Si usa la notazione asintotica
 - Si può effettuare l'analisi ammortizzata: nel caso un algoritmo viene eseguito ripetutamente nel tempo. Si analizzano le prestazioni in media
- Eventuale progettazione di algoritmi più efficienti

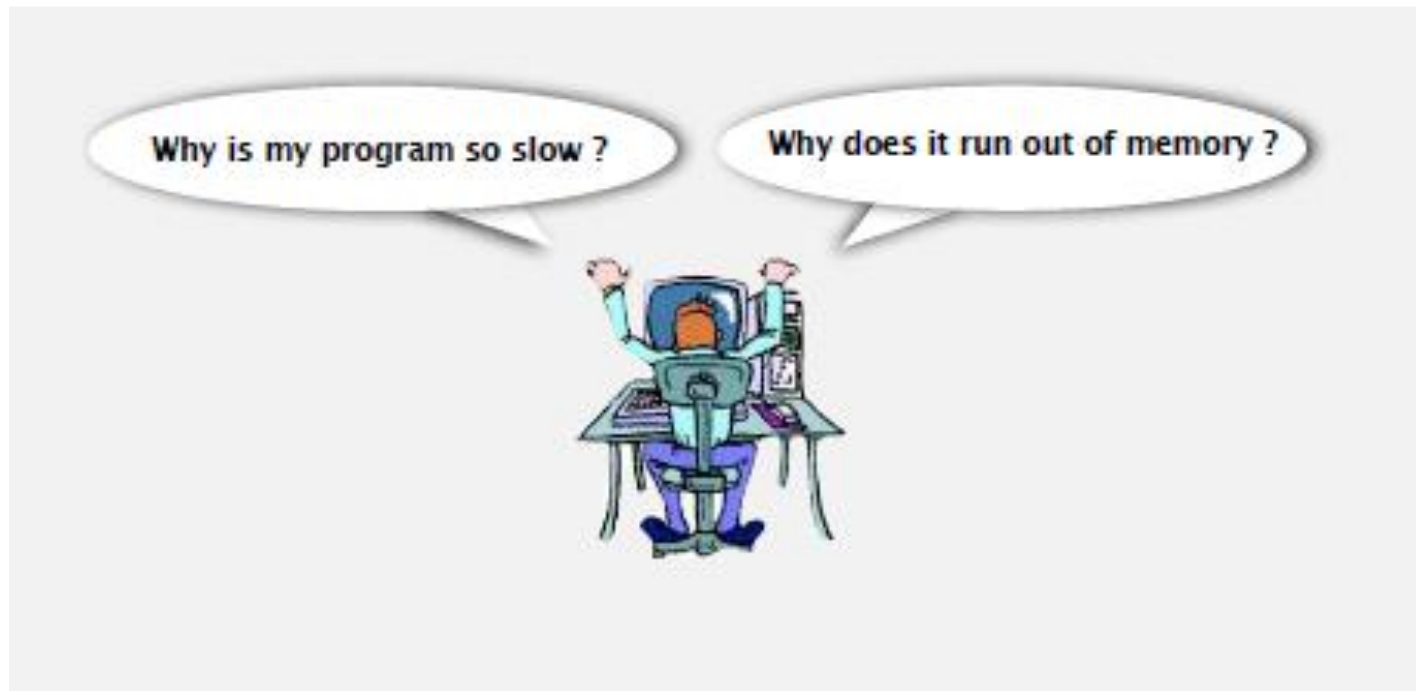
FASE REALIZZATIVA

- Implementazione in un linguaggio di programmazione: scegliere una buona implementazione e un adeguato linguaggio di programmazione può influenzare molto i tempi di esecuzione
- Collaudo e analisi sperimentale: vengono identificati eventuali errori implementativi, si analizza sperimentalmente su dati di test reali
- Ingegnerizzazione: processo richiesto per trasformare la descrizione ad alto livello in una implementazione robusta, efficiente, modulare, ben testata e facilmente riutilizzabile. Si deve tener conto anche dell'efficienza delle funzioni di libreria



COME SCEGLIERE UN ALGORITMO?

