



UNIVERSITÀ  
DEGLI STUDI  
DI BERGAMO

# Informatica

## Modulo di Programmazione

INFORMATICA  
MODULO DI PROGRAMMAZIONE  
FUNCTIONS

[mauro.pelucchi@gmail.com](mailto:mauro.pelucchi@gmail.com)

Mauro Pelucchi

2023/2024

# Agenda

- Functions

# Esercizio 1

Scrivere un programma in linguaggio C che lette dallo standard input due sequenze vettoriali ordinate di interi  $V1[n]$ ,  $V2[m]$  ne crei una terza  $V3[n+m]$  anch'essa ordinata, che contenga tutti gli elementi di  $V1$  e di  $V2$ .

Esempio:  $V1 = \langle 1, 2, 5, 6 \rangle$ ;  $V2 = \langle 3, 4, 5, 7, 9 \rangle$ ; allora  $V3 = \langle 1, 2, 3, 4, 5, 5, 6, 7, 9 \rangle$

# Esercizio 1

Idea di soluzione:

si suppone l'array  $V3$  vuoto inizializzando un indice  $k$  all'inizio di quest' ultimo e si inizializzano due indici  $i, j$  rispettivamente all'inizio dei vettori  $V1$  e  $V2$ .

Si entra quindi in un ciclo in cui si ricopia nel terzo vettore  $V3$  il minore tra  $V1[i]$  e  $V2[j]$ , incrementando l'indice corrispondente. Il ciclo termina quando una delle due sequenze  $V1$  o  $V2$  è terminata, dopodiché si procede con l'accodare in  $V3$  la restante parte del vettore rimasto.

# Esercizio 2

Data una sequenza vettoriale  $V[n]$  permutare i suoi elementi in modo che risulti ordinata in senso non decrescente:

$$V[0] \leq V[1] \leq V[2] \leq \dots \leq V[n-1].$$

# Esercizio 2

Metodo di Soluzione: **Ordinamento per selezione**

L'idea è quella di trovare il minimo fra gli elementi nella sequenza e scambiare con l'elemento al primo posto.

Trovare il minimo fra gli elementi tra il secondo posto e l'ultimo, e scambiare con il secondo e così via. Il metodo è chiamato "per selezione" perché si basa sulla ripetizione della selezione dell'elemento minore, tra quelli rimasti da ordinare.

# Esercizio 2

Metodo di Soluzione: **Ordinamento Bubblesort**


L'idea di questo algoritmo è quella di confrontare i primi due elementi e se non sono ordinati li si scambia, poi si confrontano il secondo e il terzo e se non sono ordinati li si scambia e così via sino a confrontare penultimo e ultimo elemento.

Dopo aver scandito una prima volta tutto il vettore si è sicuri che l'elemento maggiore è nella cella più a destra, quindi si comincia un nuovo ciclo che confronta ancora a due a due le celle dalla prima all' ultima.

Se  $n$  è il numero di elementi del vettore si itera questo processo di scansione per  $n-1$  volte al termine delle quali il vettore risulterà completamente ordinato.



# First pass

									
54	26	93	17	77	31	44	55	20	Exchange
26	54	93	17	77	31	44	55	20	No Exchange
26	54	93	17	77	31	44	55	20	Exchange
26	54	17	93	77	31	44	55	20	Exchange
26	54	17	77	93	31	44	55	20	Exchange
26	54	17	77	31	93	44	55	20	Exchange
26	54	17	77	31	44	93	55	20	Exchange
26	54	17	77	31	44	55	93	20	Exchange
26	54	17	77	31	44	55	20	93	93 in place after first pass



# Esercizio 2

$V = [ 8 \ 5 \ 3 \ 0 \ 6 ]$        $\text{inizio } i = 0, j = 0$   
 $V = [ 5 \ 8 \ 3 \ 0 \ 6 ]$        $j = 1$   
 $V = [ 5 \ 3 \ 8 \ 0 \ 6 ]$        $j = 2$   
 $V = [ 5 \ 3 \ 0 \ 8 \ 6 ]$        $j = 3$

