

Corso “Programmazione 1”

Capitolo 00: Presentazione del Corso

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato@unitn.it¹

¹ In via di definizione

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL:

<https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 14 settembre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Avviso importante: Iscrizione All'esame

Gli studenti iscritti al primo anno a INF nell'AA 2020/21.

Gli studenti iscritti al primo anno a INF o ICE nell'AA 2019/20 oppure a INF, IngCOM e IngORG nell'AA 2017/18 o 2018/19 sono tenuti ad attenersi al programma di e a sostenere l'esame con:

- Prof. RICCARDI (corso parallelo a questo) **se matricola dispari**,
- Prof. ROVERI (questo corso) **se matricola pari**.

Gli studenti iscritti al primo anno nell'AA 2016/17 o precedenti sono tenuti ad attenersi al programma di e a sostenere l'esame con:

- Prof. RICCARDI (corso parallelo a questo) **se iscritti a IngCOM o IngORG**
- Prof. ROVERI (questo corso) **se iscritti ad INF**.

Si ricorda che ESSE3 consentirà l'iscrizione all'esame secondo le regole riportate qui sopra, **applicate al corso indicato nel proprio libretto elettronico**. Se in dubbio, si consiglia quindi di controllare tale informazione prima di prepararsi per l'esame.

Informazioni Utili

- PERIODO: I Semestre, 14/09/2020 \Rightarrow 21/12/2020
- MODALITÁ: “**Blended**”
- DURATA: \approx 14 settimane
- Ore (accademiche) lezione: \approx 60
- Ore (accademiche) di esercitazione: \approx 40
- CREDITI: 12

Informazioni Utili: Lezione “Teoria”

- In presenza:
 - Aula T2 - Mesiano
 - Tutti i lunedì 11:30-13:30
 - Tutti i mercoledì 15:30-18:30
- On line via Zoom:
 - Topic: Programmazione 1
 - Join Zoom Meeting: <https://unitn.zoom.us/j/92225974809>
 - Meeting ID: 922 2597 4809
 - Passcode: 375001
- Per ora accesso libero, tra poco solo con email studenti.unitn.it

Informazioni Utili: Laboratorio

- In presenza:

- Gruppo A-P:

- Aula B106 - Povo 2
 - Tutti i martedì 8:30-10:30
 - Tutti i giovedì 11:30-13:30

- Gruppo Q-Z:

- Aula B106 - Povo 2
 - Tutti i martedì 11:30-13:30
 - Tutti i giovedì 8:30-10:30

I gruppi sono fissi e non possono essere cambiati.

Accesso al laboratorio solo se preventivamente autorizzati.

Per gli altri, accesso da remoto via zoom.

- On line via zoom (no gruppi, stesso link della teoria)

- Tutti i martedì 8:30-10:30
 - Tutti i giovedì 8:30-10:30

Informazioni Utili: Laboratorio

- In presenza:

- Gruppo A-P:

- Aula B106 - Povo 2
 - Tutti i martedì 8:30-10:30
 - Tutti i giovedì 11:30-13:30

- Gruppo Q-Z:

- Aula B106 - Povo 2
 - Tutti i martedì 11:30-13:30
 - Tutti i giovedì 8:30-10:30

I gruppi sono fissi e non possono essere cambiati.

Accesso al laboratorio solo se preventivamente autorizzati.

Per gli altri, accesso da remoto via zoom.

- On line via zoom (no gruppi, stesso link della teoria)

- Tutti i martedì 8:30-10:30
 - Tutti i giovedì 8:30-10:30

- **Sito Web del corso**

<https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

- **Piattaforma Moodle (materiale corso + registrazioni)**

<https://webapps.unitn.it/gestionecorsi/>

- **Esse3**

<https://www.esse3.unitn.it/>

Ricordatevi di iscrivervi al corso sulla piattaforma Moodle!

- Insegnare i fondamenti della programmazione imperativa.
- Come linguaggio, viene usato (un sottinsieme stretto del) C++, **visto prettamente come linguaggio di programmazione imperativo**
 - \Rightarrow in questo corso aspetti di programmazione Object-Oriented non verranno analizzati (o verranno analizzati solo marginalmente).

Coding is not Programming

Coding

- I computer “capiscono” Codice Macchina (si veda il corso di Calcolatori), di difficile comprensione dagli esseri umani:
 - Linguaggi di Programmazione.
- Coding è il processo di usare un linguaggio di programmazione per fare sì che un computer faccia quello che lo sviluppatore vuole.
- In parole semplici, coding consiste in scrivere molteplici linee di codice per creare un programma software (application, game, or website).

Programming

- Sebbene coding sia una fase molto importante del ciclo di sviluppo software, non è una componente essenziale!
- Per creare una applicazione, molteplici passi sono necessari: planning, design, testing, deployment, e maintenance.
- Programming è una attività complessa che comprende non solo coding, ma anche altre attività e.g. **progettare, analizzare e implementare** algoritmi, **capire** le strutture dati, e **risolvere problemi**.
- Programming è essenziale. Per scrivere codice è necessario essere trasparenti con lo schema o la struttura del programma: e.g. scrivere pseudocodice per descrivere la logica di un algoritmo, ed usarlo come strumento per spiegare l'algoritmo ad un programmatore (coder).

<https://hackr.io/blog/coding-vs-programming-difference-you-should-know>

Coding is not Programming

Coding

- I computer “capiscono” Codice Macchina (si veda il corso di Calcolatori), di difficile comprensione dagli esseri umani:
 - Linguaggi di Programmazione.
- Coding è il processo di usare un linguaggio di programmazione per fare sì che un computer faccia quello che lo sviluppatore vuole.
- In parole semplici, coding consiste in scrivere molteplici linee di codice per creare un programma software (application, game, or website).

Programming

- Sebbene coding sia una fase molto importante del ciclo di sviluppo software, non è una componente essenziale!
- Per creare una applicazione, molteplici passi sono necessari: planning, design, testing, deployment, e maintenance.
- Programming è una attività complessa che comprende non solo coding, ma anche altre attività e.g. **progettare, analizzare e implementare** algoritmi, **capire** le strutture dati, e **risolvere problemi**.
- Programming è essenziale. Per scrivere codice è necessario essere trasparenti con lo schema o la struttura del programma: e.g. scrivere pseudocodice per descrivere la logica di un algoritmo, ed usarlo come strumento per spiegare l'algoritmo ad un programmatore (coder).

<https://hackr.io/blog/coding-vs-programming-difference-you-should-know>

Coding is not Programming

Coding

Scopo È un processo di conversione di un insieme di istruzioni in un linguaggio comprensibile ad un computer

Skill Come coder, è richiesta buona conoscenza della sintassi e semantica del linguaggio di programmazione scelto

Programming

Oltre al coding, comprende la definizione dei requisiti, problem solving, scrivere pseudocodice, pensare algoritmi, ottimizzare, testing, creare eseguibili, manutenzione

Come programmatore, sono richieste oltre alle coding skill, capacità di pensiero (high-level thinking) e di analisi

<https://hackr.io/blog/coding-vs-programming-difference-you-should-know>

If Programming is the process of writing a whole book.

Then Coding is just about writing a single chapter of the book.

Coding is not Programming

Coding

Scopo È un processo di conversione di un insieme di istruzioni in un linguaggio comprensibile ad un computer

Skill Come coder, è richiesta buona conoscenza della sintassi e semantica del linguaggio di programmazione scelto

Programming

Oltre al coding, comprende la definizione dei requisiti, problem solving, scrivere pseudocodice, pensare algoritmi, ottimizzare, testing, creare eseguibili, manutenzione

Come programmatore, sono richieste oltre alle coding skill, capacità di pensiero (high-level thinking) e di analisi

<https://hackr.io/blog/coding-vs-programming-difference-you-should-know>

If Programming is the process of writing a whole book.

Then Coding is just about writing a single chapter of the book.

Coding is not Programming

	<u>Coding</u>	<u>Programming</u>
Scopo	È un processo di conversione di un insieme di istruzioni in un linguaggio comprensibile ad un computer	Oltre al coding, comprende la definizione dei requisiti, problem solving, scrivere pseudocodice, pensare algoritmi, ottimizzare, testing, creare eseguibili, manutenzione
Skill	Come coder, è richiesta buona conoscenza della sintassi e semantica del linguaggio di programmazione scelto	Come programmatore, sono richieste oltre alle coding skill, capacità di pensiero (high-level thinking) e di analisi

<https://hackr.io/blog/coding-vs-programming-difference-you-should-know>

If Programming is the process of writing a whole book.

Then Coding is just about writing a single chapter of the book.

Pre-requisiti

- **Nessuna conoscenza informatica pregressa**
- Nozioni di base di matematica
- Padronanza della lingua italiana, scritta e orale
- Discrete capacità logiche/analitiche

Pre-requisiti

- **Nessuna conoscenza informatica pregressa**
- **Nozioni di base di matematica**
- Padronanza della lingua italiana, scritta e orale
- Discrete capacità logiche/analitiche

Pre-requisiti

- Nessuna conoscenza informatica pregressa
- Nozioni di base di matematica
- Padronanza della lingua italiana, scritta e orale
- Discrete capacità logiche/analitiche

Pre-requisiti

- Nessuna conoscenza informatica pregressa
- Nozioni di base di matematica
- Padronanza della lingua italiana, scritta e orale
- Discrete capacità logiche/analitiche

Programma di massima (non necessariamente in ordine)

- Concetti generali
- Sviluppo di un programma
- I/O standard, I/O su files (argc & argv)
- Variabili e costanti
- Tipi (booleani, interi, reali, caratteri)
- Sintassi del C++ (cenni)
- Istruzioni elementari
- Istruzioni strutturate (condizioni, cicli)
- Funzioni e passaggi di parametri
- Array, Stringhe
- Organizzazione di un programma su più file: scope, visibilità e durata

Programma di massima (2)

- Puntatori e Riferimenti, Algebra dei puntatori
- Array e puntatori, Passaggio di parametri per puntatore
- Struct
- Allocazione dinamica di memoria
- Allocazione dinamica di array, struct
- [Cenni di programmazione Object-Oriented]
- Strutture dati fondamentali (liste, stack, code) e loro realizzazione tramite array e struct
- Strutture dati dinamiche e loro implementazione con liste concatenate
- Alberi binari

Riferimenti

- Appunti delle lezioni
- Handouts/slides “Corso Programmazione 1” (disponibile sul sito)
- **ESEMPI / Materiale didattico aggiuntivo** (disponibile sul sito)
- Libro di testo suggerito (alternativi):
 - John R. Hubbard. “Programming with C++” (2ed) McGraw Hill (ENGL): ISBN: 0-07-135346-1, <http://www.mathcs.richmond.edu/~hubbard/Books.html> (ITA): ISBN: 88-386-5045-4 “Programmare in C++” (non più disponibile nuovo)
 - Luis Joyanes Aguilar “Fondamenti di programmazione in C++” McGraw Hill, ISBN: 9788838664779 http://www.catalogo.mcgraw-hill.it/catLibro.asp?item_id=2299
- Riferimento per il linguaggio C++:
 - Bjarne Stroustrup “Il Linguaggio C++” Pearson, ISBN: ISBN 9788871920788 (4^a ed.) http://www.pearson.it/opera/pearson/0-2613-c_%2B%2B_linguaggio_libreria_standard_principi_di_programmazione

Modalità d'esame

- Prova pratica di programmazione al calcolatore
 - 5 appelli: **2 gennaio/febbraio, 2 giugno/luglio, 1 settembre**
 - Durata: **2 ore**
 - **3 esercizi + uno facoltativo molto difficile (per la lode)**
 - La modalità di svolgimento dipenderà da come evolve la situazione.
 - o svolti in laboratorio (aule A201/202 o B106), sotto linux Ubuntu
 - o svolti remotamente con meccanismo da definirsi
 - ammesso l'uso di editor/compilatore/debugger, nessun altro strumento, nessun testo
 - ...

Copiare all'esame pericolosissimo è.



© & TM Lucasfilm Ltd

Suggerimenti

- **FREQUENTARE/SEGUIRE LE LEZIONI E LE ESERCITAZIONI!!!!**
 - N.B.: ogni cosa detta a lezione è potenziale argomento di esame, non solo ciò che è esplicitamente contenuto nelle slide/materiale
- Chiedere ciò che non si capisce
- Non posticipare lo studio:
 - Studiare/provare a implementare dopo ogni lezione
 - ⇒ molto più efficiente per massimizzare il risultato a parità di sforzo
- Svolgere **sempre** gli “esercizi proposti”
- **Implementare, implementare, implementare!**
 - N.B.: la capacità di capire i messaggi del compilatore e/o del debugger è parte del bagaglio di conoscenze implicitamente richiesto per passare l'esame.
- ...

Suggerimenti

- **FREQUENTARE/SEGUIRE LE LEZIONI E LE ESERCITAZIONI!!!!**
 - N.B.: ogni cosa detta a lezione è potenziale argomento di esame, non solo ciò che è esplicitamente contenuto nelle slide/materiale
- Chiedere ciò che non si capisce
- Non posticipare lo studio:
 - Studiare/provare a implementare dopo ogni lezione
 ⇒ molto più efficiente per massimizzare il risultato a parità di sforzo
- Svolgere **sempre** gli “esercizi proposti”
- Implementare, implementare, implementare!
 - N.B.: la capacità di capire i messaggi del compilatore e/o del debugger è parte del bagaglio di conoscenze implicitamente richiesto per passare l'esame.
- ...

Suggerimenti

- **FREQUENTARE/SEGUIRE LE LEZIONI E LE ESERCITAZIONI!!!!**
 - N.B.: ogni cosa detta a lezione è potenziale argomento di esame, non solo ciò che è esplicitamente contenuto nelle slide/materiale
- Chiedere ciò che non si capisce
- Non posticipare lo studio:
 - Studiare/provare a implementare dopo ogni lezione
 - ⇒ molto più efficiente per massimizzare il risultato a parità di sforzo
- Svolgere **sempre** gli “esercizi proposti”
- Implementare, implementare, implementare!
 - N.B.: la capacità di capire i messaggi del compilatore e/o del debugger è parte del bagaglio di conoscenze implicitamente richiesto per passare l'esame.
- ...

Suggerimenti

- **FREQUENTARE/SEGUIRE LE LEZIONI E LE ESERCITAZIONI!!!!**
 - N.B.: ogni cosa detta a lezione è potenziale argomento di esame, non solo ciò che è esplicitamente contenuto nelle slide/materiale
- Chiedere ciò che non si capisce
- Non posticipare lo studio:
 - Studiare/provare a implementare dopo ogni lezione
 - ⇒ molto più efficiente per massimizzare il risultato a parità di sforzo
- Svolgere **sempre** gli “esercizi proposti”
- Implementare, implementare, implementare!
 - N.B.: la capacità di capire i messaggi del compilatore e/o del debugger è parte del bagaglio di conoscenze implicitamente richiesto per passare l'esame.
- ...

Suggerimenti

- **FREQUENTARE/SEGUIRE LE LEZIONI E LE ESERCITAZIONI!!!!**
 - N.B.: ogni cosa detta a lezione è potenziale argomento di esame, non solo ciò che è esplicitamente contenuto nelle slide/materiale
- Chiedere ciò che non si capisce
- Non posticipare lo studio:
 - Studiare/provare a implementare dopo ogni lezione
 - ⇒ molto più efficiente per massimizzare il risultato a parità di sforzo
- Svolgere **sempre** gli “esercizi proposti”
- **Implementare, implementare, implementare!**
 - N.B.: la capacità di capire i messaggi del compilatore e/o del debugger è parte del bagaglio di conoscenze implicitamente richiesto per passare l'esame.
- ...

Suggerimenti (2)

- ...

- Quando si implementa:

- Pensare e farsi uno schema prima si cominciare a scrivere codice
- **Adottare sempre le soluzioni più semplici possibile**
- **Scrivere il codice in modo il più chiaro e leggibile possibile**
 - Indentare il codice!!!
 - Lasciare spazi (ma non troppi spazi)
 - Usare nomi significativi e/o convenzionali
 - ...

- Seguire “coding standards”:

- Un “coding standard” da un aspetto “uniforme” a codice scritto da diversi programmati, migliora la leggibilità, la manutenibilità del codice, contribuisce a ridurne la complessità di comprensione, favorisce il riuso del codice, e contribuisce positivamente all’identificazione di errori.

Suggerimenti (2)

- ...

- Quando si implementa:

- Pensare e farsi uno schema prima si cominciare a scrivere codice
- *Adottare sempre le soluzioni più semplici possibile*
- *Scrivere il codice in modo il più chiaro e leggibile possibile*
 - Indentare il codice!!!
 - Lasciare spazi (ma non troppi spazi)
 - Usare nomi significativi e/o convenzionali
 - ...

- Seguire “coding standards”:

- Un “coding standard” da un aspetto “uniforme” a codice scritto da diversi programmati, migliora la leggibilità, la manutenibilità del codice, contribuisce a ridurne la complessità di comprensione, favorisce il riuso del codice, e contribuisce positivamente all’identificazione di errori.

Suggerimenti (2)

- ...

- Quando si implementa:

- Pensare e farsi uno schema prima si cominciare a scrivere codice
- **Adottare sempre le soluzioni più semplici possibile**
- **Scrivere il codice in modo il più chiaro e leggibile possibile**
 - Indentare il codice!!!
 - Lasciare spazi (ma non troppi spazi)
 - Usare nomi significativi e/o convenzionali
 - ...

- Seguire “coding standards”:

- Un “coding standard” da un aspetto “uniforme” a codice scritto da diversi programmati, migliora la leggibilità, la manutenibilità del codice, contribuisce a ridurne la complessità di comprensione, favorisce il riuso del codice, e contribuisce positivamente all’identificazione di errori.

Suggerimenti (2)

- ...
- Quando si implementa:
 - Pensare e farsi uno schema prima si cominciare a scrivere codice
 - **Adottare sempre le soluzioni più semplici possibile**
 - **Scrivere il codice in modo il più chiaro e leggibile possibile**
 - Indentare il codice!!!
 - Lasciare spazi (ma non troppi spazi)
 - Usare nomi significativi e/o convenzionali
 - ...
- Seguire “coding standards”:
 - Un “coding standard” da un aspetto “uniforme” a codice scritto da diversi programmati, migliora la leggibilità, la manutenibilità del codice, contribuisce a ridurne la complessità di comprensione, favorisce il riuso del codice, e contribuisce positivamente all’identificazione di errori.

Suggerimenti (2)

- ...
- Quando si implementa:
 - Pensare e farsi uno schema prima si cominciare a scrivere codice
 - **Adottare sempre le soluzioni più semplici possibile**
 - **Scrivere il codice in modo il più chiaro e leggibile possibile**
 - Indentare il codice!!!
 - Lasciare spazi (ma non troppi spazi)
 - Usare nomi significativi e/o convenzionali
 - ...
- Seguire “coding standards”:
 - Un “coding standard” da un aspetto “uniforme” a codice scritto da diversi programmati, migliora la leggibilità, la manutenibilità del codice, contribuisce a ridurne la complessità di comprensione, favorisce il riuso del codice, e contribuisce positivamente all’identificazione di errori.

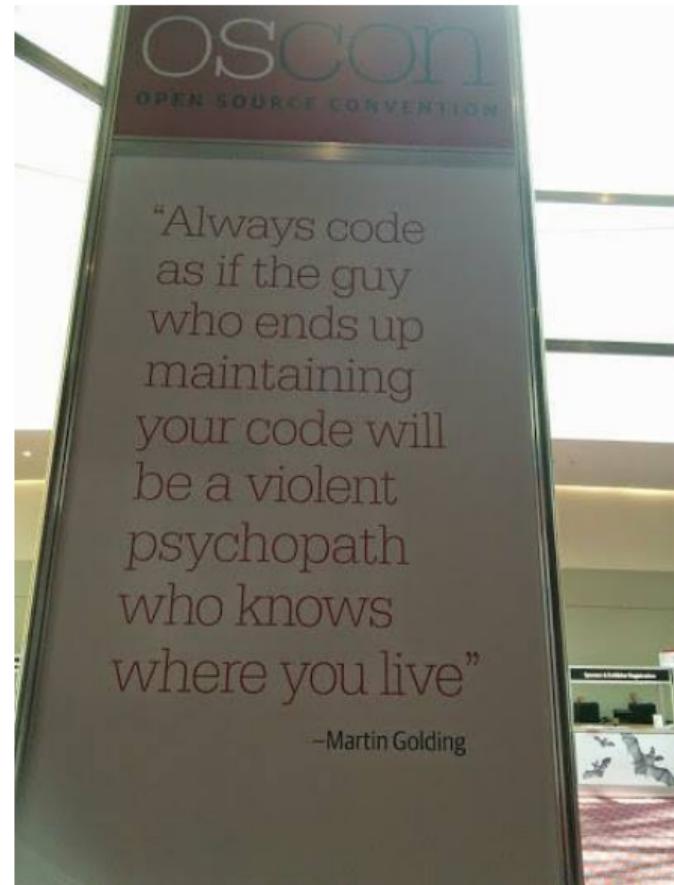
“Dovete programmare sempre come se il programmatore che deve mantenere il vostro codice fosse uno psicopatico violento che sa dove abitate.”