QUALI PARAMETRI USARE?



COME VALUTARE UN ALGORITMO?

- Si analizzano le risorse impiegate per risolvere il problema, in funzione della dimensione e della tipologia dell'input
- Le risorse sono di due tipi:
 - Tempo: tempo impiegato per completare l'algoritmo
 - Spazio: quantità di memoria utilizzata

COMINCIAMO CON LA COMPLESSITÀ TEMPORALE...

- In genere si esprime in funzione della dimensione dell'input, ovvero il numero di elementi che lo costituiscono
- Si può misurare in vari modi:
 - Come tempo di orologio (wall-clock time)
 - Come numero di operazioni rilevanti
 - Come numero di operazioni elementari

COMINCIAMO CON UN ESEMPIO SEMPLICE

- Problema del minimo in un vettore: misuriamo il numero di confronti
- Algoritmo ingenuo

```
min(S, n)
```

```
for i = 1 to n do
    isMin = true
    for j = 1 to n do
        if S[j] < S[i] then
            isMin = false
    if isMin then
        return S[i]
```

- Complessità di tempo: n^2 confronti

MIGLIORIAMO LA SOLUZIONE PER TROVARE IL MINIMO

$\min(S, n)$

$min = S[1]$

for $i = 2$ to n do

 if $S[i] < min$ then

$min = S[i]$

return min

Minimo parziale

Nuovo minimo
parziale

- Complessità di tempo: $n-1$ confronti
- E' un algoritmo ottimale

ALTRO PROBLEMA SEMPLICE: RICERCA IN UNA SEQUENZA ORDINATA

- Algoritmo ingenuo: n confronti

```
lookup( $S, n, v$ )  
for  $i = 1$  to  $n$  do  
    if  $S[i] = v$  then  
        return  $i$   
return 0
```

RICERCA BINARIA

- E' una soluzione più efficiente
- Richiede al più $\lfloor \log n \rfloor + 1$ confronti
- Perché?

CONSIDERIAMO UN ALTRO PROBLEMA...

- Nell'ambito dell'analisi delle sequenze genomiche, un problema di ricerca consiste nel localizzare le regioni biologicamente significative, come le regioni GC-rich. Un approccio comune consiste nell'assegnare uno score ad ogni nucleotide, per poi cercare il segmento di somma massima.
- Problema: cercare in un array la sottosequenza di somma massima
 - Input: una sequenza di interi $A[1...n]$
 - Output: la sottosequenza $A[i ... j]$ di somma massima, ovvero la sottosequenza la cui somma degli elementi dalla posizione i alla j è maggiore o uguale alla somma degli elementi di una qualsiasi altra sottosequenza

1	3	4	-9	2	3	-1	3	4	-3	10	-3	2
---	---	---	----	---	---	----	---	---	----	----	----	---

SOTTOSEQUENZA MASSIMALE

- 1977: Il problema è stato posto per la prima volta da Ulf Grenander (Brown), come versione semplificata di un problema più generale in immagini 2D (maximum likelihood in image processing)
- Jon Bentley. **Programming pearls: algorithm design techniques**. Commun. ACM 27(9):865-873. September, 1984.
“while the first algorithm we'll study takes 39 days to solve a problem of size 10,000, the final algorithm solves the same problem in just a third of a second.”

Dato un array di numeri reali, l'output è la somma massima trovata in ogni sottosequenza di elementi contigui nel vettore.

Se gli elementi sono tutti positivi, la sottosequenza massimale è l'array stesso. Se sono tutti negativi, è il vettore vuoto.