$V = [ 5 \ 3 \ 0 \ 6 \ 8 ]$        $i = 1, j = 0$   
 $V = [ 3 \ 5 \ 0 \ 6 \ 8 ]$        $j = 1$   
 $V = [ 3 \ 0 \ 5 \ 6 \ 8 ]$        $j = 2$   
 **$V = [ 3 \ 0 \ 5 \ 6 \ 8 ]$**        **$j = 3$**

$V = [ 3 \ 0 \ 5 \ 6 \ 8 ]$        $i = 2, j = 0$   
 $V = [ 0 \ 3 \ 5 \ 6 \ 8 ]$        $j = 1$   
 **$V = [ 0 \ 3 \ 5 \ 6 \ 8 ]$**        **$j = 2$**   
 **$V = [ 0 \ 3 \ 5 \ 6 \ 8 ]$**        **$j = 3$**

$V = [ 0 \ 3 \ 5 \ 6 \ 8 ]$        $i = 3, j = 0$   
 **$V = [ 0 \ 3 \ 5 \ 6 \ 8 ]$**        **$j = 1$**   
 **$V = [ 0 \ 3 \ 5 \ 6 \ 8 ]$**        **$j = 2$**   
 **$V = [ 0 \ 3 \ 5 \ 6 \ 8 ]$**        **$j = 3$**

# Functions

- Functions
  - Modularize a program
  - Software reusability
  - Call function multiple times
- Local variables
  - Known only in the function in which they are defined
  - All variables declared in function definitions are local variables
- Parameters
  - Local variables passed to function when called
  - Provide outside information

# Function Definitions

- Function prototype
  - Tells compiler argument type and return type of function
  - **int square( int );**
    - Function takes an **int** and returns an **int**
- Calling/invoking a function
  - **square(x);**
- Format for function definition

```
return-value-type function-name ( parameter-list )
{
    declarations and statements
}
```
- Prototype must match function definition
  - Function prototype

```
double maximum( double, double, double );
```
  - Definition

```
double maximum( double x, double y, double z )
{
    ...
}
```

# Function Definitions

- Example function

```
int square( int y )  
{  
    return y * y;  
}
```

- **return** keyword

- Returns data, and control goes to function's caller
  - If no data to return, use **return**;
- Function ends when reaches right brace
  - Control goes to caller

- Functions cannot be defined inside other functions

# Function Prototypes

- Function signature

- Part of prototype with name and parameters

- `double maximum( double, double, double );`

Function signature

- Argument Coercion

- Force arguments to be of proper type

- Converting `int` (4) to `double` (4.0)

- `cout << sqrt(4)`

- Conversion rules

- Arguments usually converted automatically

- Changing from `double` to `int` can truncate data

- 3.4 to 3

- Mixed type goes to highest type (promotion)

- `int * double`

```
1
2 // A scoping example.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 void useLocal( void );
9 void useStaticLocal( void );
10 void useGlobal( void );
11
12 int x = 1; // global variable
13
14 int main()
15 {
16     int x = 5; // local variable to main
17
18     cout << "local x in main's out
19 { // start new scope
20
21     int x = 7;
22
23     cout << "local x in main's inner scope is " << x << endl;
24
25 } // end new scope
26
```

Declared outside of function;  
global variable with file  
scope.

Local variable with function  
scope.

Create a new block, giving **x**  
block scope. When the block  
ends, this **x** is destroyed.



# Esercizio 3

## Definizione funzioni–passaggio parametri per copia

Scrivere una funzione per definire se un numero è primo e un programma principale minimale che ne testa la funzionalità.