[Martin Golding]

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live”

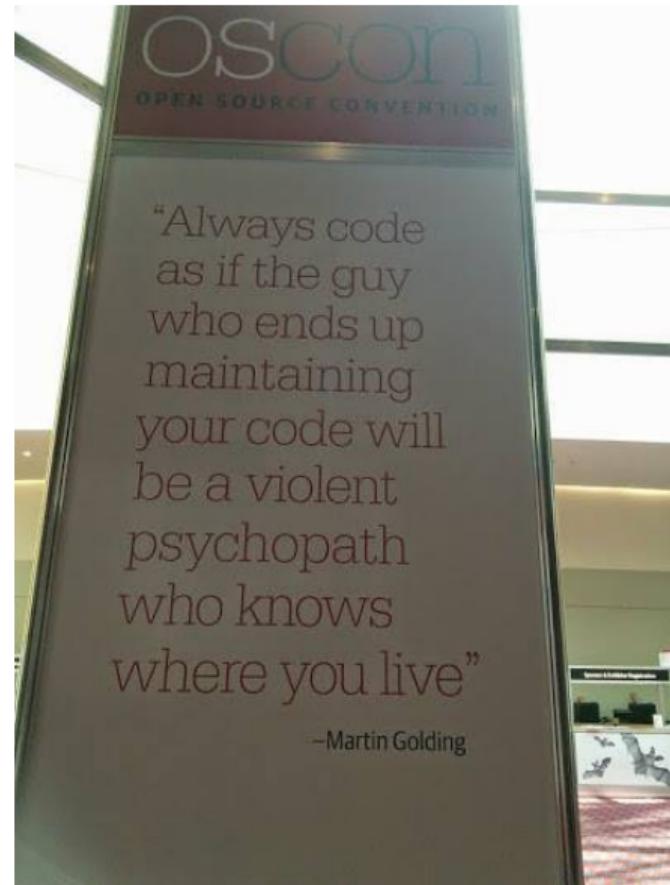
—Martin Golding

Vedere anche: <http://www.journaldev.com/240/my-25-favorite-programming-quotes-that-are-funny-too>



“Dovete programmare sempre come se il professore che deve **correggere** il vostro codice fosse uno psicopatico violento che sa dove abitate.”

[Marco Roveri and Roberto Sebastiani]



Vedere anche: <http://www.journaldev.com/240/my-25-favorite-programming-quotes-that-are-funny-too>



Suggerimenti (3)

- Suggerimento generale: **affrontare la programmazione con umiltà.**

- Ultime parole famose:

“Tanto io sono già capace di programmare”

- Se necessario, essere pronti a **cambiare il modo di lavorare**

- Ultime parole famose #2:

“Ho sempre fatto in questo modo”



Si vedano i risultati degli esami anni precedenti:

http://disi.unitn.it/rseba/DIDATTICA/prog1_2020/RISULTATI_ANNO_PRECEDENTE/

Suggerimenti (3)

- Suggerimento generale: **affrontare la programmazione con umiltà.**
 - Ultime parole famose:
“Tanto io sono già capace di programmare”
- Se necessario, essere pronti a **cambiare il modo di lavorare**
 - Ultime parole famose #2:
“Ho sempre fatto in questo modo”



The most dangerous phrase in the language is, 'We've always done it this way.'"

Si vedano i risultati degli esami anni precedenti:

http://disi.unitn.it/rseba/DIDATTICA/prog1_2020/RISULTATI_ANNO_PRECEDENTE/

Suggerimenti (3)

- Suggerimento generale: **affrontare la programmazione con umiltà.**
 - Ultime parole famose:
“Tanto io sono già capace di programmare”
- Se necessario, essere pronti a **cambiare il modo di lavorare**
 - Ultime parole famose #2:
“Ho sempre fatto in questo modo”

Si vedano i risultati degli esami anni precedenti:

http://disi.unitn.it/rseba/DIDATTICA/prog1_2020/RISULTATI_ANNO_PRECEDENTE/



Suggerimenti (3)

- Suggerimento generale: **affrontare la programmazione con umiltà.**
 - Ultime parole famose:
“Tanto io sono già capace di programmare”
- Se necessario, essere pronti a **cambiare il modo di lavorare**
 - Ultime parole famose #2:
“Ho sempre fatto in questo modo”



The most dangerous phrase in the language is, 'We've always done it this way.'"

Si vedano i risultati degli esami anni precedenti:

http://disi.unitn.it/rseba/DIDATTICA/prog1_2020/RISULTATI_ANNO_PRECEDENTE/

Suggerimenti (3)

- Suggerimento generale: **affrontare la programmazione con umiltà.**
 - Ultime parole famose:
“Tanto io sono già capace di programmare”
- Se necessario, essere pronti a **cambiare il modo di lavorare**
 - Ultime parole famose #2:
“Ho sempre fatto in questo modo”



The most dangerous phrase in the language is, 'We've always done it this way.'"

Si vedano i risultati degli esami anni precedenti:

http://disi.unitn.it/rseba/DIDATTICA/prog1_2020/RISULTATI_ANNO_PRECEDENTE/

Interazione con docente ed esercitatori

- Domande durante la lezione
 - fortemente incoraggiate
- Domande durante l'intervallo
 - piena disponibilità
- Ricevimento: su appuntamento, con orario da concordare di volta in volta (per email o a lezione),
 - Solo durante il periodo delle lezioni
- Invio di email
 - con moderazione e entro certi limiti (vedi slide successive)
- MAI PER TELEFONO!

Nota importante

È attivo un servizio di **tutorato** per studenti con difficoltà o lacune pregresse, gestito da studenti degli ultimi anni o dottorandi.

Interazione con docente ed esercitatori

- Domande durante la lezione
 - fortemente incoraggiate
- Domande durante l'intervallo
 - piena disponibilità
- Ricevimento: su appuntamento, con orario da concordare di volta in volta (per email o a lezione),
 - Solo durante il periodo delle lezioni
- Invio di email
 - con moderazione e entro certi limiti (vedi slide successive)
- MAI PER TELEFONO!

Nota importante

È attivo un servizio di tutorato per studenti con difficoltà o lacune pregresse, gestito da studenti degli ultimi anni o dottorandi.

Interazione con docente ed esercitatori

- Domande durante la lezione
 - fortemente incoraggiate
- Domande durante l'intervallo
 - piena disponibilità
- Ricevimento: su appuntamento, con orario da concordare di volta in volta (per email o a lezione),
 - Solo durante il periodo delle lezioni
- Invio di email
 - con moderazione e entro certi limiti (vedi slide successive)
- MAI PER TELEFONO!

Nota importante

È attivo un servizio di **tutorato** per studenti con difficoltà o lacune pregresse, gestito da studenti degli ultimi anni o dottorandi.

Interazione con docente ed esercitatori

- Domande durante la lezione
 - fortemente incoraggiate
- Domande durante l'intervallo
 - piena disponibilità
- Ricevimento: su appuntamento, con orario da concordare di volta in volta (per email o a lezione),
 - Solo durante il periodo delle lezioni
- Invio di email
 - con moderazione e entro certi limiti (vedi slide successive)
- MAI PER TELEFONO!

Nota importante

È attivo un servizio di tutorato per studenti con difficoltà o lacune pregresse, gestito da studenti degli ultimi anni o dottorandi.

Interazione con docente ed esercitatori

- Domande durante la lezione
 - fortemente incoraggiate
- Domande durante l'intervallo
 - piena disponibilità
- Ricevimento: su appuntamento, con orario da concordare di volta in volta (per email o a lezione),
 - Solo durante il periodo delle lezioni
- Invio di email
 - con moderazione e entro certi limiti (vedi slide successive)
- MAI PER TELEFONO!

Nota importante

È attivo un servizio di tutorato per studenti con difficoltà o lacune pregresse, gestito da studenti degli ultimi anni o dottorandi.

Interazione con docente ed esercitatori

- Domande durante la lezione
 - fortemente incoraggiate
- Domande durante l'intervallo
 - piena disponibilità
- Ricevimento: su appuntamento, con orario da concordare di volta in volta (per email o a lezione),
 - Solo durante il periodo delle lezioni
- Invio di email
 - con moderazione e entro certi limiti (vedi slide successive)
- MAI PER TELEFONO!

Nota importante

È attivo un servizio di **tutorato** per studenti con difficoltà o lacune pregresse, gestito da studenti degli ultimi anni o dottorandi.

Usi ed abusi dell'email (1)

Ogni email va inviata da `nome.cognome@studenti.unitn.it`, e deve sempre avere in copia tutti gli esercitatori!

Buoni motivi per inviarci una email

- Richieste di ricevimento
- Segnalazioni di eventuali refusi o errori nelle slides/programmi, problemi di accesso al sito web, ecc.
- Segnalazioni di problemi oggettivi (sovraposizioni di orario,...)
- Segnalazione di problemi individuali particolari, ad es:
 - studenti diversamente abili o con disturbi specifici dell'apprendimento (DSA) certificati
 - studenti non ancora iscritti/in corso di trasferimento
 - studenti lavoratori
 - studenti con problemi o situazioni particolari

(In questi casi il colloquio diretto è comunque preferibile.)



Usi ed abusi dell'email (1)

Ogni email va inviata da `nome.cognome@studenti.unitn.it`, e deve sempre avere in copia tutti gli esercitatori!

Buoni motivi per inviarci una email

- Richieste di ricevimento
- Segnalazioni di eventuali refusi o errori nelle slides/programmi, problemi di accesso al sito web, ecc.
- Segnalazioni di problemi oggettivi (sovraposizioni di orario,...)
- Segnalazione di problemi individuali particolari, ad es:
 - studenti diversamente abili o con disturbi specifici dell'apprendimento (DSA) certificati
 - studenti non ancora iscritti/in corso di trasferimento
 - studenti lavoratori
 - studenti con problemi o situazioni particolari

(In questi casi il colloquio diretto è comunque preferibile.)



...

Usi ed abusi dell'email (1)

Ogni email va inviata da `nome.cognome@studenti.unitn.it`, e deve sempre avere in copia tutti gli esercitatori!

Buoni motivi per inviarci una email

- Richieste di ricevimento
- Segnalazioni di eventuali refusi o errori nelle slides/programmi, problemi di accesso al sito web, ecc.
- Segnalazioni di problemi oggettivi (sovraposizioni di orario,...)
- Segnalazione di problemi individuali particolari, ad es:
 - studenti diversamente abili o con disturbi specifici dell'apprendimento (DSA) certificati
 - studenti non ancora iscritti/in corso di trasferimento
 - studenti lavoratori
 - studenti con problemi o situazioni particolari

(In questi casi il colloquio diretto è comunque preferibile.)

- ...

Usi ed abusi dell'email (1)

Ogni email va inviata da `nome.cognome@studenti.unitn.it`, e deve sempre avere in copia tutti gli esercitatori!

Buoni motivi per inviarci una email

- Richieste di ricevimento
- Segnalazioni di eventuali refusi o errori nelle slides/programmi, problemi di accesso al sito web, ecc.
- Segnalazioni di problemi oggettivi (sovraposizioni di orario,...)
- Segnalazione di problemi individuali particolari, ad es:
 - studenti diversamente abili o con disturbi specifici dell'apprendimento (DSA) certificati
 - studenti non ancora iscritti/in corso di trasferimento
 - studenti lavoratori
 - studenti con problemi o situazioni particolari

(In questi casi il colloquio diretto è comunque preferibile.)

- ...

Usi ed abusi dell'email (1)

Ogni email va inviata da `nome.cognome@studenti.unitn.it`, e deve sempre avere in copia tutti gli esercitatori!

Buoni motivi per inviarci una email

- Richieste di ricevimento
- Segnalazioni di eventuali refusi o errori nelle slides/programmi, problemi di accesso al sito web, ecc.
- Segnalazioni di problemi oggettivi (sovraposizioni di orario,...)
- Segnalazione di problemi individuali particolari, ad es:
 - studenti diversamente abili o con disturbi specifici dell'apprendimento (DSA) certificati
 - studenti non ancora iscritti/in corso di trasferimento
 - studenti lavoratori
 - studenti con problemi o situazioni particolari

(In questi casi il colloquio diretto è comunque preferibile.)

- ...

Usi ed abusi dell'email (1)

Ogni email va inviata da `nome.cognome@studenti.unitn.it`, e deve sempre avere in copia tutti gli esercitatori!

Buoni motivi per inviarci una email

- Richieste di ricevimento
- Segnalazioni di eventuali refusi o errori nelle slides/programmi, problemi di accesso al sito web, ecc.
- Segnalazioni di problemi oggettivi (sovraposizioni di orario,...)
- Segnalazione di problemi individuali particolari, ad es:
 - studenti diversamente abili o con disturbi specifici dell'apprendimento (DSA) certificati
 - studenti non ancora iscritti/in corso di trasferimento
 - studenti lavoratori
 - studenti con problemi o situazioni particolari
- ...

(In questi casi il colloquio diretto è comunque preferibile.)

Usi ed abusi dell'email (2)

Motivi inutili per inviarci una email

- Giustificazioni di assenze/risultati ecc.
 - “Non sono potuto venire a lezione perche'...”
 - “Le prossime due settimane sarò assente...”
 - “Ho sbagliato il terzo esercizio dell'esame, ...”
 - “Mi sono iscritto all'appello ma non sono potuto venire ...”
 - ...

Usi ed abusi dell'email (2)

Motivi inutili per inviarci una email

- Giustificazioni di assenze/risultati ecc.
 - “Non sono potuto venire a lezione perche’...”
 - “Le prossime due settimane sarò assente...”
 - “Ho sbagliato il terzo esercizio dell’esame, ...”
 - “Mi sono iscritto all’appello ma non sono potuto venire ...”
 - ...

Usi ed abusi dell'email (3)

Cattivi motivi per inviarci una email

- Richieste di informazioni già date a lezione o presenti sul sito
- Richieste di risoluzione di problemi individuali
 - *“Perche’ il programma in allegato non funziona?”*
 - *“Ho istallato la versione ZBX1002.34.56 del programma/sistema operativo XYZ, ma mi riporta i seguenti errori:...”*
 - *“Vorrei comprare un pc/laptop/tablet/etc., va meglio il modello ZK45.5 della XYZ o il KW32.7 della ZYX?”*
- Domande/richieste/affermazioni sull’esame
 - *“Sono usciti i risultati?”*
 - *“Perche’ non sono ancora usciti i risultati?”*
 - *“io credo di aver fatto questo errore, ma è stata una distrazione, ...”*
 - *“io avevo fatto tutto giusto, poi il computer mi ha cancellato il file...”*
 - *“Io giuro, in aula compilava!”*
 - ...

Usi ed abusi dell'email (3)

Cattivi motivi per inviarci una email

- Richieste di informazioni già date a lezione o presenti sul sito
- Richieste di risoluzione di problemi individuali
 - *“Perche' il programma in allegato non funziona?”*
 - *“Ho istallato la versione ZBX1002.34.56 del programma/sistema operativo XYZ, ma mi riporta i seguenti errori:...”*
 - *“Vorrei comprare un pc/laptop/tablet/etc., va meglio il modello ZK45.5 della XYZ o il KW32.7 della ZYX?”*
- Domande/richieste/affermazioni sull'esame
 - *“Sono usciti i risultati?”*
 - *“Perche' non sono ancora usciti i risultati?”*
 - *“io credo di aver fatto questo errore, ma è stata una distrazione, ...”*
 - *“io avevo fatto tutto giusto, poi il computer mi ha cancellato il file...”*
 - *“Io giuro, in aula compilava!”*
 - ...

Usi ed abusi dell'email (3)

Cattivi motivi per inviarci una email

- Richieste di informazioni già date a lezione o presenti sul sito
- Richieste di risoluzione di problemi individuali
 - *“Perche’ il programma in allegato non funziona?”*
 - *“Ho istallato la versione ZBX1002.34.56 del programma/sistema operativo XYZ, ma mi riporta i seguenti errori:...”*
 - *“Vorrei comprare un pc/laptop/tablet/etc., va meglio il modello ZK45.5 della XYZ o il KW32.7 della ZYX?”*
- Domande/richieste/affermazioni sull’esame
 - *“Sono usciti i risultati?”*
 - *“Perche’ non sono ancora usciti i risultati?”*
 - *“io credo di aver fatto questo errore, ma è stata una distrazione, ...”*
 - *“io avevo fatto tutto giusto, poi il computer mi ha cancellato il file...”*
 - *“Io giuro, in aula compilava!”*
 - ...

Usi ed abusi dell'email (3)

Cattivi motivi per inviarci una email

- Richieste di informazioni già date a lezione o presenti sul sito
- Richieste di risoluzione di problemi individuali
 - *“Perche’ il programma in allegato non funziona?”*
 - *“Ho istallato la versione ZBX1002.34.56 del programma/sistema operativo XYZ, ma mi riporta i seguenti errori:...”*
 - *“Vorrei comprare un pc/laptop/tablet/etc., va meglio il modello ZK45.5 della XYZ o il KW32.7 della ZYX?”*
- Domande/richieste/affermazioni sull’esame
 - *“Sono usciti i risultati?”*
 - *“Perche’ non sono ancora usciti i risultati?”*
 - *“io credo di aver fatto questo errore, ma è stata una distrazione, ...”*
 - *“io avevo fatto tutto giusto, poi il computer mi ha cancellato il file...”*
 - *“Io giuro, in aula compilava!”*
 - ...

Usi ed abusi dell'email (4)

Infine, la buona educazione e il rispetto nei confronti del docente **e degli esercitatori** via email (come nell'interazione diretta) è d'obbligo.

(Vedi anche: *“Bad Email Reply - What not to say to your professor...”*
[https://www.youtube.com/watch?v=zSNc8F9tqzY. \)](https://www.youtube.com/watch?v=zSNc8F9tqzY.)

PS:

Per evitare gaffes o situazioni sgradevoli, ricordatevi che anche i vostri docenti/esercitatori usano internet, social media, ecc:

<https://www.facebook.com/spottedunitn/posts/410484665765366?fbref=nf&pnref=story>

[cfr Sebastiani]

Usi ed abusi dell'email (4)

Infine, la buona educazione e il rispetto nei confronti del docente **e degli esercitatori** via email (come nell'interazione diretta) è d'obbligo.

(Vedi anche: *“Bad Email Reply - What not to say to your professor...”*
[https://www.youtube.com/watch?v=zSNc8F9tqzY. \)](https://www.youtube.com/watch?v=zSNc8F9tqzY.)

PS:

Per evitare gaffes o situazioni sgradevoli, ricordatevi che anche i vostri docenti/esercitatori usano internet, social media, ecc:

<https://www.facebook.com/spottedunitn/posts/410484665765366?fbref=nf&pnref=story>

[cfr Sebastiani]

Corso “Programmazione 1”

Capitolo 00: Regole di comportamento COVID-19

Docente: **Marco Roveri** - marco.roveri@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

Ultimo aggiornamento: 14 settembre 2020

MODALITÀ DI ACCESSO STUDENTI ALLE SEDI

- Dovete essere autorizzati (nominativamente)
- Dovete accedere alle sedi conoscendo già l'aula di destinazione e seguendo le indicazioni (check-in/out con app) - uscire seguendo le indicazioni
- Indossare **SEMPRE** la mascherina
- Prima di prendere posto nell'aula dovete igienizzare le mani
- Dovete accedere all'aula uno per volta rispettando le distanze interpersonali
- Portare con sé solo gli oggetti strettamente indispensabili, evitando poi ogni scambio di oggetti personali

MODALITÀ DI ACCESSO STUDENTI ALLE SEDI

- Dovete portare con voi solo gli oggetti strettamente indispensabili, evitando poi ogni scambio di oggetti personali
- Non appoggiare nemmeno temporaneamente borse o zaini sui banchi o sulle sedie non utilizzate tra le postazioni
- Dovete prendere posto nei posti disponibili
- Dovete minimizzare gli spostamenti all'interno dell'aula, mantenendo le distanze e evitando la contaminazione delle superfici
- Al termine delle lezioni, o per la pausa, dovete lasciare l'aula procedendo possibilmente una fila alla volta, al fine di non creare assembramenti
- Al rientro dalle pause, mantengono per quanto possibile il proprio posto iniziale
- **Turnazione in blocchi di lezione in aula in gruppi prestabili e non modificabili.**

MODALITÀ DI ACCESSO STUDENTI ALLE SEDI

Nel caso di uno studente che, durante le lezioni/esercitazioni, manifesti sintomi simil-influenzali, va sottoposto alla medesima procedura prevista dal protocollo operativo generale. In particolare, deve misurarsi la temperatura corporea (usando il termometro presente in portineria) e, in caso di valore superiore a 37,5°C, egli deve essere posto in un locale separato, preventivamente individuato (es. infermeria); deve contattare il proprio medico curante e dare comunicazione al docente responsabile dell'attività in aula o in laboratorio. Questi si adopera affinché venga attivata la procedura di comunicazione all'Azienda Sanitaria. Se lo studente non è in grado di rientrare al proprio domicilio con mezzi personali, viene richiesta indicazione al 112.