LA SOLUZIONE PIÙ SEMPLICE

```
MaxSoFar := 0.0
for L := 1 to N do
  for U := L to N do
    Sum := 0.0
    for I := L to U do
      Sum := Sum + X[I]
    /* Sum now contains the
       sum of X[L..U] */
    MaxSoFar := max(MaxSoFar, Sum)
```

- Qual è la complessità di tempo?
- E' un tempo di $O(n^3)$

SOLUZIONI PIÙ EFFICIENTI

- Si nota che la somma $X[L..U]$ si ottiene da $X[L..U-1] + X[U]$

```
MaxSoFar := 0.0
for L := 1 to N do
  Sum := 0.0
  for U := L to N do
    Sum := Sum + X[U]
    /* Sum now contains the
       sum of X[L..U] */
    MaxSoFar := max(MaxSoFar, Sum)
```

Complessità di
tempo quadratica

1	3	4	-9	2	3	-1	3	4	-3	10	-3	2
1	4	8	-1	1	4	3	6	10	7	17	14	16
	3	7	-2	0	3	2	5	9	6	16	13	15
		4	-5	-3	0	-1	2	6	3	13	10	12

USO DI ARRAY CUMULATIVI

- Un'altra soluzione computa la somma nel ciclo interno accedendo ad una struttura dati (chiamata CumArray) costruita prima che il ciclo esterno venga eseguito. L'elemento h-esimo di CumArray contiene la somma dei valori di $X[1..h]$, cosicchè la somma dei valori in $X[L..U]$ si trova computando $\text{CumArray}[U] - \text{CumArray}[L-1]$

```
CumArray[0] := 0.0
for I := 1 to N do
    CumArray[I] := CumArray[I - 1] + X[I]
MaxSoFar := 0.0
for L := 1 to N do
    for U := L to N do
        Sum := CumArray[U] - CumArray[L - 1]
        /* Sum now contains the
           sum of X[L..U] */
        MaxSoFar := max(MaxSoFar, Sum)
```

- Qual è la complessità?

APPROCCIO DIVIDE AND CONQUER

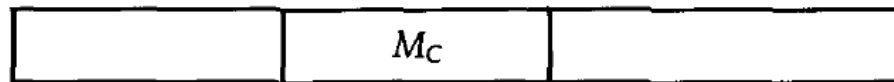
- Si divide il vettore in due parti A e B, approssimativamente della stessa taglia.



- Si trova la sottosequenza massimale in A e quella in B.



- Non basta confrontare M_A e M_B



APPROCCIO DIVIDE AND CONQUER

```
recursive function MaxSum(L, U)
  if L > U then      /* Zero-element vector */
    return 0.0
  if L = U then      /* One-element vector */
    return max(0.0, X[L])

  M := (L + U)/2      /* A is X[L..M], B is X[M + 1..U] */
  /* Find max crossing to left */
  Sum := 0.0; MaxToLeft := 0.0
  for I := M downto L do
    Sum := Sum + X[I]
    MaxToLeft := max(MaxToLeft, Sum)
  /* Find max crossing to right */
  Sum := 0.0; MaxToRight := 0.0
  for I := M + 1 to U do
    Sum := Sum + X[I]
    MaxToRight := max(MaxToRight, Sum)
  MaxCrossing := MaxToLeft + MaxToRight

  MaxInA := MaxSum(L, M)
  MaxInB := MaxSum(M + 1, U)
  return max(MaxCrossing, MaxInA, MaxInB)
```

- Qual è la complessità di tempo?

ALTRA SOLUZIONE?

- Si scansiona il vettore da sinistra a destra, tenendo traccia della sottosequenza massima vista fino a quel momento.
- Supponiamo di aver risolto il problema per $X[1..h-1]$. Come possiamo estendere la soluzione ai primi h ?
- La somma massima nei primi h elementi è o la somma massima nei primi $h-1$ elementi (MaxSoFar) o è quella di una sottosequenza che termina in I (MaxEndingHere)

	<i>MaxSoFar</i>		<i>MaxEndingHere</i>
--	-----------------	--	----------------------

```

MaxSoFar := 0.0
MaxEndingHere := 0.0
for I := 1 to N do
    MaxEndingHere := max(0.0,
                        MaxEndingHere + X[I])
    MaxSoFar := max(MaxSoFar,
                    MaxEndingHere)

```

```

      A = [ 1, 3, 4, -9, 2, 3, -1, 3, 4, -3, 10, -3, 2 ]
maxHere = 0 1 4 8 0 2 5 4 7 11 8 18 15 17
maxSoFar = 0 1 4 8 8 8 8 8 8 11 11 18 18 18

```



LINGUAGGIO C++

QUALE COMPILATORE E QUALE IDE USARE?