# References and Reference Parameters

- Call by value
  - Copy of data passed to function
  - Changes to copy do not change original
  - Prevent unwanted side effects
- Call by reference
  - Function can directly access data
  - Changes affect original
- Reference parameter
  - Alias for argument in function call
    - Passes parameter by reference
  - Use **&** after data type in prototype
    - **void myFunction( int &data )**
    - Read “**data** is a reference to an **int**”
  - Function call format the same
    - However, original can now be changed



# Esercizio 4

Simulare l'esecuzione di ciascuna delle 4 chiamate a sottoprogrammi, limitandosi alla descrizione dei valori stampati tramite `cout << .`

Variabili visibili dal main:	varA	varB	varC
situazione iniziale	12	25	63
dopo <code>varC = proc1(&amp;varA);</code>	5	25	23

```

int proc1(int* par) {
    *par = *par - 7;
    return(23);
}

```

dopo <code>proc2(&amp;varA, varC);</code>	23	25	23
---	----	----	----

```

void proc2(int* varX, int varY) {
    int varB;

    *varX = varY;
    varB = *varX + varY;
}

```

dopo <code>varB = proc3(varC);</code>	23	40	23
---------------------------------------	----	----	----

```

int proc3(int varA) {
    int varZ = 17;

    varA = varA + varZ;
    return(varA);
}

```

dopo <code>varB = proc1(&amp;varB);</code>	23	23	23
--	----	----	----

```

int proc1(int* par) {
    *par = *par - 7;
    return(23);
}

```

## Esercizio 5

### Definizione funzioni – passaggio parametri [by value & by reference]

Si individui l'errore presente nel seguente programma.

Si corregga l'errore lasciando invariata l'intestazione del sottoprogramma funz.

Si indichino i valori stampati dall'istruzione `cout << ...` del programma principale per effetto dell'esecuzione del programma corretto, nell'ipotesi che il valore letto dalla `cin >> c` sia 20, motivando la risposta.

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

```
void funz(int, int&);
```

```
int main( ){
```

```
    int e, f;
```

```
    f = 10;
```

```
    funz(f, e);
```

```
    cout << "f = " << f << "    e = " << e;
```

```
    system("PAUSE");
```

```
    return EXIT_SUCCESS;
```

```
}
```

```
void funz(int a, int b) {
```

```
    int c;
```

```
    cin >> c;
```

```
    b = a+c;
```

```
    a = b;
```

```
}
```

# Esercizio 6

## Array monodimensionali come parametri

Si consideri un array  $v[\dots]$  di  $n$  interi.  
Si vuole scrivere un sottoprogramma per "esplorare" il vettore, iniziando dal primo elemento  $v[0]$  e muovendosi con la regola descritta in seguito.  
Il sottoprogramma restituisce 0 e modifica il vettore inserendo in ogni cella il valore 0 se l'esplorazione porta ad un percorso "infinito", perché il vettore è ciclico, altrimenti restituisce 1 e non modifica il vettore se a un certo punto termina.