Lo studente deve attenersi a quanto viene indicato dal proprio medico curante e dall'Azienda Sanitaria; verranno poi definiti dall'Azienda Sanitaria i luoghi e le persone con le quali è entrato in contatto. Salvo diversa indicazione dell'Azienda Sanitaria stessa, i possibili contatti stretti vanno invitati a limitare i propri contatti sociali sino al termine dell'indagine.

Zoom link per lezioni teoriche e laboratorio

Topic: Programmazione 1

Join Zoom Meeting: <https://unitn.zoom.us/j/92225974809>

Meeting ID: 922 2597 4809

Passcode: 375001

Dial by your location

+39 020 066 7245 Italy +39 021 241 28 823 Italy +39 069 480 6488 Italy

Meeting ID: 922 2597 4809

Passcode: 375001

Find your local number: <https://unitn.zoom.us/u/ac06hTmmPG>

Join by SIP

92225974809@zoomcrc.com

Passcode: 375001

Aule ed orari

Lezioni teoriche:

Lunedí: 11:30-13:30 Mesiano Aula T2

Mercoledí: 15:30-18:30 Mesiano Aula T2

Laboratorio:

Martedí:

08:30-10:30 Povo Aula pc B106

11:30-13:30 Povo Aula pc B106

Giovedí:

08:30-10:30 Povo Aula pc B106

11:30-13:30 Povo Aula pc B106



Aule ed orari

Lezioni teoriche:

Lunedí: 11:30-13:30 Mesiano Aula T2

Mercoledí: 15:30-18:30 Mesiano Aula T2

Laboratorio:

Martedí:

08:30-10:30 Povo Aula pc B106

11:30-13:30 Povo Aula pc B106

Giovedí:

08:30-10:30 Povo Aula pc B106

11:30-13:30 Povo Aula pc B106



Corso “Programmazione 1”

Capitolo 01: Concetti Elementari

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato@unitn.it¹

¹ In via di definizione

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL:

<https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 13 settembre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Ambiente di sviluppo ed installazione

- Assumiamo un ambiente di sviluppo Linux
- Distribuzioni Linux
 - Linux Ubuntu and similar
 - sudo apt-get install build-essential
 - Linux CentOS, RHEL, Fedora and similar
 - yum install gcc gcc-c++ kernel-devel make
 - yum groupinstall "Development Tools"
 - Linux Mint and similar
 - dnf groupinstall "Development Tools"
- Per ambiente MS Windows (Windows 10)
 - Windows Linux Subsystem, e poi come per Unix like
 - <https://docs.microsoft.com/en-us/windows/wsl/install-win10>
 - MinGW - <http://www.mingw.org/>
 - Cygwin - <http://www.cygwin.com>
- Per ambiente MacOS
 - Installare “Xcode” attraverso l’App Store

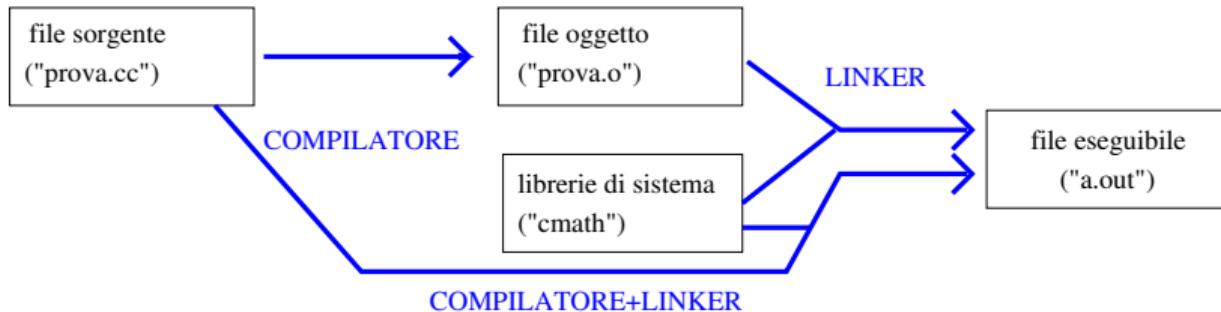
Scrittura di un programma

- Scrittura di uno o più file di testo, chiamato/i **sorgente**
 - Scritto in un linguaggio di programmazione (es C++)
 - Stampabile, comprensibile ad un essere umano
- Creazione e modifica tramite uno strumento chiamato **editor**
 - Es. **Emacs** o **vi** o **gedit** ... sotto linux

in C++ tipicamente l'estensione del nome è “.cc”, “.cpp” (o “.h” o “.hpp”)

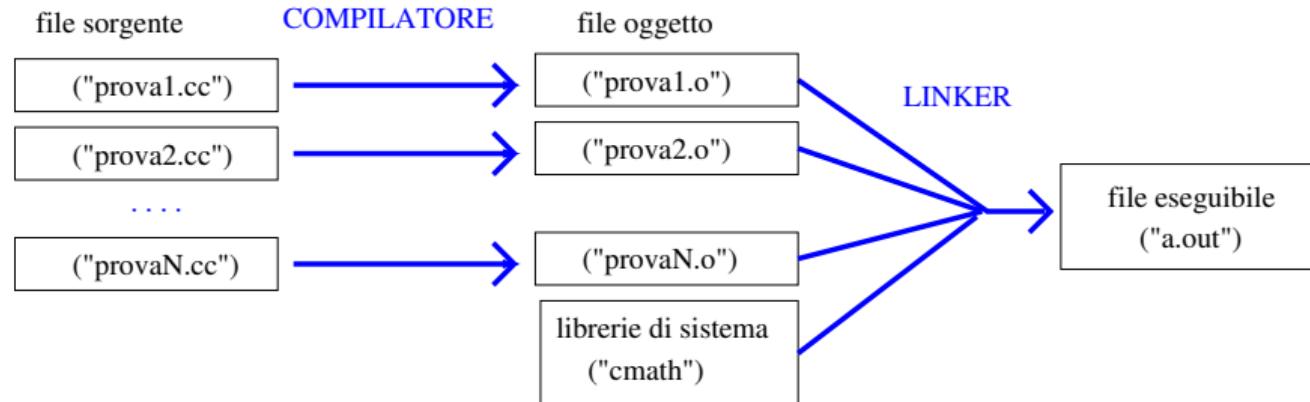
- Es. “prova.cc”, “prova.cpp”

Compilazione (un unico file)



- File sorgente tradotto in un file oggetto dal compilatore
 - Es: `g++ -c prova.cc`
 - Imp: file oggetto illeggibile ad un essere umano e non stampabile!
- File oggetto collegato (linked) a librerie di sistema dal linker, generano un file eseguibile (default `a.out`, opzione `-o <nome>`)
 - Es: `g++ prova.o`
 - Es: `g++ prova.o -o prova`
 - File incomprensibili agli umani, ma eseguibili da una macchina
- Compilazione e linking possibile in un'unica istruzione
 - Es: `g++ prova.cc`

Compilazione (su più files)



- File sorgente tradotti nei rispettivi **file oggetto** uno alla volta
- File oggetto collegato (linked) a librerie di sistema dal **linker**, generano un **file eseguibile (default a.out)**
 - **Es:** g++ prova1.o prova2.o ... provaN.o
- Compilazione e linking possibile in un'unica istruzione
 - **Es:** g++ prova1.cc prova2.cc ... provaN.cc

ESEMPI

- Esempio di com'è fatto un programma:
 { .../ESEMPI_BASE/esempio_fattoriale.cc }
- Con compilazione separata:
 { .../ESEMPI_BASE/fact.cc
 .../ESEMPI_BASE/esempio_fattoriale2.cc }

Un “template” di programma C++

Componente “comune” a tutti i programmi di questo corso::

```
{ ..../ESEMPI_BASE/template.cc }
```

```
using namespace std;

#include <iostream>

int main ()
{
    return 0;
}
```

Un programma C++ elementare

Output di una stringa predefinita:

```
{ ./ESEMPI_BASE/ciao.cc }
```

```
using namespace std;
#include <iostream> // chiamata a libreria

int main ()           // funzione principale
{
    cout << "Ciao_a_tutti\n"; // istruzione di output
    return 0;
}
```

Variante con “endl”:

```
{ ./ESEMPI_BASE/ciao2.cc }
```

Fine Lezione 01

Corso “Programmazione 1”

Capitolo 01: Concetti Elementari

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato@unitn.it¹

¹ In via di definizione

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL:

<https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 13 settembre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Identifieri (Identifiers)

- Le entità di un programma C++ devono avere dei nomi che ne consentano l'individuazione
- Nel caso più semplice i nomi sono degli **identifieri** liberamente scelti
- Un identificatore è una parola inizianta con una lettera

Nota

- Il carattere “_” è una lettera
- Il C++ distingue tra maiuscole e minuscole (si dice che è **case sensitive**).
 - Es: Fact è diverso da fact

Parole Chiave (keywords)

Le parole chiave del C++ sono un insieme di simboli il cui significato è stabilito dal linguaggio e non può essere ridefinito in quanto il loro uso è riservato.

Principali parole chiavi del C++

```
asm auto break case catch char class const continue
default delete do double else enum extern
float for friend goto if inline int long new
operator private protected public register return
short signed sizeof static struct switch
template this throw try typedef union virtual void
volatile while ...
```

Le Espressioni Letterali

- Denotano valori costanti, spesso sono chiamati solo “letterali” (o “costanti senza nome”, o “valori costanti”)
- Possono essere:
 - costanti **carattere** (denotano singoli caratteri)
 - costanti **stringa** (denotano sequenze di caratteri)
 - costanti **numeriche intere**
 - costanti **numeriche reali**

Rappresentazione costanti carattere

- Una costante carattere viene rappresentata racchiudendo il corrispondente carattere fra apici: per esempio la costante 'a' rappresenta il carattere 'a'
- I caratteri di controllo vengono rappresentati da combinazioni speciali **dette sequenze di escape** che iniziano con una barra invertita (backslash '\')

Sequenze di Escape

Nome	Abbreviazione	Sequenza di Escape
nuova riga	NL (LF)	\n
tabulazione orizzontale	HT	\t
tabulazione verticale	VT	\v
spazio indietro	BS	\b
ritorno carrello	CR	\r
avanzamento modulo	FF	\f
segnale acustico	BEL	\a
barra invertita	\	\\
apice	'	\'
virgolette	"	\"

Esempio:

```
{ ./ESEMPI_BASE/escape.cc }
```

Rappresentazione di Costanti Stringa

- Una costante stringa è una lista di caratteri compresa fra una coppia di virgolette
 - esempio: "Hello!"
 - possono includere anche spaziature, caratteri non alfanumerici e sequenze di escape: per esempio "Hello, world!\n"

Rappresentazione di Numeri

- Un **numero intero** viene rappresentato da una sequenza di cifre, che vengono interpretate
 - in **base decimale** (default)
 - in **base ottale** se inizia con uno zero
 - in **base esadecimale** se inizia con '0x' (o '0X')
- Un **numero reale** (in virgola mobile) viene rappresentato facendo uso del punto decimale e della lettera 'e' (o 'E') per separare la parte in virgola fissa dall'esponente;
 - ad esempio: -1235.6e-2 rappresenta -12,356

Nota:

In un numero reale, la virgola viene rappresentata secondo la notazione anglosassone con un punto “.”.

Esempio di rappresentazione di numeri in C++:

```
{ .. /ESEMPI_BASE /rappr_numeri.cc }
```

Operatori

Alcuni caratteri speciali e loro combinazioni sono usati come operatori, cioè servono a denotare certe operazioni nel calcolo delle espressioni.

- Esempio: $2 * 3 + 4$

Principali operatori del C++

+	-	*	/	%	^	&		~
!	=	<	>	+=	-=	*=	/=	%=
^=	&=	=	<<	>>	>>=	<<=	==	!=
<=	>=	&&		++	--	,	->*	->
.*	::	()	[]	?:				

Proprietà degli Operatori

- La **posizione** rispetto ai suoi operandi (o argomenti):
un operatore si dice
 - **prefisso** se precede gli argomenti
 - **postfisso** se segue gli argomenti
 - **infisso** se sta fra gli argomenti
- Il **numero di argomenti** (o **arità**)
- La **precedenza** (o **priorità**) nell'ordine di esecuzione;
 - ad esempio: l'espressione $1+2*3$ viene calcolata come $1+(2*3)$ e non come $(1+2)*3$
- L'**associatività**: l'ordine in cui vengono eseguiti operatori della stessa priorità
 - gli operatori possono essere associativi **a destra** o **a sinistra**

I Separatori

Principali Separatori del C++

() , ; : { }

- I separatori sono simboli di interruzione, che indicano il termine di una istruzione, separano elementi di liste, raggruppano istruzioni o espressioni, ecc.
- Alcuni caratteri, come la virgola, possono essere sia separatori che operatori, a seconda del contesto
- Le parentesi devono essere sempre usate in coppia, come nel normale uso matematico

Corso “Programmazione 1”

Capitolo 02: Variabili, Costanti, Tipi

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato@unitn.it¹

¹ In via di definizione

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL:

<https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 17 settembre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Le Variabili e le Costanti

- Per memorizzare un valore in un'area di memoria si utilizzano entità chiamate **variabili** o **costanti**.
- Le **variabili** permettono la modifica del loro valore durante l'esecuzione del programma.
- L' **area di memoria** corrispondente è identificata da un **nome**, che ne individua l'**indirizzo** in memoria.

Le Variabili

Le variabili sono caratterizzate da una quadrupla:

- il **nome** (identificatore)
- il **tipo**
- la **locazione** di memoria (l-value o indirizzo)
- il **valore** (r-value)

Definizione e Dichiarazione di Variabili I

- **Definizione:**

- Formato: tipo identificatore;
- Esempio: int x;
- Se (e solo se!) una variabile è definita esternamente ad ogni funzione, main() inclusa, (variabile **globale**) viene automaticamente inizializzata al valore 0

- **Definizione con inizializzazione:**

- Formato: tipo identificatore=exp;
- Esempio: int x=3*2;
- Esempio: int y=3*z; // z definita precedentemente

- **Dichiarazione:**

- Formato: extern tipo identificatore;
- Esempio: extern int x;

output di variabile

Per produrre in output il valore di una variabile x, si usa l'istruzione:

```
cout << x ...
```

Definizione e Dichiarazione di Variabili II

- **Definizione:** quando il compilatore incontra una definizione di una variabile, esso predisponde l'allocazione di un'area di memoria in grado di contenere la variabile del tipo scelto
- **Dichiarazione:** specifica solo il tipo della variabile e presuppone dunque che la variabile venga definita in un'altra parte del programma

Esempio di definizioni di variabili:

```
{ ..../ESEMPI_BASE/variabili.cc }
```

... con inizializzazione:

```
{ ..../ESEMPI_BASE/variabili2.cc }
```

Dichiarazione di Costanti

- Sintassi:

- Formato: `const tipo identificatore = exp;`
- `exp` deve essere una espressione il cui valore deve poter essere calcolato in fase di compilazione
(su alcuni compilatori è possibile inizializzare costanti a valore non costanti, ma il risultato è imprevedibile e varia a seconda dei casi \Rightarrow **da evitare assolutamente**)

- Esempi

```
const int kilo = 1024;  
const double pi = 3.14159;  
const int mille = kilo - 24;
```

Esempio di uso di costanti:

```
{ ./ESEMPI_BASE/costanti.cc }
```

Concetto di stream

- Un programma comunica con l'esterno tramite uno o più **flussi di caratteri (stream)**
- Uno stream è una struttura logica costituita da una sequenza di caratteri, in numero teoricamente infinito, terminante con un apposito carattere che ne identifica la fine.
- Gli stream vengono associati (con opportuni comandi) ai dispositivi fisici collegati al computer (tastiera, video) o a file residenti sulla memoria di massa

Stream predefiniti

- In C++ esistono i seguenti stream predefiniti
 - `cin` (stream standard di ingresso)
 - `cout` (stream standard di uscita)
 - `cerr` (stream standard di errore)
- Le funzioni che operano su questi stream sono in una libreria di ingresso/uscita e per usarle occorre la direttiva:

```
#include <iostream>
```

Stream di uscita

- Lo stream di uscita standard (`cout`) è quello su cui il programma scrive i dati
 - è tipicamente associato **allo schermo**
- Per scrivere dati si usa l'istruzione di scrittura:
`stream << espressione ;`
- L'istruzione di scrittura comporta:
 - il calcolo del valore dell'espressione
 - la conversione in una sequenza di caratteri
 - il trasferimento della sequenza nello stream

Nota:

La costante predefinita `endl` corrisponde ad uno ' `\n`' , per cui viene iniziata una nuova linea con il cursore nella prima colonna

Scritture multiple

- L'istruzione di scrittura ha una forma più generale che consente **scritture multiple**, nel formato

```
cout << espressione1 << espressione2 << ...
```

- Ad esempio,

```
cout << x << y << endl;
```

corrisponde a:

```
cout << x;  
cout << y;  
cout << endl;
```

Esempio di uso di operazioni di output:

```
{ ..../ESEMPI_BASE/esempio_cout.cc }
```

... con output multiplo:

```
{ ..../ESEMPI_BASE/esempio_cout_multiplo.cc }
```

Stream di ingresso

- Lo stream di ingresso standard (`cin`) è quello da cui il programma preleva i dati
 - è tipicamente associato **alla tastiera**
- Per prelevare dati si usa l'istruzione di lettura:

```
stream >> espressione ;
```

dove `espressione` deve essere un'**espressione dotata di indirizzo** (per ora, una variabile)

- L'istruzione di lettura comporta in ordine:
 1. il prelievo dallo stream di una sequenza di caratteri
 2. la conversione di tale sequenza in un valore che viene assegnato alla variabile

Lettura di un carattere da cin

- Consideriamo l'istruzione `cin >> x`, dove `x` è di tipo `char`

□□□□a□□-14.53□□728□□ . . .
↑

(qui “□” rappresenta uno spazio e “↑” il cursore)

- se il carattere puntato dal cursore è una spaziatura:
 - il cursore si sposta in avanti per trovare un carattere che non sia una spaziatura
- se il carattere puntato dal cursore non è una spaziatura:
 - il carattere viene prelevato
 - il carattere viene assegnato alla variabile `x`
 - il puntatore si sposta alla casella successiva

Lettura di un numero da cinque cifre

- Consideriamo l'istruzione `cin >> x`, dove `x` è un numero (ad esempio di tipo `int`)
 - se il carattere puntato dal cursore è una spaziatura:
 - come nel caso precedente
 - se la sequenza di caratteri è interpretabile come un numero compatibile con il tipo di `x`:

□□□□*a*□□-14.53□□728□□ ...
↑

- 1. la sequenza di caratteri viene prelevata
 - 2. la sequenza viene convertita nel suo corrispondente valore (es, il valore intero 728) che viene assegnato alla variabile x
 - 3. il puntatore si sposta alla casella successiva
 - altrimenti, l'operazione di prelievo non avviene e lo stream si porta in uno stato di errore

Letture multiple

- L'istruzione di ingresso ha una forma più generale che consente **lettura multiple**, nel formato

```
cin >> var1 >> var2 >> ...
```

- Ad esempio, se `x`, `y` , `z` sono risp. `char`, `double` e `int`:

```
cin >> x >> y >> z;
```

corrisponde a:

```
cin >> x;  
cin >> y;  
cin >> z;
```

□□□□a□□-14.53□□728□□ ...