- Il compilatore è g++. Si può usare un text-editor e riga di comando
- IDE: Integrated development environment, ovvero ambiente di sviluppo integrato.
- Molti IDE lavorano con più linguaggi di programmazione, mentre alcuni sono costruiti per un solo linguaggio, come il C++.
- Un IDE è un ambiente di codifica che include sia un editor che una toolchain specifica per il linguaggio. Gli IDE hanno funzioni integrate come il debug, l'autocompletamento, la compilazione e l'evidenziazione della sintassi, tutte cose che rendono la programmazione più facile e veloce.
- Alcuni C++ IDE molto usati:
 - Eclipse: open-source e multi-piattaforma
 - NetBeans: open-source e multi-piattaforma
 - Visual Studio: per C++ e C# su Windows
 - CodeBlocks: open-source e multi-piattaforma

LE BASI DEL C++

- Il C++ è un'estensione del C. Quindi il programma più semplice ha solo una funzione **main**.
- Il programma è compilato con g++, che produce un eseguibile.
- Il C++ usa **while** e **for** per i loop, **if** e **switch** per i condizionali.
- Lo standard output è accessibile da **cout**. Lo standard input è accessibile da **cin**. Questi richiedono l'inclusione della libreria **iostream**.
- Il linguaggio è case-sensitive.

UN PO' DI STORIA...

- Il lavoro che ha portato **Bjarne Stroustrup** al C++ iniziò nel 1979 sotto il nome di «C with classes».
- Nel 1984 il linguaggio viene ribattezzato C++; ha acquisito il sovraccaricamento di funzioni e operatori, riferimenti e librerie dei flussi I/O e dei numeri complessi, ...
- 1985: primo rilascio commerciale del C++.
- 1998: standard ISO del C++
- 2003: Revisione dello standard ISO
- 2011: Viene approvato lo standard ISO C++11
- 2017: Dopo una revisione minore nel 2014, viene pubblicata l'ultima versione dello standard (nota informalmente come C++17).

UN PRIMO ESEMPIO DI PROGRAMMA IN C++

```
1 #include <cstdlib>
2 #include <iostream>
3 /* This program inputs two numbers x and y and outputs their sum */
4 int main( ) {
5     int x, y;
6     std::cout << "Please enter two numbers: ";
7     std::cin >> x >> y;           // input x and y
8     int sum = x + y;              // compute their sum
9     std::cout << "Their sum is " << sum << std::endl;
10    return EXIT_SUCCESS;          // terminate successfully
11 }
```

- I commenti su una linea sono indicati con doppio slash (//). Commenti più lunghi sono racchiusi tra /* e */ (linea 3).
- Gli operatori “>>” e “<<” sono usati per input e output, rispettivamente.
- Le linee 1 e 2 indicano l’uso di header files, “cstdlib” e “iostream.” IL primo fornisce alcune definizioni standard (come EXIT_SUCCESS che è uguale a 0), l’altro alcune definizioni per gestire input e output.
- std::cout, std::cin e std::cerr indicano tre importanti I/O stream del C++
- Il prefisso “std::” indica che questi oggetti provengono dalla *standard library*.

PIÙ SEMPLICEMENTE...