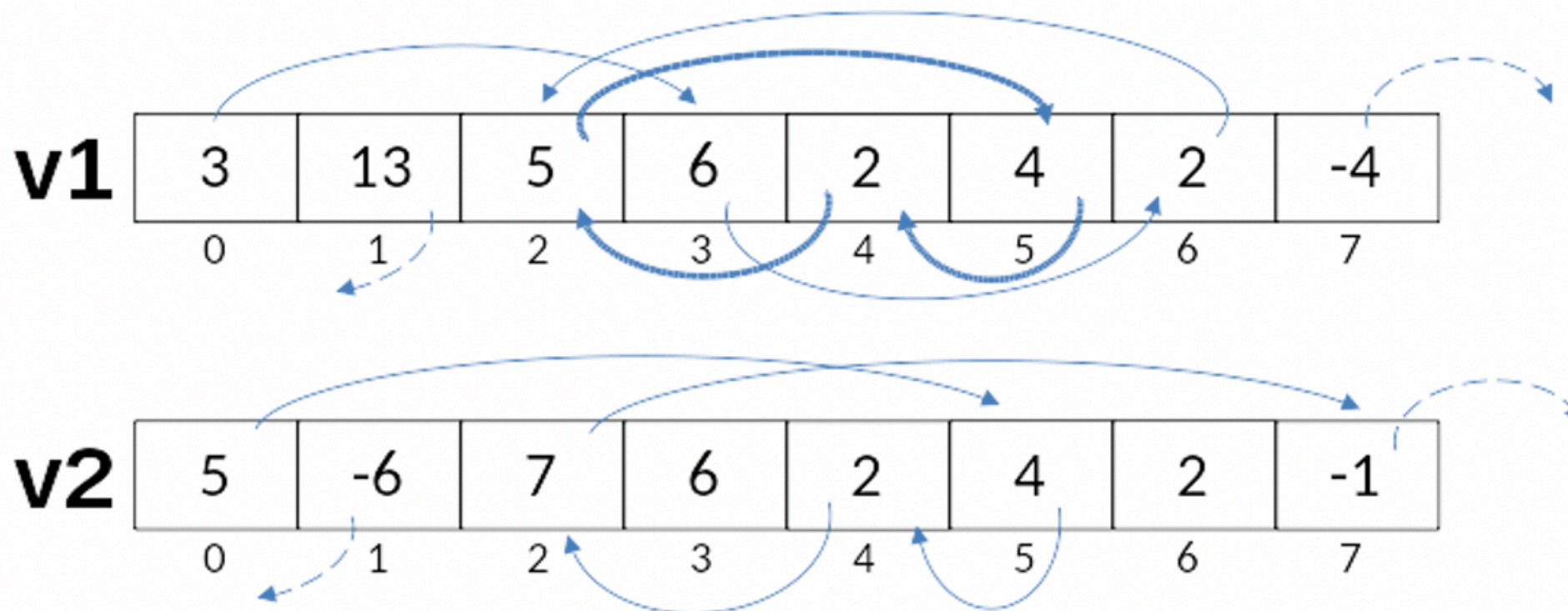
# Esercizio 6

## Array monodimensionali come parametri

Si consideri un array  $v[\dots]$  di  $n$  interi.  
Si vuole scrivere un sottoprogramma per "esplorare" il vettore, iniziando dal primo elemento  $v[0]$  e muovendosi con la regola descritta in seguito.  
Il sottoprogramma restituisce 0 e modifica il vettore inserendo in ogni cella il valore 0 se l'esplorazione porta ad un percorso "infinito", perché il vettore è ciclico, altrimenti restituisce 1 e non modifica il vettore se a un certo punto termina.



il valore contenuto in ogni elemento, se compreso tra  $0$  e  $n-1$ , rappresenta l'indice del prossimo elemento da visitare; se invece è esterno a tale intervallo rappresenta un riferimento "errato", che indica una posizione esterna al vettore, e comporta la fine del processo di esplorazione. Nell'esempio  $v1[\dots]$  è ciclico,  $v2[\dots]$  non lo è.



# Esercizio 7

Tratto dalla Prima Prova in itinere a. a. 2011-2012

Si codifichi un programma C++ che legge da tastiera (lo standard input) una sequenza (di lunghezza arbitraria) di interi positivi nell'intervallo  $[1, M]$  ( $M$  arbitrariamente scelto dal programmatore); la lettura della sequenza termina inserendo il valore 0 e, al termine della sequenza, visualizza sullo schermo (lo standard output) un messaggio che riporta la Moda della sequenza inserita.

La Moda è il valore della sequenza con la frequenza più alta (in caso di più di un valore con la frequenza più alta, si scelga il valore più piccolo).

1. Scrivere una funzione `... Moda( ... )`, che riceve un vettore di interi di lunghezza  $M+1$  contenente le frequenze di ciascun valore intero, la quale restituisce il valore della Moda.
2. Scrivere il programma principale per rispondere alla specifica precedentemente descritta usando la funzione `Moda`.



# Esercizio 8

## Definizione funzioni con parametri di tipo vettore + algoritmi

Data una sequenza di numeri interi, positivi e negativi, in valore assoluto minori di MAX\_NUM, si vuole sapere qual è la porzione di sequenza (fatta di elementi consecutivi) che rende massima la somma dei suoi elementi.

Scrivere un sotto-programma che restituisca le posizioni d'inizio e di fine della porzione di sequenza individuata e restituisca anche il valore della somma degli interi nella sottosequenza.

```
v = { 10, -21, 17, 16, 2, -37, -4, 17, -10, 12 }
```

```
sx: 3a, dx: 5a, max == 35;
```