Esempi

Esempio di uso di operazioni di input:

```
{ ..../ESEMPI_BASE/esempio_cin.cc }
```

... con input multiplo:

```
{ ..../ESEMPI_BASE/esempio_cin2.cc }
```

... prima intero e poi reale:

```
{ ..../ESEMPI_BASE/esempio_cin3.cc }
```

Alcune funzioni utili della libreria <iostream>

- `cin.eof()`: ritorna un valore diverso da 0 se lo stream cin ha raggiunto la sua fine (End Of File)
 - va usato sempre dopo almeno un'operazione di lettura
 - richiede un separatore dopo l'ultimo elemento letto
- `cin.fail()`: ritorna un valore diverso da 0 se lo stream cin ha rilevato uno stato di errore (e.g. stringa per `int`) o un end-of-file
 - non necessariamente usato dopo almeno un'operazione di lettura
 - non richiede un separatore dopo l'ultimo elemento letto
- `cin.clear()`: ripristina lo stato normale dallo stato di errore

Fine Lezione 02

Corso “Programmazione 1”

Capitolo 02: Variabili, Costanti, Tipi

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato@unitn.it¹

¹ In via di definizione

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 24 settembre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Rappresentazione binaria dei numeri

- La rappresentazione binaria dei numeri avviene tramite sequenze di bit (uni e zeri).
- Distinguiamo la rappresentazione per:
 - numeri interi positivi
 - numeri interi con segno
 - numeri reali

Numeri interi positivi

- Una sequenza di bit $b_{n-1} \dots b_1 b_0$ rappresenta il numero:

$$\sum_{i=0}^{n-1} 2^i \cdot b_i = 2^{n-1} \cdot b_{n-1} + \dots + 4 \cdot b_2 + 2 \cdot b_1 + 1 \cdot b_0$$

- Dati n bit è possibile rappresentare numeri nell'intervallo $[0, 2^n - 1]$ (ad esempio, nell'intervallo $[0, 4294967295]$ con 32 bit)
- Esempio (con 8 bit):
00001001 rappresenta il numero 9 ($2^3 + 2^0 = 8 + 1 = 9$)

Conversione Decimale-Binario

- Serie di divisioni per 2 e analisi del resto
- Convertire 728_{10} in binario

728	0	
364	0	Risultato: 1011011000_2
182	0	
91	1	
45	1	
22	0	
11	1	
5	1	Prova: $2^9 + 2^7 + 2^6 + 2^4 + 2^3$
2	0	$= 512 + 128 + 64 + 16 + 8 = 728$
1	1	

Conversione Decimale-Binario

- Convertire 3249_{10} in binario

3249	1
1624	0
812	0
406	0
203	1
101	1
50	0
25	1
12	0
6	0
3	1
1	1

Risultato: 110010110001

Prova: $2^{11} + 2^{10} + 2^7 + 2^5 + 2^4 + 2^0$
 $= 2048 + 1024 + 128 + 32 + 16 + 1 = 3249$

Esempi: operazioni tra interi positivi

95	+	01011111	64+16+8+4+2+1	0	+	0	=	0
54		00110110	32+16+4+2	1	+	0	=	1
<hr/>								
149		10010101	128+16+4+1	1	+	1	=	0 (riporto 1)
				0	+	1	=	1
149	-	10010101	128+16+4+1	0	-	0	=	0
095		01011111	64+16+8+4+2+1	1	-	0	=	1
<hr/>								
54		00110110	32+16+4+2	1	-	1	=	0
				0	-	1	=	1 (prestato 1 col. sin.)

N.B.: Se eccede il range, il bit di riporto si perde (overflow) \Rightarrow aritmetica modulo 2^N

223	+	11011111	128+64+16+8+4+2+1
54		00110110	32+16+4+2
<hr/>			
21		00010101	16+4+1 (223+54-256=21)

Interi con segno

- I numeri interi con segno sono rappresentati tramite diverse codifiche
- Le codifiche più usate sono:
 - codifica **segno-valore**
 - codifica **complemento a 2**

Codifica Segno-valore

- Il primo bit rappresenta il segno, gli altri il valore
 - Esempio (8 bit): 10001001 rappresenta -9
- Comporta una doppia rappresentazione dello zero:
00000000 e 10000000
- I numeri rappresentati appartengono all'intervallo $[-2^{n-1} + 1, 2^{n-1} - 1]$
(ad esempio $[-2147483647, 2147483647]$ con 32 bit)
- poco usato in pratica

Nota:

Occorre usare due diversi algoritmi per la somma a seconda che il segno degli addendi sia concorde o discorde

Esempi: operazioni in segno-valore

Se il segno è diverso, si stabilisce il maggiore dei due in valore assoluto, e si calcolano somme o differenze.

v			
149	+	010010101	$128+16+4+1$
-95		101011111	$64+16+8+4+2+1$

54		000110110	$32+16+4+2$
		^	
			0 - 0 = 0
			1 - 0 = 1
			1 - 1 = 0
			0 - 1 = 1 (prestato 1 col. sin.)

Codifica Complemento a 2

- Di gran lunga la più usata
- Un numero negativo è ottenuto calcolando il suo complemento (si invertono zeri e uni) e poi aggiungendo 1
- I numeri rappresentati appartengono all'intervallo $[-2^{n-1}, 2^{n-1} - 1]$ (ad es. $[-2147483648, 2147483647]$ con 32 bit)
- Rappresentazione ciclica: $-X$ in complemento a 2 si scrive come si scriverebbe $2^n - X$ nella rappresentazione senza segno
- comporta un'unica rappresentazione dello zero: 00000000

Esempio:

- 11110111 in complemento a 2 rappresenta -9:

$$\begin{array}{ccc} 9 & \text{compl.} & +1 \\ 00001001 & 11110110 & 11110111 \end{array}$$

- 11110111 in rappresentazione senza segno rappresenta 247, cioè $2^8 - 9$

Interi senza segno vs. complemento a 2

Codifica	senza segno	complemento a 2		
00000000		0	0	
00000001		1	1	
⋮		⋮	⋮	
00001001	$8 + 1 =$	9	$8 + 1 =$	9
⋮		⋮	⋮	
01111111	$64 + \dots + 2 + 1 =$	127	$64 + \dots + 2 + 1 =$	127
10000000		128	$128 - 256 =$	-128
10000001	$128 + 1 =$	129	$128 + 1 - 256 =$	-127
⋮		⋮	⋮	
10001001	$128 + 8 + 1 =$	137	$128 + 8 + 1 - 256 =$	-119
⋮		⋮	⋮	
11111111	$128 + \dots + 1 =$	255	$128 + \dots + 1 - 256 =$	-1

Esempi: Operazioni in Complemento a 2

-9 + 11110111
9 00001001
= 0 00000000

-9 + 11110111
8 00001000
= -1 11111111 => 11111110 => 00000001

-9 + 11110111
10 00001010
= 1 00000001

-9 + 11110111
-5 11111011
= -14 11110010 => 11110001 => 00001110

Conversione da Decimale con Virgola a Binario con Virgola

- Anche i numeri binari si possono esprimere con la virgola!
- Convertire 3.5_{10} in binario con virgola
 - Parte intera: $3_{10} \rightarrow 011_2$
 - Parte frazionaria: $0.5_{10} \rightarrow 2^{-1} \rightarrow 0.1_2$
 - Risultato: 11.1_2
- Prova inversa:
 - $11_2 \rightarrow 2^1 + 2^0 \rightarrow 2 + 1 \rightarrow 3_{10}$
 - $0.1_2 \rightarrow 2^{-1} \rightarrow 0.5_{10}$
 - Risultato: $3 + 0.5 = 3.5_{10}$

Numeri Reali

Rappresentazione in virgola mobile (standard IEEE 754):

- $s \cdot m \cdot 2^{e-o}$, s è il **segno** ($\{-1, +1\}$), m è la **mantissa**, e è l'**esponente** e o è l'**offset**

Formato	segno	esponente	mantissa	offset	range	precisione
32 bit	1 bit	8 bit	23 bit	$2^7 - 1$	$\approx [10^{-38}, 10^{38}]$	≈ 6 cifre dec.
64 bit	1 bit	11 bit	52 bit	$2^{10} - 1$	$\approx [10^{-308}, 10^{308}]$	≈ 16 cifre dec.
128 bit	1 bit	15 bit	112 bit	$2^{14} - 1$	$\approx [10^{-4932}, 10^{4932}]$	≈ 34 cifre dec.

(nota pratica: $2^{10} = 1024 \approx 1000 \implies 2^{10 \cdot N} \approx 10^{3 \cdot N}$)

- Per la mantissa uso di codifica binaria decimale
Es: “.100110011001...” significa $1 \cdot \frac{1}{2} + 0 \cdot \frac{1}{4} + 0 \cdot \frac{1}{8} + 1 \cdot \frac{1}{16} + \dots$
nota: (rappresentazione normale) primo bit assunto essere 1:
Es: “000100001...” significa “1.000100001...”
- La codifica dell'esponente è dipendente dal numero di bit e dalla particolare rappresentazione binaria (tipicamente codifica binaria senza segno)
- Richiede algoritmi ad-hoc per eseguire le operazioni

Conversione IEEE754

- Convertire il numero -10.75_{10} in floating point a 32 bit (singola precisione)

10	0	0.75	↓
5	1	0.50	1 (sottraggo 1)
2	0	0.	1 (sottraggo 1)
1	1	0.	

- Quindi $-1010.11_2 \rightarrow -1.01011_2 * 2^3$ in notazione scientifica
- $(-1)^s * (1 + Mantissa) * 2^{Esponente - 127}$
 - $s = 1$
 - $Mantissa = 1.01011 - 1 = 0.01011$
 - $3 = (Esponente - 127)$ quindi Esponente = 130 $\rightarrow 10000010$

s	Esp. 8bit	Mantissa 23 bit
1	10000010	0101100000000000000000000

Conversione IEEE754

- Convertire il numero 0.1875_{10} in floating point a 32 bit (singola precisione)

	0.1875	↓
0 0	0.3750	0
	0.7500	0
	0.5000	1 (sottraggo 1)
	0.0000	1 (sottraggo 1)

- Quindi $0.0011 \rightarrow 1.1_2 * 2^{-3}$ in notazione scientifica
- $(-1)^s * (1 + Mantissa) * 2^{Esponente - 127}$
 - $s = 0$
 - $Mantissa = 1.1 - 1 = 0.1$
 - $-3 = (Esponente - 127)$ quindi Esponente = 124 $\rightarrow 01111100$

s	Esp. 8bit	Mantissa 23 bit
001111100	10000000000000000000000000000000	

Conversione IEEE754

- Convertire il numero IEEE754 $0x427d0000_{16}$ in decimale virgola mobile
 - $0x427d0000_{16} = 0100\ 0010\ 0111\ 1101\ 0000\ 0000\ 0000\ 0000_2$
 $0\mid 10000100\mid 11111010000000000000000000000000_2$
 - $N = (-1)^s * (1 + \text{Mantissa}) * 2^x$ dove $x = \text{Esponente} - 127$
 - $s = 0$
 - $\text{Esponente} = 10000100 = 2^7 + 2^2 = 132$
 $x = \text{Esponente} - 127 = 132 - 127 = 5$
 - $\text{Mantissa} = 111110100000000000000000 = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-7} = 0.9765625$
 - $N = (-1)^0 * (1 + 0.9765625) * 2^5 = 1 * 1.9765625 * 32 = 63.25$

- Oggetti dello stesso **tipo**
 - utilizzano lo stesso spazio in memoria e la stessa codifica
 - sono soggetti alle stesse operazioni, con lo stesso significato
- Vantaggi sull'uso dei tipi:
 - correttezza semantica
 - efficiente allocazione della memoria dovuta alla conoscenza dello spazio richiesto in fase di compilazione

Tipi Fondamentali e Derivati in C++

Nel C++ i tipi sono distinti in:

- i **tipi fondamentali**
 - che servono a rappresentare informazioni semplici
 - Esempio: i **numeri interi** o i **caratteri** (int, char, ...)
- i **tipi derivati**
 - permettono di costruire strutture dati complesse
 - si costruiscono a partire dai tipi fondamentali, mediante opportuni costruttori (**array**, **puntatori**, ...)

I Tipi Fondamentali in C++

- i tipi **interi**: int, short, long, long long
- i tipi **Booleani**: bool
- i tipi **enumerativi**: enum
- il tipo **carattere**: char
- i tipi **reali**: float, double, long double

I primi quattro sono detti tipi **discreti** (hanno un dominio finito)

I tipi interi (con segno)

- I tipi interi (con segno) sono costituiti da numeri interi compresi tra due valori estremi, a seconda dell'implementazione
 - tipicamente codificati in **complemento a 2** con N bit ($N = 16, 32, \dots$)
 - appartengono all'intervallo $[-2^{N-1}, 2^{N-1} - 1]$
- Quattro tipi, in ordine crescente di dimensione
 - short [int], int, long [int], long long [int]
- Dimensioni dipendenti dall'implementazione (macchina, s.o., ...)
 - $\text{sizeof(short)} \leq \text{sizeof(int)} \leq \text{sizeof(long)} \leq \text{sizeof(long long)}$
(`sizeof()` restituisce la dimensione del tipo in byte)
 - short tipicamente non ha più di 16 bit: $[-32768, 32767]$
 - long tipicamente ha almeno 32 bit: $[-2147483648, 2147483647]$

Il valore di un'espressione intera non esce dal range $[-2^{N-1}, 2^{N-1} - 1]$ perché i valori "ciclano": il successore di **2147483647** è **-2147483648**

Esempio di short, int, long, long long:

```
{ ..../ESEMPI_BASE/shortlong.cc }
```

I tipi interi (senza segno)

- Il tipo `unsigned` . . . rappresenta numeri interi non negativi di varie dimensioni
 - codifica interi senza segno a N bit ($N=16,32,\dots$), range $[0, 2^N - 1]$
 - tipicamente usati molto poco, come rappresentazioni di sequenze di bit (applicazioni in elettronica)

```
unsigned int x=1232;  
unsigned short int x=567;  
unsigned long int x=878678687;
```

Il valore di un'espressione `unsigned` non esce mai dal range $[0, 2^N - 1]$ perché, quando il valore aritmetico X esce da tale range, il valore restituito è X modulo 2^N .

Esempio di `unsigned`:

```
{ ./ESEMPI_BASE/unsigned.cc }
```

Operatori aritmetici sugli interi

operatore binario (infisso)	significato
+	addizione
-	sottrazione
*	moltiplicazione
/	divisione intera
%	resto della div. intera
operatore unario (prefisso)	significato
-	inversione di segno

Nota:

La divisione è la divisione intera: $5/2$ è 2, non 2.5!

Esempio sugli operatori aritmetici sugli interi:

```
{ ..../ESEMPI_BASE/operatori_aritmetici.cc }
```

Operatori bit-a-bit (su interi senza segno)

operatore	esempio	significato
<code>>></code>	<code>x>>n</code>	shift a destra di n posizioni
<code><<</code>	<code>x<<n</code>	shift a sinistra di n posizioni
<code>&</code>	<code>x&y</code>	AND bit a bit tra x e y
<code> </code>	<code>x y</code>	OR bit a bit tra x e y
<code>^</code>	<code>x^y</code>	XOR bit a bit tra x e y
<code>~</code>	<code>~x</code>	NOT, complemento bit a bit

Nota:

- `~`: restituisce un intero signed anche se l'input è unsigned

Operatori bit-a-bit : esempio

Siano x e y rappresentati su 16 bit

x:	0000000000001100	(12)
y:	0000000000001010	(10)
x y:	0000000000001110	(14)
x & y:	0000000000001000	(8)
x ^ y:	0000000000001110	(6)
~x:	1111111111110011	(65523 oppure -13)
x>>2:	0000000000000011	(3)
x<<2:	000000000110000	(48)

Esempio sugli operatori bit-a-bit su interi senza segno:

{ .. /ESEMPI_BASE/operatori_bitwise.cc }

Operatore di Assegnazione

- Sintassi dell'operatore di assegnazione: `exp1 = exp2`
 - `exp1` deve essere un'espressione dotata di indirizzo (l-value)
 - `exp1` e `exp2` devono essere di tipo compatibile
- Il valore di `exp2` viene valutato e poi assegnato a `exp1`

Esempio di assegnazioni:

```
{ ./ESEMPI_BASE/assegnazione_errori.cc }
```
- Un'assegnazione può occorrere dentro un'altra espressione.
 - Il valore di un'espressione di assegnazione è il valore di `exp2`.
 - L'operazione di assegnazione, `'=`, associa a destra.
- Esempio:
`a = b = c = d = 5;`
è equivalente a:
`(a=(b=(c=(d=5))));`

Esempio di assegnazioni multiple:

```
{ ./ESEMPI_BASE/assegnazione.cc }
```

Operatori misti assegnazione/aritmetica

Forma compatta	Forma estesa
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$
$x \% y$	$x = x \% y$

L'uso della forma compatta è tipicamente più efficiente
(su alcune architetture consente di utilizzare in modo ottimale funzionalità della CPU)

Esempi di operatori di assegnazione misti:

```
{ ..../ESEMPI_BASE/op_assegnazione.cc }
```

Operatori di incremento e decremento unitario

- $x++$:
 - incrementa x di un'unità
 - denota il valore di x **prima** dell'incremento
- $x--$:
 - decrementa x di un'unità
 - denota il valore di x **prima** dell'incremento
- $++x$:
 - incrementa x di un'unità
 - denota il valore di x **dopo** l'incremento
- $--x$:
 - decrementa x di un'unità
 - denota il valore di x **dopo** l'incremento

Nota

L'uso della forma compatta: “ $x++;$ ” è tipicamente più efficiente della forma estesa corrispondente:

“ $x = x + 1;$ ”

Esempi di operatori di incremento unitario:

{ .../ESEMPI_BASE/op_incremento.cc }

Operatori Relazionali

Operatore	Significato
<code>==</code>	uguale
<code>!=</code>	diverso
<code><=</code>	minore o uguale
<code>>=</code>	maggiore o uguale
<code><</code>	minore
<code>></code>	maggiore

Ordine di valutazione di un'espressione

- In C++ *non* è specificato l'ordine di valutazione degli operandi di ogni operatore, in particolare:
 - l'ordine di valutazione di sottoespressioni in un'espressione
 - (l'ordine di valutazione degli argomenti di una funzione)
- Es: nel valutare `expr1 <op> expr2`, non è specificato se `expr1` venga valutata prima di `expr2` o viceversa
(Con alcune importanti eccezioni, vedi operatori Booleani)
- **Problematico** quando sotto-espressioni contengono operatori con “side-effects” (e.g. gli operatori di incremento). Esempio:

```
j = i++ * i++; // undefined behavior
i = ++i + i++; // undefined behavior
```

⇒ **evitare l'uso di operatori con side-effects** in sotto-espressioni

Per approfondimenti si veda ad esempio

http://en.cppreference.com/w/cpp/language/eval_order

Il tipo Booleano



Il tipo Booleano

Il C++ prevede un tipo Booleano `bool`:

- il valore **falso** è rappresentato dalla costante `false` (equivalente a 0)
- il valore **vero** è rappresentato dalla costante `true` (equivalente ad un valore intero diverso da 0)
- si può usare a tal scopo anche il tipo `int`
- **Operatori Booleani:** `!` (not), `&&` (and), `||` (or)

x	y	<code>!x</code>	<code>&&</code>	<code> </code>
<code>false</code>	<code>false</code>	<code>true</code>	<code>false</code>	<code>false</code>
<code>false</code>	<code>true</code>	<code>true</code>	<code>false</code>	<code>true</code>
<code>true</code>	<code>false</code>	<code>false</code>	<code>false</code>	<code>true</code>
<code>true</code>	<code>true</code>	<code>false</code>	<code>true</code>	<code>true</code>

- $!!x \iff x$,
- $!(x \mid\mid y) \iff (\neg x \And \neg y)$,
- $!(x \And y) \iff (\neg x \mid\mid \neg y)$,

Il tipo Booleano (II)

Nota 1: Priorità degli operatori Booleani

- l'operatore `!` ha priorità sugli operatori `&&` e `||`:
`!x && y` è equivalente a `(!x) && y`, non a `! (x && y)`

Nota 2: Lazy Evaluation

- In C++ `&&` e `||` sono valutati in modi “pigro” (**lazy evaluation**):
 - in `(exp1 && exp2)`, se `exp1` è valutata `false`, allora `exp2` non viene valutata
 - idem se `exp1` è valutata `true` in `(exp1 || exp2)`
 - l'ordine di valutazione tra `exp1` e `exp2` è da sinistra a destra
- ⇒ più efficiente
- ⇒ può causare **seri effetti collaterali** se usata con espressioni che modificano valori di variabili (es “`++`”)
- ⇒ evitare di usare tali costrutti all'interno di espressioni Booleane

Il tipo Booleano (esempi)

Esempio di valori Booleani rappresentati con `bool`:

```
{ ..../ESEMPI_BASE/booleano_bool.cc }
```

Esempio di lazy evaluation con operatori Booleani:

```
{ ..../ESEMPI_BASE/booleano_sideeffects.cc }
```

Corso “Programmazione 1”

Capitolo 02: Variabili, Costanti, Tipi

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato@unitn.it¹

¹ In via di definizione

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 24 settembre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Operatore di Assegnazione

- Sintassi dell'operatore di assegnazione: `exp1 = exp2`
 - `exp1` deve essere un'espressione dotata di indirizzo (l-value)
 - `exp1` e `exp2` devono essere di tipo compatibile
- Il valore di `exp2` viene valutato e poi assegnato a `exp1`

Esempio di assegnazioni:

```
{ ./ESEMPI_BASE/assegnazione_errori.cc }
```
- Un'assegnazione può occorrere dentro un'altra espressione.
 - Il valore di un'espressione di assegnazione è il valore di `exp2`.
 - L'operazione di assegnazione, `'=`, associa a destra.
- Esempio:
`a = b = c = d = 5;`
è equivalente a:
`(a=(b=(c=(d=5))));`

Esempio di assegnazioni multiple:

```
{ ./ESEMPI_BASE/assegnazione.cc }
```

Operatori misti assegnazione/aritmetica

Forma compatta	Forma estesa
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$
$x \% y$	$x = x \% y$