- E' possibile informare il compilatore che si intendono usare gli oggetti della standard library mediante l'istruzione **using**

```
#include <iostream>
using namespace std;           // makes std:: available
// ...
cout << "Please enter two numbers: "; // (std:: is not needed)
cin >> x >> y;
```

PRINCIPALI DIFFERENZE SINTATTICHE CON IL LINGUAGGIO C

```
#include <stdio.h>
main()
{
    int dim,i;
    float a, tot=0;
    /* Chiedo la dimensione del problema */
    printf("Introduci il numero di \
    elementi da sommare\n");
    scanf("%d",&dim);
    printf("Introduci gli elementi\n");
    for (i=0;i<dim;i++)
    {
        scanf("%d",&a);
        tot = tot + a;
    }
    /* Stampo il risultato */
    printf("Il risultato è %f",tot);
    return 0;
}
```

C

```
#include <iostream.h>
main()
{
    int dim;
    // Chiedo la dimensione del problema
    cout << "Introduci il numero di \
    elementi da sommare" << endl;
    cin >> dim;
    float tot=0,a;
    cout<<"Introduci gli elementi"<<endl;
    for (int i=0 ; i< dim ; i++)
    {
        cin >> a;
        tot = tot + a;
    }
    // Stampo il risultato
    cout<<"Il risultato è "<<tot<<endl;
}
```

C++

using namespace std;

DAL C AL C++

Commenti

- In C++ viene introdotto un nuovo tipo di commento mediante `//` che delimita l'inizio di un commento che finisce al termine della linea

```
int dim;
```

```
// Chiedo la dimensione del problema
```

```
cout << "Introduci il numero di \
```

Input ed Output

- L'header file per la gestione dell'input ed output `stdio.h` viene sostituito in C++ da `iostream`
- In C++ al posto delle funzioni definite da `stdio.h`, `printf` e `scanf` si usano `cin` e `cout` con gli operatori `" >> "` e `" << "`

```
cout << "Introduci il numero di \
```

```
elementi da sommare" << endl;
```

```
cin >> dim ;
```

- `endl` viene utilizzato per indicare il carattere di terminazione della linea in alternativa a `('\\n')`

DAL C AL C++

Input ed Output

- La fase di input viene così gestita:
 - In C: `scanf("%s",s);`
 - In C++: `cin >> s;`
- La fase di output viene così gestita:
 - In C: `printf("Ciao %s\n",s);`
 - In C++: `cout << "Ciao" << s << endl;`

Dichiarazioni delle variabili locali

- Le variabili possono essere dichiarate in qualsiasi punto del programma e non devono essere necessariamente elencate all'inizio.

...

// Chiedo di inserire gli elementi

float a, tot=0;

cout<<"Introduci gli elementi"<<endl;

...

- è buona norma dichiarare una variabile lì dove si comincia ad usarla poiché questo rende il codice più leggibile

DAL C AL C++

Blocchi di istruzioni

- Le istruzioni di un programma possono essere raggruppate in blocchi. I blocchi sono delimitati tra parentesi graffe

Visibilità delle variabili

- Una variabile esiste dal momento della dichiarazione fino alla chiusura del blocco di istruzioni in cui è stata dichiarata
- Ogni variabile è accessibile soltanto nel blocco in cui è dichiarata ed in tutti i blocchi interni **ad esso**
- Nella figura mostrata la variabile `a` non è più visibile all'esterno del riquadro.

```
...  
float tot=0  
{  
    float a;  
    cout<<"Introduci gli elementi"<<endl;  
    for (int i=0 ; i< dim ; i++)  
    {  
        cin >> a;  
        tot = tot + a;  
    }  
}  
...
```

DA C AL C++

Dichiarazione della variabili nei cicli

- Le variabili possono essere dichiarate anche all'interno di un loop. In questo modo la variabile viene allocata soltanto durante l'esecuzione del ciclo
- Una volta concluso il ciclo la variabile non è più accessibile
- Nella figura mostrata la variabile i non è più visibile all'esterno del ciclo.

```
...  
for (int i=0 ; i< dim ; i++)  
{  
    cin >> a;  
    tot = tot + a;  
}  
...
```