L'uso della forma compatta è tipicamente più efficiente
(su alcune architetture consente di utilizzare in modo ottimale funzionalità della CPU)

Esempi di operatori di assegnazione misti:

```
{ ..../ESEMPI_BASE/op_assegnazione.cc }
```

Operatori di incremento e decremento unitario

- $x++$:
 - incrementa x di un'unità
 - denota il valore di x **prima** dell'incremento
- $x--$:
 - decrementa x di un'unità
 - denota il valore di x **prima** dell'incremento
- $++x$:
 - incrementa x di un'unità
 - denota il valore di x **dopo** l'incremento
- $--x$:
 - decrementa x di un'unità
 - denota il valore di x **dopo** l'incremento

Nota

L'uso della forma compatta: “ $x++;$ ” è tipicamente più efficiente della forma estesa corrispondente:

“ $x = x + 1;$ ”

Esempi di operatori di incremento unitario:

{ .../ESEMPI_BASE/op_incremento.cc }

Operatori Relazionali

Operatore	Significato
<code>==</code>	uguale
<code>!=</code>	diverso
<code><=</code>	minore o uguale
<code>>=</code>	maggiore o uguale
<code><</code>	minore
<code>></code>	maggiore

Ordine di valutazione di un'espressione

- In C++ *non* è specificato l'ordine di valutazione degli operandi di ogni operatore, in particolare:
 - l'ordine di valutazione di sottoespressioni in un'espressione
 - (l'ordine di valutazione degli argomenti di una funzione)
- Es: nel valutare `expr1 <op> expr2`, non è specificato se `expr1` venga valutata prima di `expr2` o viceversa
(Con alcune importanti eccezioni, vedi operatori Booleani)
- **Problematico** quando sotto-espressioni contengono operatori con “side-effects” (e.g. gli operatori di incremento). Esempio:

```
j = i++ * i++; // undefined behavior
i = ++i + i++; // undefined behavior
```

⇒ **evitare l'uso di operatori con side-effects** in sotto-espressioni

Per approfondimenti si veda ad esempio

http://en.cppreference.com/w/cpp/language/eval_order

Il tipo Booleano



Il tipo Booleano

Il C++ prevede un tipo Booleano `bool`:

- il valore **falso** è rappresentato dalla costante `false` (equivalente a 0)
- il valore **vero** è rappresentato dalla costante `true` (equivalente ad un valore intero diverso da 0)
- si può usare a tal scopo anche il tipo `int`
- **Operatori Booleani:** `!` (not), `&&` (and), `||` (or)

x	y	<code>!x</code>	<code>&&</code>	<code> </code>
<code>false</code>	<code>false</code>	<code>true</code>	<code>false</code>	<code>false</code>
<code>false</code>	<code>true</code>	<code>true</code>	<code>false</code>	<code>true</code>
<code>true</code>	<code>false</code>	<code>false</code>	<code>false</code>	<code>true</code>
<code>true</code>	<code>true</code>	<code>false</code>	<code>true</code>	<code>true</code>

- $!!x \iff x$,
- $!(x \mid\mid y) \iff (\neg x \And \neg y)$,
- $!(x \And y) \iff (\neg x \mid\mid \neg y)$,

Il tipo Booleano (II)

Nota 1: Priorità degli operatori Booleani

- l'operatore `!` ha priorità sugli operatori `&&` e `||`:
`!x && y` è equivalente a `(!x) && y`, non a `! (x && y)`

Nota 2: Lazy Evaluation

- In C++ `&&` e `||` sono valutati in modi “pigro” (**lazy evaluation**):
 - in `(exp1 && exp2)`, se `exp1` è valutata `false`, allora `exp2` non viene valutata
 - idem se `exp1` è valutata `true` in `(exp1 || exp2)`
 - l'ordine di valutazione tra `exp1` e `exp2` è da sinistra a destra
- ⇒ più efficiente
- ⇒ può causare **seri effetti collaterali** se usata con espressioni che modificano valori di variabili (es “`++`”)
- ⇒ evitare di usare tali costrutti all'interno di espressioni Booleane

Il tipo Booleano (esempi)

Esempio di valori Booleani rappresentati con `bool`:

```
{ ..../ESEMPI_BASE/booleano_bool.cc }
```

Esempio di lazy evaluation con operatori Booleani:

```
{ ..../ESEMPI_BASE/booleano_sideeffects.cc }
```

I Tipi Reali

- I tipi reali hanno come insieme di valori un sottoinsieme dei numeri reali, ovvero quelli rappresentabili all'interno del computer in un formato prefissato
- Ne esistono vari tipi, in ordine crescente di precisione:
 - `float`
 - `double`
 - `long double`
- Operatori aritmetici: `+, -, *, /`
(" / " diverso da divisione tra interi: $7.0/2.0 = 3.5$)
- Precisione e occupazione di memoria dipendono dalla macchina

Nota

- in realtà sottoinsieme dei **razionali**
- anche i tipi reali hanno un range finito!

I Tipi Reali: esempi

```
double a = 2.2, b = -14.12e-2;  
double c = .57, d = 6.;  
  
float g = -3.4F; // literal float  
float h = g-.89F; // suffisso F (f)  
  
long double i = +0.001;  
long double j = 1.23e+12L;  
// literal long double  
// suffisso L (l)
```

Esempi con i tipi reali:

```
{ ..../ESEMPI_BASE/reali.cc }
```

Esempi di confronto tra tipi reali e interi:

```
{ ..../ESEMPI_BASE/reali_vs_interi.cc }
```

Precisione dei tipi reali

- La rappresentazione dei numeri reali ha intrinseci limiti di **precisione**, dovuti a:
 - limitato numero di bit nella rappresentazione della mantissa (vedi codifica floating-point)
 - uso di codifica binaria nei decimali
 - Es: “.100110011001” significa $1 \cdot \frac{1}{2} + 0 \cdot \frac{1}{4} + 0 \cdot \frac{1}{8} + 1 \cdot \frac{1}{16} + \dots$
 - ⇒ alcuni numeri non hanno rappresentazione esatta
 - (Es: 0.1, 11.1,...):
- ⇒ Intrinseca sorgente di errori di precisione
- talvolta non visualizzabili con “`cout << ...;`”;
 - confronto con “`... == ...`” tra tipi reali spesso problematico

Problemi di precisione:

{ `../ESEMPI_BASE/reali_precisione.cc` }

Il Tipo Enumerato

- Un tipo enumerato è un insieme finito di costanti intere, definite dal programmatore, ciascuna individuata da un identificatore
- Sintassi: `enum typeid { id_or_init1, ..., id_or_initn }`
- Se non specificato esplicitamente, i valori sono equivalenti rispettivamente agli interi `0, 1, 2, ...`
- Ad una variabile di tipo enumerativo è possibile assegnare solo un valore del tipo enumerativo
- I valori vengono stampati come interi!

Esempi di uso di enumerativi:

```
{ ..../ESEMPI_BASE/enum.cc }
```

Il tipo Carattere I

- Il tipo `char` ha come insieme di valori i caratteri stampabili
 - es. 'a', 'Y', '6', '+', ' '
 - generalmente un carattere occupa 1 byte (8 bit)
- Il tipo `char` è un sottoinsieme del tipo `int`
- Il **valore numerico** associato ad un carattere è detto **codice** e dipende dalla **codifica** utilizzata dal computer
 - es. ASCII, EBCDIC, BCD, ...
 - la più usata è la codifica ASCII

Il tipo Carattere II

Nota importante:

Il tipo char è indipendente dalla particolare codifica adottata!

⇒ un programma deve funzionare sempre nello stesso modo, indipendentemente dalla codifica usata nella macchina su cui è eseguito!!

⇒ evitare di far riferimento al valore ASCII di un carattere:

```
char c;  
c = 65; // NO!!!!!!  
c = 'A'; // SI
```

Codifica dei caratteri: regole generali

Qualunque codifica deve soddisfare le seguenti regole

- Precedenza:

- 'a' < 'b' < ... < 'z'
- 'A' < 'B' < ... < 'Z'
- '0' < '1' < ... < '9'

- La consecutività tra lettere minuscole, lettere maiuscole, numeri

- 'a', 'b', ..., 'z'
- 'A', 'B', ..., 'Z'
- '0', '1', ..., '9'

Nota

Non è fissa la relazione tra maiuscole e minuscole o fra i caratteri non alfabetici

La codifica ASCII

0	NUL	1	^A	2	^B	3	^C	4	^D	5	^E	6	^F	7	^G
8	^H	9	^I	10	^J	11	^K	12	^L	13	^M	14	^N	15	^O
16	^P	17	^Q	18	^R	19	^S	20	^T	21	^U	22	^V	23	^W
24	^X	25	^Y	26	^Z	27	^[28	^`	29	^]	30	^`	31	^-
32	SP	33	!	34	"	35	#	36	\$	37	%	38	&	39	'
40	(41)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[92	\	93]	94	^	95	_
96	'	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124		125	}	126	~	127	DEL

Esempio sull'uso dei caratteri

```
char c = 'f';
n = '\n';
char l = 'a';
((l >= 'a') && (l <= 'z')) // test: l e' una
                                // lettera minuscola?
l += 3;           // l diventa 'd'
l--;
l = 'a' - 'A'; // l diventa 'C'
```

Nota

È possibile applicare operatori aritmetici agli oggetti di tipo char!

Esempio sull'uso di char:

```
{ ..../ESEMPI_BASE/char.cc }
```

L'Operatore sizeof

- L'operatore sizeof, può essere prefisso a:
 - una variabile (esempio: `sizeof x`)
 - una costante (esempio: `sizeof 'a'`)
 - al nome di un tipo (esempio: `sizeof double`)
- Può essere usato in alcuni casi con o senza parentesi
 - `sizeof x`
 - `sizeof double`
- Restituisce un intero rappresentante la dimensione in **byte**
- È applicabile a espressioni di qualsiasi tipo (non solo ai tipi fondamentali)

Ok

Ko

Esempio di uso di `sizeof` applicato a tipi:

```
{ ..../ESEMPI_BASE/sizeof.cc }
```

Esempio di uso di `sizeof` applicato a espressioni:

```
{ ..../ESEMPI_BASE/sizeof2.cc }
```

Operazioni miste e conversioni di tipo

- Spesso si usano operandi di tipo diverso in una stessa espressione o si assegna ad una variabile un valore di tipo diverso della variabile stessa
- In ogni operazione mista è sempre necessaria una conversione di tipo che può essere
 - implicita
 - esplicita

Esempio:

```
int prezzo = 27500;  
double peso = 0.3;  
int costo = prezzo * peso;
```

Conversioni Implicite

- Le conversioni implicite vengono effettuate dal compilatore
- Le conversioni implicite più significative sono:
 - nella valutazione di espressioni numeriche, gli operandi sono convertiti al tipo di quello di dimensione maggiore
 - nell'assegnazione, un'espressione viene sempre convertita al tipo della variabile

Esempi:

```
float x = 3;    // equivale a: x = 3.0
int y = 2*3.6; // equivale a: y = 7
```

Esempio di conversioni implicite:

```
{ ..../ESEMPI_BASE/conversioni_err.cc }
```

Esempio di conversioni implicite:

```
{ ..../ESEMPI_BASE/conversioni_corr.cc }
```

Conversioni Esplicite

- Il programmatore può richiedere una **conversione esplicita** di un valore da un tipo ad un altro (**casting**)
- Esistono due notazioni:
 - prefissa**. Esempio:
`int i = (int) 3.14;`
 - funzionale**. Esempio:
`double f = double(3)`

Conversioni tra tipi numerici - approfondimenti

- **Promozione:** conversione da un tipo ad uno simile più grande

- $\text{short} \Rightarrow \text{int} \Rightarrow \text{long} \Rightarrow \text{long long}$,
 $\text{float} \Rightarrow \text{double} \Rightarrow \text{long double}$
- garantisce di mantenere lo stesso valore

- **Conversioni tra tipi compatibili**

- Conversione da tipo reale a tipo intero: il valore viene **troncato**

```
int x = (int) 4.7 => x==4,  
int x = (int) -4.7 => x== -4
```

- Conversione da tipo intero a reale: il valore può perdere precisione

```
float y = float(2147483600); // 2^31-48  
=> y == 2147483648.0; // 2^31
```

Esempio di conversioni tra tipi numerici:

```
{ .../ESEMPI_BASE/conversioni_miste.cc }
```

Esempio di conversioni tra tipi reali:

```
{ .../ESEMPI_BASE/conversioni_real.cc }
```

Per approfondimenti vedere, ad esempio:

<http://www.cplusplus.com/doc/tutorial/typecasting/>

Corso “Programmazione 1”

Capitolo 03: Istruzioni

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato@unitn.it¹

¹ In via di definizione

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 28 settembre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Struttura di un programma

- Un programma consiste in un insieme di funzioni, eventualmente suddivise in più file
 - La funzione che costituisce il programma principale si deve necessariamente chiamare `main`
- Ogni programma contiene una lista di istruzioni di ogni tipo:
istruzioni semplici o istruzioni strutturate

Esempio:

```
(...)  
int main() {  
    int x=2, y=6, z;  
    z=x*y;  
    return 0;  
}
```

Istruzioni semplici

- Le **istruzioni semplici** sono la base delle istruzioni più complesse (**istruzioni strutturate**)
- Sono sempre terminate da un punto-e-virgola “;”
- Si distinguono in:
 - definizioni/dichiarazioni** (declaration-statement) di
 - variabili, es. `int x, y, z;`
 - costanti, es. `const int kilo=1024;`
 - espressioni** (expression-statement)
 - di input, es: `cin >> x`
 - di output, es: `cout << 3*x`
 - di assegnamento, es. `x=2*(3-y)`
 - matematiche, es. `(x-3)*sin(x)`
 - logiche, es. `x==y && x!=z`
 - costanti, es. `3*12.7`
 - condizionali (a seguire)

ogni espressione seguita da un “;” è anche un’istruzione

L'espressione condizionale

- Sintassi: `exp1 ? exp2 : exp3 :`
Se `exp1` è vera equivale a `exp2`, altrimenti equivale a `exp3`

Esempio

```
prezzo = valore * peso * (peso>10) ? 0.9 : 1;
```

- se il peso è maggiore di 10, equivale a:
`prezzo = valore * peso * 0.9;`
- altrimenti, equivale a
`prezzo = valore * peso * 1;`

Esempio di uso di espressione condizionale:

```
{ IF_THEN_ELSE_SWITCH/espressione_condizionale.cc }
```

Istruzioni strutturate

- Le **istruzioni strutturate** consentono di specificare azioni complesse
- Si distinguono in
 - **istruzione composta** (compound-statement)
 - **istruzioni condizionali** (conditional-statement)
 - **istruzioni iterative** (iteration-statement)
 - **istruzioni di salto** (jump-statement)

Istruzione Composta

- Trasforma una sequenza di istruzioni in una singola istruzione
 - sequenza delimitata per mezzo della coppia di delimitatori '{' e '}'
 - la sequenza così delimitata è detta **blocco**
- Le definizioni possono comparire in qualunque punto del blocco
 - sono visibili solo all'interno del blocco
 - possono accedere ad oggetti definiti esternamente
 - in caso di identificatori identici, prevale quello più interno

```
{  
    int a=4;  
    a*=6;  
    char b= 'c' ;  
    b+=3;  
}
```

Esempio di blocchi e visibilità:

```
{ IF_THEN_ELSE_SWITCH/visibilita.cc }
```

L'Istruzione Condizionale if-then

- Istruzione “if” semplice (if-then):

- Sintassi:

- if** (exp)
 istruzione1

- Significato: se exp è vera, viene eseguita istruzione1, altrimenti non viene eseguito nulla

- istruzione1 può a sua volta essere un’istruzione complessa

Esempio

```
if (x!=0)  
    y=1/x;
```

Esempio di if-then:

```
{ IF_THEN_ELSE_SWITCH/divisibilita.cc }
```

L'Istruzione Condizionale if-then-else

- Istruzione “if” composta (if-then-else):
 - Sintassi: `if (exp) istruzione1 else istruzione2`
 - Significato: se `exp` è vera, viene eseguita `istruzione1`, altrimenti viene eseguita `istruzione2`
- `istruzione1` e `istruzione2` possono essere a loro volta istruzioni complesse (un blocco, un’altro if-then-else, ...)

Esempio

```
if (x<0)
    y=-x ;
else
    y=x ;
```

Esempio di if-then-else:

```
{ IF_THEN_ELSE_SWITCH/divisibilita2.cc }
```

If annidati

- Nei costrutti if-then e if-then-else, istruzione1 e istruzione2 possono essere a loro volta istruzioni complesse (un blocco, un'altro if-then-else, ...)
- L'annidamento di if-then-else e l'uso di operatori logici permettono di costruire strutture decisionali complesse
 - Uso di if-then-else annidati, ...:
 { IF_THEN_ELSE_SWITCH/eq_1grado.cc }
 - ..., con diverso ordine, ...:
 { IF_THEN_ELSE_SWITCH/eq_1grado2.cc }
 - ... e con operatori logici:
 { IF_THEN_ELSE_SWITCH/eq_1grado3.cc }

L'indentazione del codice è importantissima!!!

- individua a colpo d'occhio l'inizio e la fine del codice/blocco
- contribuisce grandemente alla leggibilità del codice

If-then-else annidati: esempi

- Esempi di alternative all'utente:
 { IF_THEN_ELSE_SWITCH/conversione2.cc }
- Esempio di “Dangling else”:
 { IF_THEN_ELSE_SWITCH/dangling_else.cc }
- Esempio di “Dangling else” (2):
 { IF_THEN_ELSE_SWITCH/dangling_else2.cc }
- Errore tipico con “if”:
 { IF_THEN_ELSE_SWITCH/ifeq_err.cc }
- Versione corretta:
 { IF_THEN_ELSE_SWITCH/ifeq_corr.cc }
- Minimo tra due numeri:
 { IF_THEN_ELSE_SWITCH/minimo.cc }
- Minimo tra tre numeri:
 { IF_THEN_ELSE_SWITCH/minimo2.cc }
- Scelte tra valori multipli:
 { IF_THEN_ELSE_SWITCH/simple_calc.cc }

L'Istruzione Condizionale `switch`

- Sintassi

```
switch (exp) {  
    case const-exp1: istruzione1 break;  
    case const-exp2: istruzione2 break;  
    ...  
    default: istruzione-default  
}
```

- L'esecuzione dell'istruzione `switch` consiste

- nel calcolo dell'espressione `exp`
- nell'esecuzione dell'istruzione corrispondente all'alternativa specificata dal valore calcolato
- se nessuna alternativa corrisponde, se esiste, viene eseguita `istruzione-default`

Scelte tra valori multipli con `switch`:

```
{ IF_THEN_ELSE_SWITCH/simple_calc2.cc }
```

Scelte multiple con switch

- Se dopo l'ultima istruzione di un'alternativa non c'è un `break`, viene eseguita anche l'alternativa successiva
- Questo comportamento è **sconsigliato** ma può essere giustificato in alcuni casi

Esempio

```
switch (giorno)
{ case lun: case mar:
  case mer: case gio:
  case ven: orelavorate+=8; break;
  case sab: case dom: break;
}
```

Esercizi Proposti

Esercizio su istruzioni condizionali:

{ IF_THEN_ELSE_SWITCH/ESERCIZI_PROPOSTI.txt }

L'Istruzione Iterativa `while` (while-do)

- **Sintassi:** `while (exp) istruzione`
 - `exp` è un'espressione Booleana
 - `istruzione` può essere un'istruzione complessa
- L'esecuzione dell'istruzione `while` comporta
 1. il calcolo dell'espressione `exp`
 2. se `exp` è vera, l'esecuzione di `istruzione` e la ripetizione dell'esecuzione dell'istruzione `while`
- `istruzione` **potrebbe non essere mai eseguita**
- È possibile generare **loop infiniti**.

Nota

`exp` tipicamente contiene almeno una variabile (**variabile di controllo** del ciclo), che viene modificata in `istruzione` per far convergere `exp` verso uno stato in cui diventi falsa.