Variabili con lo stesso nome

- Una conseguenza della visibilità è la possibilità di utilizzare un stesso nome di variabile in più blocchi potendolo associare anche a diversi tipi

```
int X = 5;  
... // qui è visibile int X  
{  
    ... // qui è visibile int X  
    char X = 'a'; // ora è visibile char X  
    ... // qui è visibile char X  
} // qui ritorna visibile int X  
...
```

DAL C AL C++

Allocazione dinamica

- Anche in C++ come in C è possibile allocare e liberare locazione di memoria in base alle necessità.
- A tale scopo vengono utilizzate le funzioni **new** e **delete** che rimpiazzano rispettivamente **malloc** e **free**

Sintassi di new

- **new** si utilizza per l'allocazione e l'inizializzazione di variabili con la sintassi:

```
char *ptr;  
...  
ptr= new char('c');
```

- e per l'allocazione di array con la sintassi:

```
float *arr;  
...  
arr= new float[dim];
```

Sintassi di delete

- **delete** si utilizza per rilasciare l'aria di memoria di una variabile allocata tramite new con la sintassi:

```
ptr= new char('c');  
...  
delete ptr;
```

- Per liberare un array invece è necessario utilizzare la sintassi:

```
arr= new float[dim];  
...  
delete [ ] arr;
```

- A differenza di quanto succedeva con **malloc**, **new** associa una quantità di memoria della dimensione appropriata senza dover utilizzare **sizeof**

```
...  
char *ptr;  
float *arr;  
int dim=100;  
ptr=(char *)malloc(sizeof(char));  
*ptr='c'  
free(ptr);  
arr=(float*)malloc(sizeof(float)*dim);  
free(arr);  
...
```

Non serve in
casting in
ANSI C



```
...  
char *ptr;  
float *arr;  
int dim=100;  
ptr= new char('c'); //alloca ed inizializza  
delete ptr;  
arr= new float[dim]; //alloca un array  
delete [ ] arr;  
...
```

NOTE SULL'ALLOCAZIONE DINAMICA

Quando si utilizza l'allocazione dinamica è bene rispettare alcune regole basilari:

- È buona pratica non utilizzare nello stesso programma allocazioni fatte con `new` e `malloc`
- Le allocazioni prodotte da `new` non possono essere liberate mediante `free`
- Deallocare un array con `delete` invece di `delete[]` genera un errore
- In generale, sebbene sia possibile utilizzare la sintassi del C all'interno dei programmi C++ è buona regola utilizzare i costrutti del C++ perché sono estensibili e effettuano controlli sui tipi

ALCUNI TIPI IN C++

- `bool` Boolean value, true or false
- `char` character
- `short` short integer
- `int` integer
- `long` long integer
- `float` single-precision floating-point number
- `double` double-precision floating-point number
- `enum` enumeration per rappresentare un insieme di valori discreti
 - `enum Day { SUN, MON, TUE, WED, THU, FRI, SAT };` // se non divers. specificato `SUN=0, MON=1,...`
 - `enum Mood { HAPPY = 3, SAD = 1, ANXIOUS = 4, SLEEPY = 2 };`
 - `Day today = THU;` // `today` può essere uno qualsiasi tra `MON . . . SAT`
 - `Mood myMood = SLEEPY;` // `myMood` può essere `HAPPY, . . ., SLEEPY`
- `void`, assenza di informazioni di tipo

SEMPLICE ESEMPIO

- Stampare i numeri primi minori di 100

```
#include <iostream>
using namespace std;

bool isPrime(int x);
const int MAX=100;

int main( )
{
    int test=2;
    while( test<MAX ) {
        if( isPrime(test) )    // note we do not compare it with true
            cout << test << " ";
        test++;
    }
    cout << endl;
    return 0;
}

bool isPrime(int x)
{
    for(int y=2; y<x; y++) {
        if( x%y==0 )
            return false;
    }
    return true;
}
```

ALTRO SEMPLICE ESEMPIO

■ Ricerca binaria in un array ordinato

```
#include <iostream>
using namespace std;

int binarySearch( double *arr, int size, double item)
{
    int begin = 0;
    int end = size;
    // invariant: unsearched space is begin, begin+1, ..., end-1
    while( begin < end ) {
        int middle = (begin+end)/2;
        if( item == arr[middle] )
            return middle;
        else if( item > arr[middle] )
            begin = middle+1;
        else
            end = middle;
    }
    return -1;
}

int main( )
{
    double A[] = { 2.0, 3.0, 5.0, 8.0, 13.0, 21.0};
    cout << "Position of 3.14 is " << binarySearch( A, 6, 3.14) << endl;
    cout << "Position of 13.0 is " << binarySearch( A, 6, 13.0) << endl;
    return 0;
}
```

STRINGHE

- Una stringa come "Hello World", può essere rappresentata come un array di caratteri di lunghezza fissata che termina con `\0` (NULL). Le stringhe di caratteri rappresentate in questo modo sono chiamate C-style strings, poiché sono state ereditate dal C.
Sfortunatamente, questa rappresentazione da sola non fornisce molte operazioni sulle stringhe, come la concatenazione e il confronto.
- C++ fornisce un tipo di stringa come parte della sua Standard Template Standard (STL) library. Quando abbiamo bisogno di distinguere, chiamiamo queste STL-strings.
- Per utilizzare le stringhe STL è necessario includere il file header `<string>`. Poiché le STL-strings fanno parte dello standard namespace, il loro nome completo è `std::string`.
- Aggiungendo la dichiarazione `"using std::string;"` informiamo il compilatore che vogliamo accedere direttamente a questa definizione, quindi possiamo omettere il prefisso `"std::"`.
- Le STL-strings possono essere concatenate usando l'operatore `+`, possono essere confrontate tra loro usando l'ordine lessicografico (o dizionario), e possono essere date in input e output usando gli operatori `>>` e `<<`, rispettivamente.

```
#include <string>
using std::string;
// ...
string s = "to be";
string t = "not " + s;
string u = s + " or " + t;
if (s > t)
    cout << u;

// t = "not to be"
// u = "to be or not to be"
// true: "to be" > "not to be"
// outputs "to be or not to be"
```

RIFERIMENTO

- I puntatori forniscono un modo per riferirsi indirettamente ad un oggetto. Un altro modo è attraverso riferimenti. Un **riferimento** è semplicemente un nome alternativo per un oggetto. Dato un tipo T, la notazione T& indica un riferimento ad un oggetto di tipo T. Un riferimento va considerato come un secondo nome per la variabile.
- A differenza dei puntatori, che possono essere NULL, un riferimento in C++ deve riferirsi ad una variabile reale. Quando un riferimento è dichiarato, il suo valore deve essere inizializzato. In seguito, qualsiasi accesso al riferimento è trattato esattamente come se fosse un accesso all'oggetto sottostante.
- Ad esempio

`int ix; //ix è una variabile reale`

`int &rx=ix; //rx è un riferimento per ix`

`ix =1; // anche rx vale 1`

`rx =2; // anche ix vale 2`

- Ogni volta che si accede ad un riferimento si accede all'area di memoria associata alla variabile

```
string author = "Samuel Clemens";  
string& penName = author;           // penName is an alias for author  
penName = "Mark Twain";             // now author = "Mark Twain"  
cout << author;                     // outputs "Mark Twain"
```

PASSAGGIO PER RIFERIMENTO

- Generalmente i riferimenti vengono utilizzati nelle dichiarazioni (prototipi) delle funzioni perché risolvono i problemi del passaggio per valore del C
- Con il passaggio per riferimento all'interno della funzione si utilizzano direttamente i parametri della dichiarazione senza usare l'operatore * come nel C
- Nella chiamata della funzione scambia in C++ non sarà più necessario utilizzare l'operatore & come è usuale in C

```
void swap( int &a, int &b )  
{  
  int t;  
  t = a;  
  a = b;  
  b = t;  
}
```