L'Istruzione Iterativa `while`: Esempi I

- ripetizione pedissequa di un'operazione (contatore crescente):
 { LOOPS/stampaciao.cc }
- ... (contatore decrescente):
 { LOOPS/stampaciao2.cc }
- ..., con loop infinito:
 { LOOPS/stampaciao_infloop.cc }
- somma con accumulatore:
 { LOOPS/sommainteri_while.cc }
- prodotto con accumulatore:
 { LOOPS/fact_while.cc }
- condizione di uscita diversa da conteggio:
 { LOOPS/divisibile.cc }

L'Istruzione Iterativa `while`: Esempi II

- ripetizione di comando a menu:
 { LOOPS/conversione3_while.cc }
- somma con accumulatore, con conteggio:
 { LOOPS/serie_while.cc }
- somma con accumulatore, con cond. uscita :
 { LOOPS/serie_while1.cc }
- uso di “cin loops”:
 { LOOPS/cin_loop.cc }
- stessa cosa, ma con fail:
 { LOOPS/cin_loop_equivalent.cc }

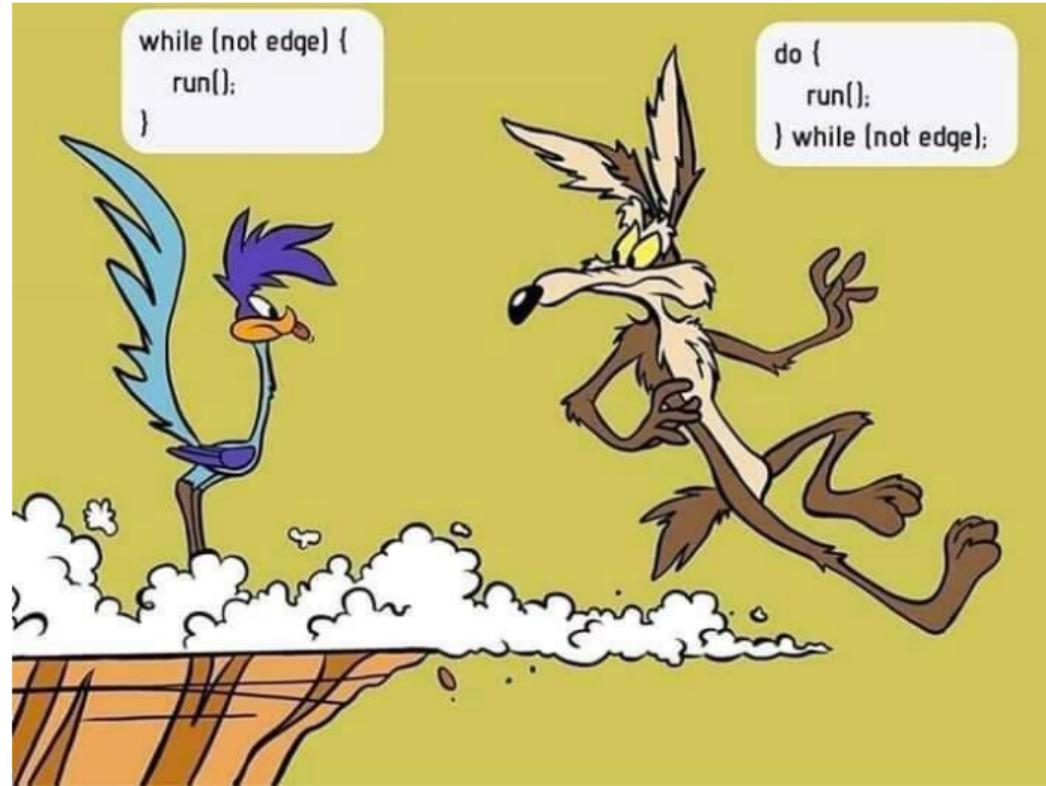
L'Istruzione Iterativa do (do-while)

- **Sintassi:** `do { istruzione } while (exp);`
 - `exp` è un'espressione Booleana
 - `istruzione` può essere un'istruzione complessa
- L'esecuzione dell'istruzione `do` comporta
 1. l'esecuzione di `istruzione`
 2. il calcolo dell'espressione `exp`
 3. se `exp` è vera, la ripetizione dell'esecuzione dell'istruzione `do`
- `istruzione` **viene sempre eseguita almeno una volta**
- è la meno usata tra le istruzioni iterative.

L'Istruzione Iterativa do: Esempi

- somma con accumulatore (do):
 { LOOPS/sommainteri_do.cc }
- ripetizione di comando a menu (do):
 { LOOPS/conversione3_do.cc }
- conversione di base:
 { LOOPS/base.cc }

While-Do vs. Do-While



©Warner Bros Inc.

Corso “Programmazione 1”

Capitolo 03: Istruzioni

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 5 ottobre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

L'Istruzione Iterativa **for**

- Sintassi: **for** (init; exp; agg) istruzione
 - init è un'istruzione di **inizializzazione** delle variabili di controllo
 - exp è un'espressione Booleana
 - istruzione può essere un'istruzione complessa
 - agg è un'istruzione di **aggiornamento** delle variabili di controllo
- L'esecuzione dell'istruzione **for** comporta:
 1. l'esecuzione di init
 2. il calcolo dell'espressione exp
 3. se exp è vera, viene eseguita istruzione, poi agg, e si ricomincia dal passo 2.
- è la più usata tra le istruzioni iterative.
- si possono definire variabili di controllo **interne al ciclo**:
for (**int** i=0; i<MAXDIM; i++) {<i occorre solo qui>}

Consente di separare le istruzioni di controllo del ciclo e concentrarle tutte in un'unica riga
⇒ miglior praticità e leggibilità del codice.

Cicli for e while

```
for ( init; exp; agg )
    istruzione
```

equivale a:

```
{ init;
while ( exp ) {
    istruzione
    agg;
}; }
```

Esempio

```
for (int i=1; i<10; i++)
    x*=2;
```

↔

```
{ int i=1;
while (i<10) {
    x*=2;
    i++;
}; }
```

L'Istruzione Iterativa `for`: Esempi I

- prodotto con accumulatore (`for`):

```
{ LOOPS/fact_for.cc }
```

- somma con accumulatore (numero iterazioni) (`for`):

```
{ LOOPS/serie_for.cc }
```

- somma con accumulatore (cond. uscita) (`for`):

```
{ LOOPS/serie_for1.cc }
```

- `for` annidati:

```
{ LOOPS/doublefor.cc }
```

L'Istruzione Iterativa `for`: Esempi II

- condizione iniziale multipla con `for`:

```
{ LOOPS/serie_for1_2init.cc }
```

- cond. iniziale multipla & uscita multipla con `for`:

```
{ LOOPS/serie_for1_2init2.cc }
```

- incremento come input dato dall'utente:

```
{ LOOPS/minmax.cc }
```

- doppio incremento:

```
{ LOOPS/doublecontrol.cc }
```

Gli Invarianti di un Ciclo (Loop Invariant)

- Tecnica per la verifica di correttezza dei cicli (proprietà P)
- Idea: suddividere la proprietà desiderata P della correttezza del ciclo in una sequenza di affermazioni P_0, P_1, \dots, P_n , in modo che:
 - (1) P_0 sia vera immediatamente prima che il ciclo inizi (dopo l'inizializzazione!)
 - (2) per ogni indice di ciclo $i \in \{1, \dots, n\}$:
se P_{i-1} è vera prima dell'inizio del ciclo i -esimo (ed è verificata la condizione di permanenza del ciclo), allora P_i è vera alla fine del ciclo i -esimo
(e quindi immediatamente prima dell'inizio del ciclo $(i+1)$ -esimo)
 - (3) Alla fine dell'ultimo ciclo (n -esimo), P_n (e la negazione della condizione di permanenza) implica la proprietà P
- Tipicamente (2) è il passo più critico
- P_i a volte ovvie, a volte molto complesse (or, if-then-else, ...)
⇒ problema **indecidibile** in generale
- Talvolta necessarie variabili ausiliarie addizionali
- Talvolta si adottano convenzioni per gestire il caso $i = 0$:
(la somma di 0 elementi è 0, il prodotto di 0 elementi è 1, ...)

Esempio: fattoriale

```
i = 1;  
fact = 1;  
while (i<=n) {  
    fact *= i;  
    i++;  
}
```

- **Proprietà P :** dopo il ciclo, fact vale il prodotto dei primi n numeri
- **Invariante P_i :** fact vale il prodotto dei primi i numeri
 - ✓(1) prima del ciclo, fact vale il prodotto dei primi 0 numeri (cioè 1)
 - ✓(2) prima dell'i-esimo ciclo fact vale il prodotto dei primi $i-1$ numeri
 ⇒ dopo l'i-esimo ciclo fact vale il prodotto dei primi i numeri
 - ✓(3) Alla fine dell'ultimo ciclo (n -esimo), P_n (più la negazione della condizione di permanenza del ciclo) implica la proprietà P

Nota

“dopo l'i-esimo ciclo” i è incrementato di 1. (Ex: dopo il 3^o ciclo, $i = 4$).

Esempio: divisibilità per 2

```
ndiv2=0; tmp=num; // "tmp" ausiliaria
while ( tmp%2 == 0 ) {
    ndiv2++;
    tmp/=2;
}
```

- **Proprietà P :** dopo il ciclo, $\text{tmp} \% 2 \neq 0$ e $\text{tmp} * (2^{\text{ndiv2}}) == \text{num}$
- **Invariante P_i :** $\text{tmp} * (2^{\text{ndiv2}}) == \text{num}$
 - ✓(1) prima del ciclo, $\text{tmp} * (2^0) == \text{num}$
 - ✓(2) prima dell' i -esimo ciclo tmp è divisibile per due e $\text{tmp} * (2^{\text{ndiv2}}) == \text{num}$
 \Rightarrow dopo l' i -esimo ciclo $\text{tmp} * (2^{\text{ndiv2}}) == \text{num}$
(infatti $\text{tmp}/2 * (2^{\text{ndiv2}+1}) == \text{num}$)
 - ✓(3) Alla fine dell'ultimo ciclo (n -esimo), P_n (più la negazione della condizione del ciclo) implica la proprietà P :
 $\text{tmp} * (2^{\text{ndiv2}}) == \text{num}$ e $\text{tmp} \% 2 \neq 0$

Esercizi Proposti

Esercizi sui cicli:

{ LOOPS/00ESERCIZI_PROPOSTI.txt }

Istruzioni di salto (**break**, **continue**, **goto**):
come non si deve programmare in C/C++ !!!

L'Istruzione di Salto **break**

L'istruzione **break** termina direttamente tutto il ciclo

```
while ( . . . ) {  
    . . .  
    break;          // --+  
    . . .          //  |  
}                  //  |  
                  // <-----+  
                  //           |
```

- Da evitare! \Rightarrow si può sempre fare modificando la condizione
- semplice break (while):
{ LOOPS/break_while.cc }
- come evitare un break (while):
{ LOOPS/nobreak_while.cc }

L'Istruzione **return** in un loop (salto implicito)

L'istruzione **return** termina direttamente il ciclo (e l'intera funzione)

```
int main () {  
...  
while (...) {  
...  
    return 0;    // --+  
    ...          //  |  
}  
}          // <-----+  
           // |
```

- **Da evitare!** \Rightarrow si può sempre fare modificando la condizione
- **semplice return (while):**
`{ LOOPS/return_while.cc }`
- **come evitare un return (while):**
`{ LOOPS/noreturn_while.cc }`

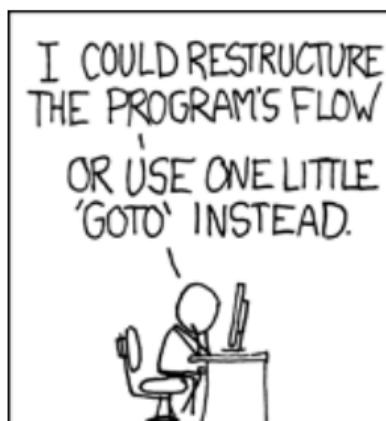
L'Istruzione di Salto **continue**

L'istruzione **continue** termina il ciclo attualmente in esecuzione e passa al successivo

```
while (....) {  
    ...  
    continue;    // --+  
    ...          //   |  
    // <-----+  
}
```

- nel caso di ciclo **for** viene saltata l'istruzione di aggiornamento
- **Da evitare!** \Rightarrow si può sempre fare lo stesso con un “if”
- semplice **continue (while):**
`{ LOOPS/continue.cc }`
- come evitare **continue (while):**
`{ LOOPS/nocontinue.cc }`

L'Istruzione di Salto goto II



© xkcd www.xkcd.com



Nota di servizio:

Nella soluzione di un testo di esame, **NON è ammesso** l'uso di **break**, **continue**, o **goto** (con l'importante eccezione dell'uso di **break** all'interno del costrutto **switch**), o di **return** all'interno di loop, pena l'annullamento dell'esercizio stesso.

Corso “Programmazione 1”

Capitolo 04: Riferimenti e Puntatori

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 7 ottobre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

I Tipi Derivati

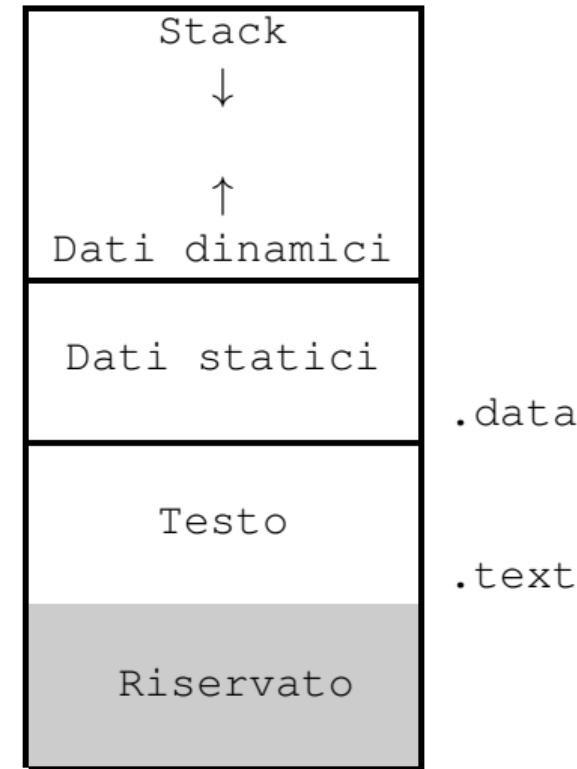
- Dai tipi fondamentali, attraverso vari meccanismi, si possono derivare tipi più complessi
- I principali costrutti per costruire tipi derivati sono:
 - i riferimenti
 - i puntatori
 - gli array
 - le strutture
 - le unioni
 - le classi

Struttura della memoria di un programma

SP → 0000 003f ffff ffff0₁₆

0000 0000 1000 0000₁₆

PC → 0000 0000 0040 0000₁₆



Il Tipo “Riferimento a”

- I meccanismo dei riferimenti (reference) consente di **dare nomi multipli** a una variabile (o a un'espressione dotata di indirizzo)
 - Un riferimento è un **sinonimo** dell'espressione a cui fa riferimento
 ⇒ modificando l'una, si modifica anche l'altra (“aliasing”)
 - Un riferimento è un'**espressione dotata di indirizzo**
- Sintassi: `tipo & id = exp;`
dove `exp` è un'**espressione dotata di indirizzo**

Esempio:

```
int x=1;
int &y=x; // y e' di tipo reference, e' sinonimo di x
y = 6;    // viene modificato anche x!
```

Vincoli sull'uso dei Riferimenti

- Nelle dichiarazioni di reference, l'inizializzazione è obbligatoria!

```
int &y; // errore!
```

- Non è possibile ridefinire una variabile di tipo riferimento precedentemente definita:

```
double x1, x2;
```

```
double &y=x1; // ok
```

```
double &y=x2; // errore! gia' definita!
```

- Non è (più) possibile definire un riferimento a

- un'espressione non dotata di indirizzo,

- o a un'espressione dotata di indirizzo ma di tipo diverso.

```
float &y=10.2; // Errore
```

```
double d=3.1; // Ok
```

```
int &z=d; // Errore
```

- Esempio di uso di references:

```
{ RIF_PUNT/reference.cc }
```

L'Operatore address-of “&”

- L'operatore & (“address-of”) ritorna l'indirizzo (l-value) dell'espressione a cui è applicato
- Può essere **applicato solo** a espressioni dotate di indirizzo!
- È **differente** dall'uso di “&” nella definizione di riferimenti!

Esempio

```
int l = 10;
cout << &l << endl;      // stampa l'indirizzo di l
cout << &(l*5) << endl; // errore!
int n = 10;
int& r = n; // r e' alias di n, ``punta'' stessa area di memoria
cout << "&n=" << &n << ", "<< "&r=" << &r << endl;
```

- Esempio di uso di address-of:
 { RIF_PUNT/address_1.cc }
- Riferimenti e address-of:
 { RIF_PUNT/rifVsAddressof.cc }

Il Tipo “Puntatore a...”

- Un **puntatore** contiene **l'indirizzo** di un altro oggetto
 - L'r-value di un puntatore è un indirizzo
- Definizione di un puntatore:
 - Sintassi: `tipo *id_or_init`
 - Esempio: `int *px; //px puntatore a un intero`
 - È sempre necessario indicare il **tipo** di oggetto a cui punta
- Un puntatore a tipo T può contenere solo indirizzi di oggetti di tipo T
- Ad una variabile puntatore viene associata una spazio di memoria atto a contenere un indirizzo di memoria,
 - ...ma **non viene riservato spazio di memoria per l'oggetto puntato!**
- Lo spazio allocato a una variabile di tipo puntatore è sempre uguale, indipendentemente dal tipo dell'oggetto puntato

L'Operatore di Dereference “*”

- Per accedere all'oggetto puntato da una variabile puntatore occorre applicare l'**operatore di dereference ***
- Se `px` punta a `x`, `*px` è un **sinonimo temporaneo** di `x`
 \Rightarrow modificando `*px` modifco `x`, e vice versa
- `*px` è un'**espressione dotata di indirizzo**

Esempio

```
int x=1; // x variabile tipo int
int *px; // px variabile puntatore a tipo int
px=&x; // accede alla variabile puntatore
*px=x+1; // accede alla cella di memoria puntata
           // dalla variabile puntatore
```

L'esempio di cui sopra:

{ RIF_PUNT/pointer.cc }

Assegnazioni tra Puntatori

- Assegnando a un puntatore `q` il valore di un altro puntatore `p`, `q` punterà allo stesso oggetto puntato da `p`
 - *`p`, *`q` e l'oggetto puntato da loro sono temporaneamente sinonimi

```
int i, j;
int *p, *q;
p = &i;      // p=indirizzo di i, *p sinonimo di i
q = &j;      // q=indirizzo di j, *q sinonimo di j
*q = *p;    // equivale a j=i
q = p;      // equivale a q=indirizzo di i
```

- L'esempio di cui sopra espanso:
{ RIF_PUNT/pointer1.cc }
- L'esempio di cui sopra espanso (2):
{ RIF_PUNT/pointer2.cc }

Esempi su puntatori e riferimenti

- Esempio: riferimento ad un oggetto puntato da un puntatore:
il riferimento “segue” il puntatore?:
{ RIF_PUNT/rif_deref.cc }

Puntatori a **void** (cenni)

- In alcuni casi è utile avere una variabile puntatore che possa puntare ad entità di tipo diverso;
- Tale variabile viene dichiarata di tipo “puntatore a **void**” cioè a tipo non specificato

```
int i; int *pi=&i;
char c; char *pc=&c;
void *tp;
tp = pi;           // punta a int
*(int*)tp=3;
tp = pc;           // punta a char
*(char*)tp='C';
```

Esempio di cui sopra:

{ RIF_PUNT/punt_a_void.cc }

Puntatori a costante (cenni)

- Definizione
 - Sintassi: **const** tipo *id_or_init;
 - Esempio: **const int** *pc1 = &c1;
- Intuizione: non permettono di modificare l'oggetto puntato tramite dereference del puntatore stesso
- Nota: non rendono l'oggetto puntato una costante

```
const int c1 = 3; int c2 = 5;
const int *pc1 = &c1; // ok
const int *pc2 = &c2; // ok
pc2 = pc1; // ok
pc1 = &c2; // ok
*pc1 = 2; // errore
c2 = 2; // ok
```

Esempio di cui sopra:

```
{ RIF_PUNT/punt_a_cost.cc }
```

Costanti puntatore (cenni)

- Definizione
 - Sintassi: tipo ***const** id=exp;
 - Esempio: **int *const** pa = &a;
- Intuizione: non permettono di puntare ad un altro oggetto
- L'oggetto puntato può essere modificato tramite dereference del puntatore stesso

```
int a, b;  
int *const pa = &a;  
*pa = 3; // ok  
pa = &b // errore: pa e' costante
```

Esempio di cui sopra:

{ RIF_PUNT/const_punt.cc }

Costanti puntatore a costante (cenni)

- Definizione
 - Sintassi: **const** tipo ***const** id=exp;
 - Esempio: **const int** ***const** a = &c;
- Intuizione: non permettono di puntare ad un altro oggetto
- L'oggetto puntato **non** può essere modificato tramite dereference del puntatore stesso

```
const int b = 2;
const int c = 3;
const int *const a = &c;
a = &b; // errore
*a= 2; // errore
c = 5; // errore
```

Esempio di cui sopra:

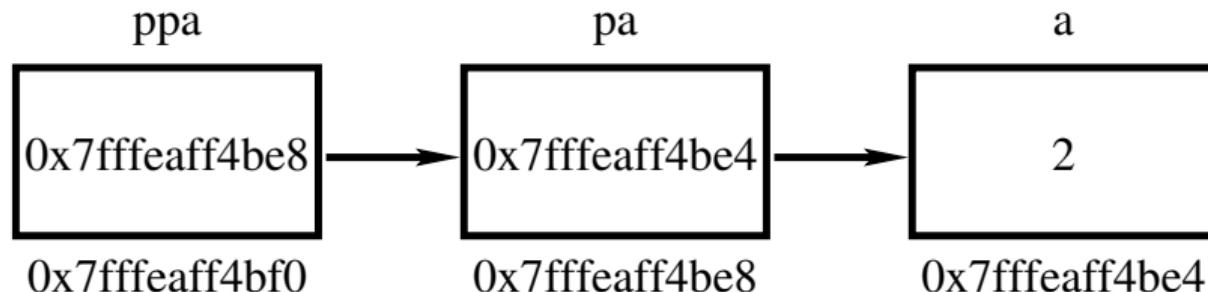
{ RIF_PUNT/const_punt_const.cc }

Puntatori a puntatori

- Una variabile puntatore è una variabile con un tipo (similmente a qualunque altra variabile), per cui è possibile definire puntatori a tali variabili.
- Il suo indirizzo è un puntatore ad un puntatore.
- Esempi:
 - **int** **p; //puntatore a puntatore ad intero
 - **char** **c //puntatore a puntatore a carattere

Puntatori a puntatori (II)

```
int main () {
    int a, *pa, **ppa;
    a = 2; pa = &a; ppa = &pa;
    cout << "Ind. di a = " << &a;
    cout << "valore di a = " << a << endl;
    cout << "Ind. di pa = " << &pa;
    cout << "valore di pa = " << pa << endl;
    cout << "Ind. di ppa = " << &ppa;
    cout << "valore di ppa = " << ppa << endl;
}
```



Utilizzo pratico di puntatori

Dear Santa,
How are you? I'm good.
Here is what I want for
Christmas.

https://www.amazon.com/gp/product/B0032HF60M/ref=s9_hps_bw_g21-ir03?pf_rd_m=ATVPDKIKXODER&pf_rd_s=center-3&pf_rd_t=1XW442FH1K03Y73MWQNM&pf_rd_r=101&pf_rd_p=1328901542&pf_rd_i=6579

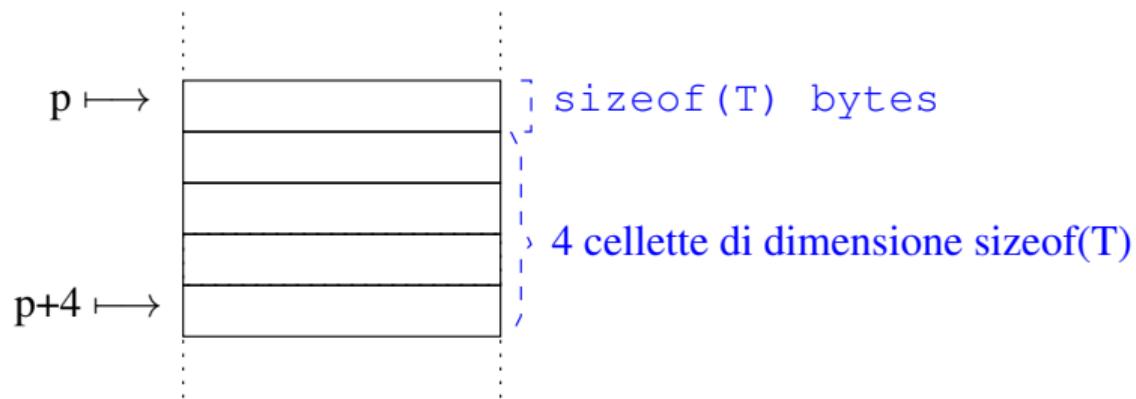
Aritmetica di Puntatori ed Indirizzi

Gli indirizzi e i puntatori hanno un'aritmetica:

se p è di tipo T^* e i è un intero, allora:

- $p+i$ è di tipo T^* ed è l'indirizzo di un oggetto di tipo T che si trova in memoria dopo i posizioni di dimensione **sizeof** (T)
- analogo discorso vale per $p++$, $++p$, $p--$, $--p$, $p+=i$, ecc.

$\Rightarrow i$ viene implicitamente moltiplicato per **sizeof** (T)

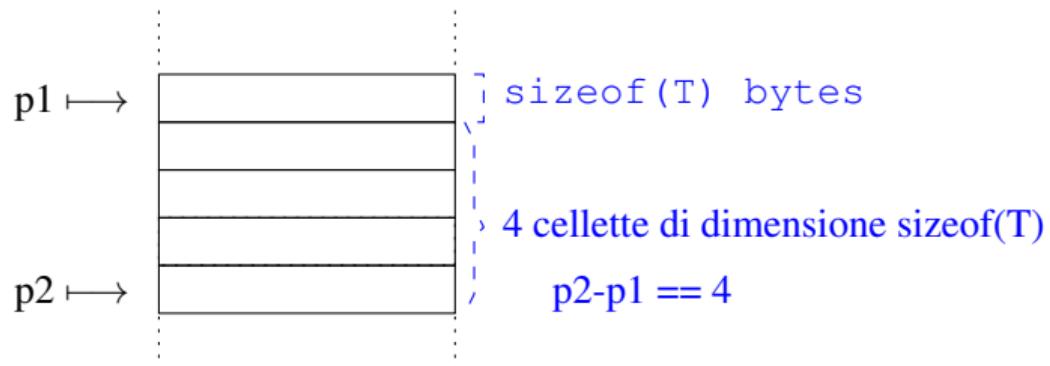


Aritmetica di Puntatori ed Indirizzi II

se $p1$, $p2$ sono di tipo T^* , allora:

- $p2 - p1$ è un intero ed è il numero di posizioni di dimensione **sizeof** (T) per cui $p1$ precede $p2$ (negativo se $p2$ precede $p1$)
- si possono applicare operatori di confronto $p1 < p2$, $p1 \geq p2$, ecc.

⇒ $p2 - p1$ viene implicitamente diviso per **sizeof** (T)



Esempio di operazioni aritmetiche su puntatori:

{ RIF_PUNT/aritmetica_punt.cc }

Priorità tra dereference e operatori aritmetici

Nota

Attenzione alle priorità tra l'operatore dereference “*” e gli operatori aritmetici:

- $*pv+1$ è equivalente a $(*pv)+1$, non a $*(pv+1)$
- $*pv++$ è equivalente a $*(pv++)$, non a $(*pv)++$

⇒ è consigliabile usare le parentesi per non confondersi.

- Esempio di cui sopra:
{ RIF_PUNT/priorita.cc }

Corso “Programmazione 1”

Capitolo 05: Le Funzioni

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 12 ottobre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Concetto di funzione

In un programma è sempre opportuno e conveniente strutturare il codice raggruppandone delle sue parti in **sotto-programmi autonomi**, detti **funzioni**, che vengono eseguite in ogni punto in cui è richiesto.

- L'organizzazione in funzioni ha moltissimi vantaggi:
 - Miglior strutturazione e organizzazione del codice
 - Maggior leggibilità del codice
 - Maggior mantenibilità del codice
 - Riutilizzo di sotto-parti di uno stesso programma più volte
 - Condivisioni di sotto-programmi tra programmi distinti
 - Utilizzo di codice fatto da altri/librerie
 - Sviluppo di un programma in parallelo, tra più autori
 - ...
- In un programma C++ è possibile **definire** e **chiamare** funzioni
- È possibile anche chiamare funzioni definite altrove
 - funzioni definite in altri file
 - **funzioni di libreria**

Funzioni di libreria

- Una funzione è un sotto-programma che può essere utilizzato ripetutamente in un programma, o in programmi diversi
- Una **libreria** è un insieme di **funzioni precompilate**.
- Alcune librerie C++ sono disponibili in tutte le implementazioni e con le stesse funzioni (ad es. `cmath`)
- Una libreria è formata da una coppia di file:
 - un file di intestazione (header) contenente le dichiarazioni dei sotto-programmi stessi
 - un file oggetto contenente le funzioni compilate
- Per utilizzare in un programma le funzioni in una libreria bisogna:
 - includere il file di intestazione della libreria con la direttiva **#include <nomelibreria>**
 - in alcuni casi, indicare al linker il file contenente le funzioni compilate della libreria
 - introdurre nel programma chiamate alle funzioni della libreria

Alcuni esempi di funzioni di libreria I

- Libreria `<cmath>`: funzioni matematiche (da `double` a `double`)
 - `fabs (x)`: valore assoluto di tipo float
 - `sqrt (x)`: radice quadrata di x
 - `pow (x, y)`: eleva x alla potenza di y
 - `exp (x)`: eleva e alla potenza di x
 - `log (x)`: logaritmo naturale di x
 - `log10 (x)`: logaritmo in base 10 di x
 - `sin (x)` e `asin (x)`: seno e arcoseno trigonometrico
 - `cos (x)` e `acos (x)`: coseno e arcocoseno trigonometrico
 - `tan (x)` e `atan (x)`: tangente e arcotangente trig.
 - ...
- possono essere usate con tutti gli altri tipi numerici tramite conversione implicita o esplicita

Alcuni esempi di funzioni di libreria II

- Libreria `<cctype>`, funzioni di riconoscimento (da `char` a `bool`):

- `isalnum(c)`: carattere alfabetico o cifra decimale
- `isalpha(c)`: carattere alfabetico
- `iscntrl(c)`: carattere di controllo
- `isdigit(c)`: cifra decimale
- `isgraph(c)`: carattere grafico, diverso da spazio
- `islower(c)`: lettera minuscola
- `isprint(c)`: carattere stampabile, anche spazio
- `isspace(c)`: spazio, salto pagina, nuova riga o tab.
- `isupper(c)`: lettera maiuscola
- `isxdigit(c)`: cifra esadecimale
- ...

- Libreria `<cctype>`, funzioni di conversione (da `char` a `char`):

- `tolower(c)`: se `c` è una lettera maiuscola restituisce la corrispondente lettera minuscola, altrimenti restituisce `c`
- `toupper(c)`: come sopra ma in maiuscolo
- ...

Esempio di uso di funzioni di libreria

```
#include <cmath>
(...)
  for (float i=1.0; i<=MAX; i+=1.0)
    cout << log(i)/log(2.0) << endl; // log2(i)
(...)
// dallo header della libreria cmath:
double log(double x);
```

- alla chiamata `log(i)`:
 - Il programma **trasferisce il controllo** dal codice di `main` al codice di `log` in `cmath`, e lo riprende al termine della funzione
 - il valore di `i` viene valutato e passato in input alla funzione `log`
 - `log(i)` viene **valutata** al valore restituito dalla computazione della funzione `log` con il valore `i` in input

Esempio di cui sopra (esteso):

{ FUNCTIONS/tavola_logaritmi.cc }

Funzioni: Dichiarazione, Definizione e Chiamata

- **Definizione:**

- Sintassi: tipo id(tipo1 id1, ..., tipoN idN) {...}
- Esempio: **double** pow(**double** x, **double** y) {...}
- id1, ..., idN sono i **parametri formali** (sempre presenti) della funzione

- **Dichiarazione:**

- Sintassi: tipo id(tipo1 [id1], ..., tipoN [idN]);
- Esempio: **double** pow(**double**, **double** e);
- Serve per “richiamare” una definizione fatta altrove, e consentirne l’uso!
- Nota: id1, ..., idN sono opzionali!

- **Chiamata:**

- Sintassi: id (exp1, ..., expN)
- Esempio: x = pow(2.0*y, 3.0);
- exp1, ..., expN sono i **parametri attuali** della chiamata

Nota

I parametri attuali exp1, ..., expN della chiamata devono essere compatibili per numero, ordine e rispettivamente per tipo ai corrispondenti parametri formali!

L'istruzione **return**

- Il corpo di una funzione può contenere una o più istruzioni **return**
 - Sintassi: **return** expression;
 - Esempio: **return** 3*x;
- expression deve essere **compatibile** con il tipo restituito dalla funzione
- L'esecuzione dell'istruzione **return**:
 - fa terminare la funzione
 - fa sì che il valore della chiamata alla funzione sia il valore dell'espressione expression (con conversione implicita se di tipo diverso)

Nota

È buona prassi che una funzione contenga un'unica istruzione **return**!

Esempio: la funzione mcd

Esempio di funzione:

{ FUNCTIONS/mcd.cc }

La chiamata `mcd(n1, n2)` viene eseguita nel modo seguente:

- (i) vengono calcolati i valori dei parametri attuali `n1` e `n2` (l'ordine non è specificato)
- (ii) i valori vengono copiati, nell'ordine, nei parametri formali `a` e `b` (chiamata **per valore**)
- (iii) viene eseguita la funzione e modificati i valori di `a` e `b` e della variabile locale `resto` (`n1` e `n2` rimangono con il loro valore originale)
- (iv) la funzione `mcd` restituisce al programma chiamante il valore dell'espressione che appare nell'istruzione `return`

```
int mcd(int a, int b) {  
    int resto;  
    while(b!=0) {  
        resto = a%b;  
        a = b;  
        b = resto;  
    }  
    return a;  
}
```

Esempi

- Chiamate miste a funzioni definite e di libreria:

{ FUNCTIONS/mylog10.cc }

- funzione fattoriale:

{ FUNCTIONS/fact.cc }

- ...con dichiarazione (header):

{ FUNCTIONS/fact1.cc }

- ... con identificatore "fattoriale" locale e globale:

{ FUNCTIONS/fact2.cc }

- ... con parametro formale stesso nome di parametro attuale:

{ FUNCTIONS/fact3.cc }

- decomposto in più file:

{ FUNCTIONS/fact4*.{cc|h} }

- Esempio di funzione Booleana:

{ FUNCTIONS/isprime.cc }

Procedure (funzioni void)

In C++ c'è la possibilità di definire **procedure**, cioè funzioni che non ritornano esplicitamente valori (ovvero funzioni il cui valore di ritorno è di tipo **void**)

```
void pippo (int x) // definizione di funzione void
{ ... }
( ... )
pippo(n*2); // chiamata di funzione void
```

Nelle funzioni void, l'espressione **return** può mancare, oppure apparire senza essere seguita da espressioni (termina la procedura).

- Es. di funzione void: stampa di una data:
 { FUNCTIONS/printdate.cc }
- Es. di funzione void: stampa di tutti i caratteri:
 { FUNCTIONS/printchartype.cc }
- Esempio di funzione senza argomenti:
 { FUNCTIONS/tiradadi.cc }

Return multipli in una funzione

- In una funzione è buona prassi evitare l'uso di **return** multipli (in particolare se usati come impliciti if-then-else)

```
int f (...) {  
    ...  
    return exp1;  
    ...  
    return expN;  
}
```

⇒

```
int f (...) {  
    int res;  
    ...  
    res = exp1;  
    ...  
    res = expN;  
    return res;  
}
```

```
if (...) {  
    return exp1; }  
... // altrimenti...
```

⇒

```
if (...) {  
    res = exp1; }  
else { (... ) }
```

- Es.: funzione **isprime** con return unico:
{ FUNCTIONS/isprime_onereturn.cc }

L'Istruzione **return** in un loop (salto implicito)

L'istruzione **return** termina direttamente il ciclo (e l'intera funzione)

- equivalente ad un **break**;
- **Da evitare!** \Rightarrow si può sempre fare modificando la condizione

```
int f () {  
    ...  
    while (...) {  
        ...  
        return ...; // --+  
        ...          //  |  
    }            |  
}            // <-----+  
           |
```

Corso “Programmazione 1”

Capitolo 05: Le Funzioni

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 14 ottobre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Conversione implicita dei parametri attuali

La chiamata (per valore) concettualmente analoga all'inizializzazione dei parametri formali

```
int f (int x, ...) { ... }  
...  
... f(expr) ...
```



```
...  
int x = expr;  
...
```

Nota importante

Nella chiamata a funzione in cui i parametri attuali siano di tipo **diverso** ma **compatibile** con quello dei rispettivi parametri formali, viene fatta una conversione implicita di tipo (con tutte le possibili problematiche ad essa associate)

- **Regole analoghe a quelle dell'inizializzazione/assegnazione**
- **Es:**

```
pow(2, 4)           // conv. implicita da int a double  
mcd(54.0, 30.5) // conv. implicita da double a int
```

Ordine di valutazione di un'espressione II

- In C++ non è specificato l'ordine di valutazione degli operandi di ogni operatore, in particolare:
 - l'ordine di valutazione di sottoespressioni in un'espressione
 - l'ordine di valutazione degli argomenti di una funzione
- Es: nel valutare `f(expr1, expr2)`, non è specificato se `expr1` venga valutata prima di `expr2` o viceversa
- Problematico quando sotto-espressioni contengono operatori con “side-effects” come gli operatori di incremento.
Es: `x=pow(++i, ++i); //undefined behavior`
⇒ **evitare l'uso di operatori con side-effects in chiamate a funzioni**

Per approfondimenti si veda ad esempio

http://en.cppreference.com/w/cpp/language/eval_order

Parametri e variabili locali

- Un **parametro formale** è una variabile cui viene associato il corrispondente parametro attuale ad ogni chiamata della funzione
 - [se non diversamente specificato] il **valore** del parametro attuale viene copiato nel parametro formale
(passaggio di parametri **per valore**)
- Le variabili dichiarate all'interno di una funzione sono dette **locali**
 - appartengono solo alla funzione in cui sono dichiarate
 - sono visibili solo all'interno della funzione
- Le variabili dichiarate all'esterno di funzioni sono dette **globali**
 - sono visibili all'interno di ogni funzione (se non mascherate da variabili locali con lo stesso nome)
- **Esempio sull'ambito di parametri e variabili locali:**
{ FUNCTIONS/scope.cc }

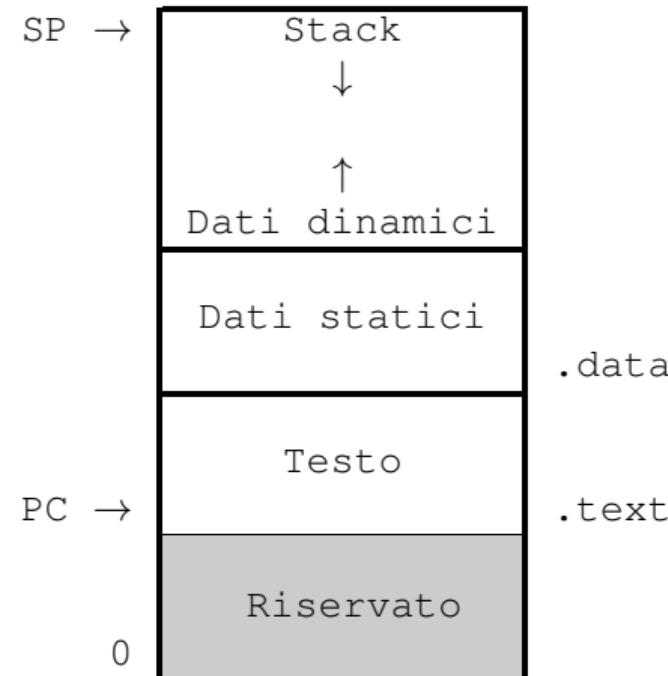
Durata di parametri e variabili locali

- I parametri formali e le variabili locali “esistono” (hanno uno spazio di memoria a loro riservato) **solo durante l'esecuzione della rispettiva funzione**
 - (i) All'atto della chiamata viene riservata loro un'area di memoria
 - (ii) Vengono utilizzati per le dovute elaborazioni
 - (iii) Al termine della funzione la memoria da essi occupata viene resa disponibile

Modello di gestione della memoria per un programma

Area di memoria allocata ad un'esecuzione di un programma:

- **Area programmi:** destinata a contenere le **istruzioni** (in linguaggio macchina)
- **Area dati statici:** destinata a contenere **variabili globali** o **allocate staticamente** e le **costanti** del programma
- **Area heap:** destinata a contenere le **variabili dinamiche** (di dimensioni non prevedibili a tempo di compilazione)
- **Area stack:** destinata a contenere le **variabili locali** e i **parametri formali** delle funzioni



Modello a Stack

- I parametri formali e le variabili locali a una funzione sono memorizzate in un'area riservata della memoria, detta **stack**
- Modello di memoria concettualmente analogo a quello di una “pila” (“stack”):
 - quando una funzione viene chiamata, il blocco di memoria necessario per contenere i suoi parametri formali e variabili locali viene allocato “sopra” quello della funzione che la chiama
 - quando la funzione termina, tale blocco viene reso di nuovo disponibile
 - politica di gestione “last in first out” (LIFO)

Esempi

- esempio di funzioni che chiamano funzioni [D]:

{ FUNCTIONS/comb.cc }

- ...con dichiarazioni (headers):

{ FUNCTIONS/comb2.cc }

- ... tracciando gli indirizzi delle variabili e parametri:

{ FUNCTIONS/comb2_track.cc }

- chiamate annidate di funzioni [D]:

{ FUNCTIONS/mymax.cc }

- ... tracciando gli indirizzi (stack):

{ FUNCTIONS/mymax_track.cc }

Passaggio di parametri

In C++ esistono tre modalità passaggio di parametri a una funzione:

- per valore
- per riferimento
- per puntatore

(Spesso le ultime due sono confuse in letteratura, perché hanno finalità simili.)