C++

```
void swap( int *a, int *b )  
{  
  int t;  
  t = *a;  
  *a = *b;  
  *b = t;  
}
```

C

```
...  
int x,y;  
swap(x,y)  
...
```

Main

```
...  
int x,y;  
swap(&x,&y)  
...
```

OVERLOADING

- Overloading vuol dire che funzioni o operatori hanno lo stesso nome, ma argomenti o operandi diversi e il cui effetto dipende dai tipi dei loro operandi o argomenti reali.

Overloading di funzioni

In C++ è possibile definire più funzioni con lo stesso nome purché accettino diversi argomenti in input.

- Facciamo un esempio definiamo tre funzioni **isnull** che accettano come argomento rispettivamente:
 - un intero
 - un reale (float)
 - un reale a doppia precisione (double)
- Questo risulta molto utile nel caso di funzioni hanno fini simili ma compiono diverse operazioni perché operano su argomenti di tipo diverso. Al momento della chiamata il compilatore stabilisce quale funzione invocare in base all'argomento. Ad esempio

```
...  
int i,ai;  
float f,af;  
double d,ad;
```

```
...  
ai=isnull(i); // esegue le funzione 1  
af=isnull(f); // esegue la funzione 2  
ad=isnull(d); // esegue la funzione 3
```

```
// funzione 1  
bool isnull(int x) {  
    if ( x == 0 )  
        return true;  
    return false;  
}
```

```
// funzione 2  
bool isnull(float x) {  
    if ( x < 1e-10 )  
        return true;  
    return false;  
}
```

```
// funzione 3  
bool isnull(double x) {  
    if ( x < 1e-100 )  
        return true;  
    return false;  
}
```

OVERLOADING

Overloading di operatori

- Il C++ permette anche l'overloading degli operatori, come +, *, += e <<.
- Supponiamo di voler scrivere un test di uguaglianza per due elementi di tipo Enum. Possiamo denotare questo in un modo modo naturale sovraccaricando l'operatore == come mostrato di seguito.

```
typedef enum Day {SUN, MON, TUE,  
WED, THU, FRI, SAT } Giorno;  
bool operator==(Giorno x, Giorno y){  
    if (y==0) return true;  
    else  
        return false;  
}  
...  
Giorno g=SUN,h=TUE;  
if (g==h) cout << "minore";
```


FUNZIONI INLINE


- Le funzioni molto brevi possono essere definite come "inline".
- Questo è un suggerimento per il compilatore che dovrebbe semplicemente espandere il codice della funzione sul posto, piuttosto che usare il meccanismo di call-return del sistema. Come regola generale, le funzioni in linea dovrebbero essere molto brevi (al massimo poche righe) e non dovrebbero coinvolgere alcun ciclo o condizionale.
- Questo può comportare un miglioramento delle performance del programma a scapito della grandezza dell'eseguibile prodotto

```
inline int min(int x, int y) { return (x < y ? x : y); }
```

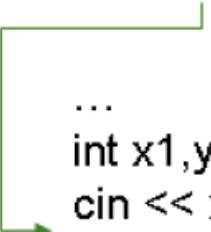
ALTRO ESEMPIO

```
..  
int x1,y2;  
cin << x1;  
cin << y2;  
c=dist(x1,y2)
```

```
...  
      inline dist(x,y){  
      {  
      return sqrt(x*x+y*y);  
      }  
      }
```



```
      ...  
      int x1,y2;  
      cin << x1;  
      cin << y2;  
      c= sqrt(x1,y2)  
      ...
```



ESERCIZI

1. Scrivere un programma in C++ che
 - Calcoli la trasposta di una matrice quadrata
 - Visualizzi tale matrice
2. Implementare in C++ gli algoritmi per calcolare la sottosequenza massimale di un vettore, allocato dinamicamente.