Il passaggio di parametri per valore

- Definizione di parametri formali analoga a definizione di variabili
 - Sintassi lista dei parametri: (tipo identificatore, ...)
 - Es: `int fact(int n) {...}`
- Simile a definire una nuova variabile locale e assegnarle il valore dell'espressione del parametro attuale.
 - Es: `fact (3*x);` //simile a: `int n = 3*x;`
- Il parametro formale acquisisce **il valore** del parametro attuale
 - il parametro attuale può essere un'espressione senza indirizzo
 - può essere di tipo diverso compatibile \Rightarrow conversione implicita
- L'informazione **viene (temporaneamente) duplicata**
 \Rightarrow possibile spreco di tempo CPU e memoria
- Se modifco il parametro formale, il parametro attuale non viene modificato
 \Rightarrow passaggio di informazione **solo dalla chiamante alla chiamata**
- **Tutti gli esempi di funzioni visti finora usano passaggio per valore:**
`{ FUNCTIONS/... }`

Il passaggio di parametri per riferimento

- Definizione di parametri formali simile a definizione di riferimenti
 - Sintassi lista dei parametri: (tipo & identificatore, ...)
 - Es: `int swap(int & n, int & m) { ... }`
- Simile a definire un riferimento “locale” ad un’espressione dotata di indirizzo
 - Es: `swap(x, y);` //simile a: `int & n=x; int & m=y`
- Il parametro è un **riferimento** al parametro attuale
 - il parametro attuale deve essere un’**espressione dotata di indirizzo**
 - deve essere dello stesso tipo
- L’informazione **non viene duplicata**
 - ⇒ evito possibile spreco di tempo CPU e memoria
- Se modifco il parametro formale, modifco il parametro attuale
 - ⇒ passaggio di informazione **anche dalla chiamata alla chiamante**

Esempi

- passaggio per valore, errato:
 { FUNCTIONS/scambia_err.cc }
- passaggio per riferimento, corretto [D]:
 { FUNCTIONS/scambia.cc }
- passaggio per riferimento non ammesso, tipo diverso:
 { FUNCTIONS/riferimento_err.cc }
- problemi ad usare il riferimento quando non dovuto:
 { FUNCTIONS/mcd_err.cc }
- restituzione di due valori :
 { FUNCTIONS/rectpolar.cc }
- parametro come input e output di una funzione:
 { FUNCTIONS/iva.cc }

Passaggio di parametri per riferimento costante

- È possibile definire passaggi per riferimento **in sola lettura** (**passaggio per riferimento costante**)
 - Sintassi: (**const** tipo & identificatore, ...)
 - Es: **int** fact (**const int** & n, ...) {...}
- Riferimento: l'informazione non viene duplicata
 - ⇒ evito possibile spreco di tempo CPU e memoria
- **Non permette di modificare n!**
 - Es: n = 5; //ERRORE!
 - ⇒ passaggio di informazione **solo dalla chiamante alla chiamata**
 - ⇒ solo un input alla funzione
- Usato per passare **in input** alla funzione oggetti “grossi”
 - efficiente (no spreco di CPU e memoria)
 - evita errori
 - permette di individuare facilmente gli input della funzione
- **uso di riferimenti costanti:**
{ FUNCTIONS/usa_const.cc }

Esempi (2)

- esempi per riferimento:

```
{ FUNCTIONS/cipeciop.cc }
```

- ...:

```
{ FUNCTIONS/pippo.cc }
```

- ...:

```
{ FUNCTIONS/paperino.cc }
```

- ...:

```
{ FUNCTIONS/topolino.cc }
```

Con i riferimenti è facile fare confusione!

Il passaggio di parametri per puntatore

- Definizione di parametri formali: puntatori passati per valore
 - Sintassi lista dei parametri: (tipo * identificatore, ...)
 - Es: `int swap(int * pn, int * pm) { ... }`
 - N.B.: nella chiamata, si passa **l'indirizzo dell'oggetto passato**
 - Simile a definire un puntatore “locale” ad un’espressione dotata di indirizzo
 - Es: `swap(&x, &y);` //simile a: `int *pn=&x; int *pm=&y`
 - Il parametro è un **puntatore** al(l’oggetto il cui indirizzo è dato dal) parametro attuale
 - che deve essere un’**espressione dotata di indirizzo**
 - che deve essere dello stesso tipo
 - L’informazione **non viene duplicata**
 - ⇒ evito possibile spreco di tempo CPU e memoria
 - Se modifco il parametro formale, modifco il parametro attuale
 - ⇒ passaggio di informazione **anche dalla chiamata alla chiamante**
- ⇒ **effetto simile al passaggio per riferimento** (vedi C)

Esempi

- come scambia.cc, con passaggio per puntatore:
 { FUNCTIONS/scambia_punt.cc }
- come iva.cc, con passaggio per puntatore:
 { FUNCTIONS/iva2.cc }
- come paperino.cc, con passaggio per puntatore:
 { FUNCTIONS/paperino2.cc }

Passaggio per valore vs. p. per riferimento/puntatore

- Vantaggi del passaggio per riferimento/puntatore:
 - Minore carico di calcolo e di memoria (soprattutto con parametri di grosse dimensioni)
 - Permette di restituire informazione da chiamata a chiamante
- Svantaggi del passaggio per riferimento/puntatore:
 - Rischio di confusione nel codice (non si sa dove cambiano i valori)
 - Aliasing (entità con più di un nome)
 - Parametro formale e attuale esattamente dello stesso tipo
 - Si possono passare solo espressioni dotate di indirizzo

Nota

- alcuni linguaggi (es C) non ammettono passaggio per riferimento (solo per puntatore)

Esercizi proposti

Vedere file ESERCIZI_PROPOSTI.txt

Funzioni che restituiscono un riferimento

- Restituisce un **riferimento** ad un'espressione (con indirizzo)
 - l'espressione deve riferirsi ad un oggetto del chiamante (es. un parametro formale passato per riferimento, un elemento di un array)
 - deve essere dello stesso tipo
- **La chiamata è un'espressione dotata di indirizzo!**

```
int& max(int& x, int& y) //restituisce un riferimento
{return (x > y ? x : y);}//x, y riferimenti a oggetti
                           //non locali
(...)

int m=44, n=22;
max(m, n) = 55;        //cambia il valore di m da 44 a 55
```

Esempio di cui sopra esteso:

{ FUNCTIONS/restituzione_riferimento.cc }

Sovrapposizione di parametri (overloading)

- In C++ è possibile **dare lo stesso nome a funzioni diverse**, purché con liste di parametri diverse, per numero e/o per tipo
- Il compilatore “riconosce” la giusta funzione per ogni chiamata.
- In caso di ambiguità, il compilatore produce un errore
- Conversioni implicite ammissibili, purché non causino ambiguità

```
int max(int, int) {...};  
int max(int, int, int) {...};  
double max(double, double) {...};  
(...)  
cout << max(99, 77) << " " << max(55, 66, 33) << " "  
    << max(3.4, 7.2) << endl;  
cout << max(3, 3.1) << endl; // errore: AMBIGUA
```

Esempio di cui sopra esteso:

{ FUNCTIONS/overloading.cc }

Funzioni con argomenti di default (cenni)

- In C++ è possibile fornire parametri opzionali, con valori di default
 - permette chiamate con liste di parametri attuali ridotte
 - i parametri opzionali devono essere gli ultimi della lista
 - il match viene effettuato da sinistra a destra

```
double p(double, double, double =0, double =0, double =0);
```

```
cout << p(x, 7) << endl;
cout << p(x, 7, 6) << endl;
cout << p(x, 7, 6, 5) << endl;
cout << p(x, 7, 6, 5, 4) << endl;
```

```
double p(double x, double a0, double a1, double a2, double a3)
{ return a0 + (a1 + (a2 + a3*x)*x)*x; }
```

Esempio di cui sopra esteso:

{ FUNCTIONS/defaultvalues.cc }

Corso “Programmazione 1”

Capitolo 05: Le Funzioni

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 14 ottobre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Funzioni ricorsive

- In C++ una funzione può invocare se stessa (**funzione ricorsiva**)
- ... o due o più funzioni possono chiamarsi a vicenda (**funzioni mutualmente ricorsive**)
- Formulare alcuni problemi in maniera ricorsiva risulta naturale:
 - il fattoriale: $0! \stackrel{\text{def}}{=} 1; n! \stackrel{\text{def}}{=} n \cdot (n - 1)!$
 - pari/dispari: $\text{even}(n) \iff \text{odd}(n - 1); \text{odd}(n) \iff \text{even}(n - 1);$
 - espressioni: $\text{somma} \stackrel{\text{def}}{=} \text{numero}; \text{somma} \stackrel{\text{def}}{=} (\text{somma} + \text{somma})$
- Due componenti:
 - una o più **condizioni di terminazione**
 - una o più **chiamate ricorsive**
- **Intrinseco rischio di produrre sequenze infinite**
 - Analoghe considerazioni rispetto ai cicli
- **Alcune “insidie” computazionali**
 - Es: funzione di Fibonacci: $f_0 \stackrel{\text{def}}{=} 1; f_1 \stackrel{\text{def}}{=} 1; f_n \stackrel{\text{def}}{=} f_{n-1} + f_{n-2}$

Ricorsione fortemente collegata al **principio di induzione matematico**.

Esempi

- fattoriale:

```
{ FUNZIONI_RICORSIVE/fact_nocomment.cc }
```

- ..., con chiamate tracciate:

```
{ FUNZIONI_RICORSIVE/fact.cc }
```

- ..., errore (loop infinito) :

```
{ FUNZIONI_RICORSIVE/fact_infloop.cc }
```

- ..., stack tracciato :

```
{ FUNZIONI_RICORSIVE/fact_stack.cc }
```

- funzioni mutualmente ricorsive:

```
{ FUNZIONI_RICORSIVE/pariDispari.cc }
```

- Fibonacci:

```
{ FUNZIONI_RICORSIVE/fibonacci_nocomment.cc }
```

- ..., con chiamate tracciate:

```
{ FUNZIONI_RICORSIVE/fibonacci.cc }
```

- versione iterativa:

```
{ FUNZIONI_RICORSIVE/fibonacci_iterativa.cc }
```

Nota sulla ricorsione

La realizzazione ricorsiva di una funzione può richiedere due funzioni:

- una funzione ausiliaria ricorsiva, con un **parametro di ricorsione** aggiuntivo (simile a contatore in loop)
 - una funzione principale (**wrapper**) che chiama la funzione ricorsiva con un valore base del parametro di ricorsione
 - situazione molto frequente nell'uso di array (prossimo capitolo)
-
- Esempio di funz. ricorsiva che necessita wrapper:
 { FUNZIONI_RICORSIVE/stampanumeri.cc }
 - ... variante 1:
 { FUNZIONI_RICORSIVE/stampanumeri1.cc }
 - ... variante 2:
 { FUNZIONI_RICORSIVE/stampanumeri2.cc }
 - analoga variante del fattoriale, con wrapper:
 { FUNZIONI_RICORSIVE/fact_rec1.cc }

Ricorsione vs. Iterazione

- Ricorsione spesso più naturale, semplice ed elegante
- Efficienza della ricorsione critica:
 - Attenzione a chiamate identiche in rami diversi! (es. Fibonacci)
 ⇒ rischio esplosione combinatoria
 - Dimensione dello stack dipende dalla profondità di ricorsione
 ⇒ notevole overhead e spreco di memoria
 ⇒ quando possibile, tipicamente iterazione più efficiente
 ⇒ passando oggetti “grossi”, è indispensabile usare passaggio per riferimento o puntatore
- Molte funzioni ricorsive possono essere riscritte in forma iterativa:
 - tail recursion: una chiamata ricorsiva, operata come ultimo passo
 - Es: somma, pari/dispari, ...
 - in generale, quando non comporta una “biforcazione”
 - Es: fattoriale, Fibonacci, ...
 - g++ -O2 effettua una conversione da tail-recursive in iterative

Da ricorsione in coda a iterazione (caso void)

```
void F(int x, ...) {  
    if (CasoBase(x, ...))  
        IstrBase(...);  
    else {  
        Istr(...);  
        x=agg(x, ...);  
        F(x, ...);  
    } }  
}
```

↔

```
void F(int x, ...) {  
    while (!CasoBase(x, ...)) {  
        Istr(...);  
        x=agg(x, ...);  
    }  
    IstrBase(...);  
}
```

- Esempio funzione void tail-recursive :
 { FUNZIONI_RICORSIVE/stampanumeri3.cc }
- ... corrispondente versione iterativa :
 { FUNZIONI_RICORSIVE/stampanumeri3_while.cc }

Da ricorsione in coda a iterazione (caso generale)

```
type F(int x, ...) {  
    if (CasoBase(x, ...))  
        res = IstrBase(...);  
    else {  
        Istr(...);  
        x=agg(x, ...);  
        res = F(x, ...);  
    }  
    return res;  
}
```

↔

```
type F(int x, ...) {  
    while (!CasoBase(x, ...)) {  
        Istr(...);  
        x=agg(x, ...);  
    }  
    res = IstrBase(...);  
    return res;  
}
```

- Esempio funzione tail-recursive:
 { FUNZIONI_RICORSIVE/sum.cc }
- ... corrispondente versione iterativa:
 { FUNZIONI_RICORSIVE/sum_while.cc }
- Compilazione di funzioni tail-recursive in iterative:
 { FUNZIONI_RICORSIVE/tailrecursive-comp.cc }

Ricorsione vs. Iterazione II

- ...
- Talvolta **non** è agevole riscrivere la ricorsione in forma iterativa
 - funzioni non-tail recursive, chiamate multiple
 - Es: manipolazione di espressioni

Esempio di gestione di espressioni:

{ FUNZIONI_RICORSIVE/espressione.cc }

- In generale, convertire una funzione ricorsiva in iterativa richiede l'uso di uno stack

Sogni ricorsivi e mutualmente ricorsivi



Esercizi proposti

Vedere file ESERCIZI_PROPOSTI.txt

Corso “Programmazione 1”

Capitolo 06: Gli Array

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 26 ottobre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

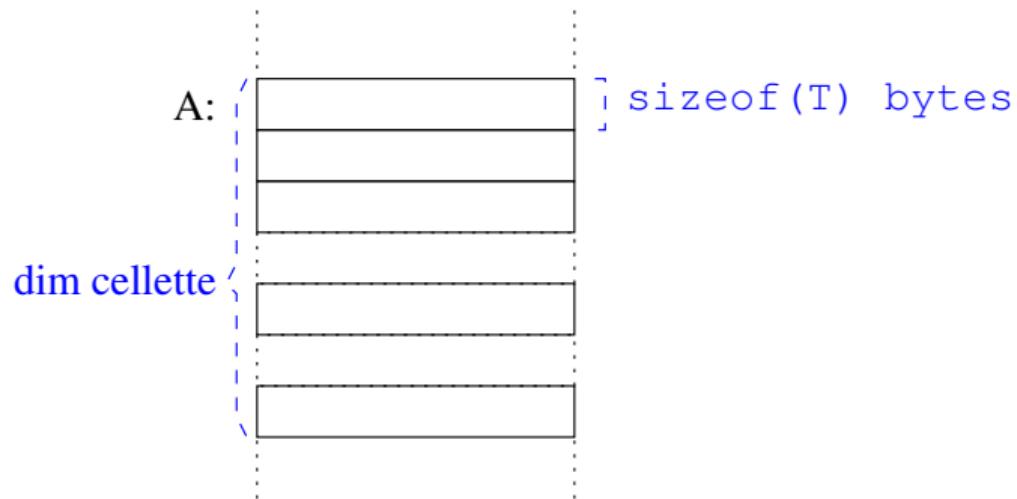
The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Outline

- 1 Definizione ed Utilizzo di Array
- 2 Array e Funzioni
- 3 Array Ordinati
- 4 Array Multi-Dimensionali
- 5 Array e Puntatori
- 6 Array, Puntatori e Funzioni

Tipi e Variabili Array

- **Array**: sequenza finita di elementi consecutivi dello stesso tipo.
- Il numero di elementi di un array (**dimensione**) è fissata a priori
- Per un array di tipo T e dimensione dim , il compilatore alloca dim cellette consecutive di **sizeof(T) bytes** (**allocazione statica**)
- Un array rappresenta l'indirizzo del primo elemento della sequenza



Definizione ed Inizializzazione di Array

- Sintassi definizione:

- tipo id[dim];
- tipo id[dim]={lista_valori};
- tipo id[]={lista_valori};

- Esempi:

```
double a[25]; //array di 25 double
const int c=2;
char b[2*c]={'a','e','i','o'}; // dimensione 4
char d[]={ 'a','e','i','o','u'}; // dimensione 5
```

- La **dimensione** dell'array deve essere valutabile **al momento della compilazione**:

- **esplicitamente**, tramite l'espressione costante `dim`
- **implicitamente**, tramite la dimensione della lista di inizializzazione

- Se mancano elementi nella lista di inizializzazione, il corrispondente valore viene inizializzato allo **zero del tipo T**

Operazioni non lecite sugli array

- Sugli array **non** sono definite operazioni **aritmetiche**, **di assegnamento**, **di input**
- Le operazioni **di confronto** e **di output** sono definite, ma danno risultati imprevedibili
⇒ per tutte queste operazioni è necessario scrivere funzioni ad-hoc

```
int a[4] = {1,2,3,4};  
int b[4] = {1,2,3,4};  
// a++;           // ERRORE IN COMPILAZIONE  
// a=b;           // ERRORE IN COMPILAZIONE  
// cin >> a;    // ERRORE IN COMPILAZIONE  
cout << (a==b) << endl; // COMPILA, MA DA' FALSE  
cout << (a<=b) << endl; // COMPILA, MA IMPREVEDIBILE  
cout << a << endl;      // COMPILA, MA IMPREVEDIBILE
```

esempio di cui sopra, espanso:
{ ARRAY/array.cc }

Operazioni sugli array: selezione con indice

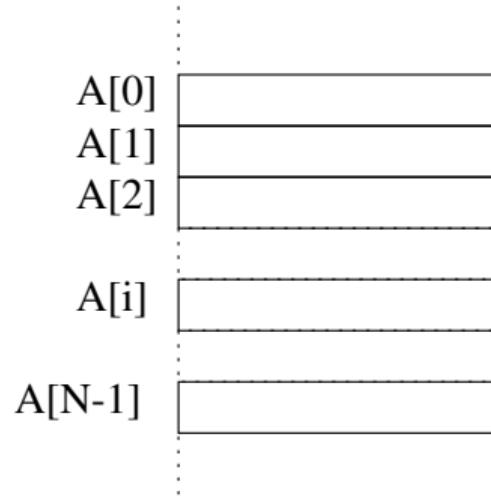
L'unica operazione definita sugli array è la **selezione con indice** (subscripting), ottenuta nella forma:

identifier[expression]

- identifier è il nome dell'array
- il valore di expression, è l'**indice** dell'elemento
 - è di tipo discreto (convertibile in intero)
- **è un'espressione variabile!!**
Es: v[100] diversissimo da v0, ..., v99, posso usare v[i]
- identifier[expression] **è un'espressione dotata di indirizzo**
⇒ può ricevere un input, essere assegnata, passata per riferimento, ecc.

```
cin >> a[i];  
a[n+3]=3*2;  
scambia(a[i],a[i+1]);
```

Range degli array



- Gli elementi di un array di dimensione N sono numerati **da 0 a N-1**
- Esempio: se i vale 3, $a[i] = 7$ assegna il **quarto** elemento di a a 7

Esempi: uso di array e range

- lettura di un valore, stampa in ordine inverso:
 { ARRAY/array2_mia.cc }
- ... con errore sul range (problema tipico):
 { ARRAY/array2_errata.cc }
- g++ -fsanitize=bounds -fsanitize=bounds-strict può aiutare ad identificare a **run-time** la maggior parte dei casi di errore di **array subscripting out of bounds!**

Esempi: inizializzazione di array e range

- inizializzazione:

{ ARRAY/array3.cc }

- ... (errore: il range di un array inizia da 0):

{ ARRAY/array3_err.cc }

- ... senza esplicitazione della dimensione:

{ ARRAY/array3_bis.cc }

- inizializzazione a zero di default:

{ ARRAY/array4.cc }

- inizializzazione a tutti zero di default:

{ ARRAY/array4_bis.cc }

- vettore non inizializzato:

{ ARRAY/array5.cc }

Attenzione: Uscita dal range di un array

In C++, l'operatore “[]” permette di “uscire” dal range $0 \dots \text{dim}-1$ di un'array!

```
int A[dim]; int i=0;  
A[i-1]; // cella sizeof(int) bytes prima di A[0]  
A[i+dim]; // cella sizeof(int) bytes dopo A[dim-1]
```

⇒ Effetti potenzialmente catastrofici in caso di errore

⇒ È responsabilità del programmatore garantire che un'operazione di subscripting non esca mai dal range di un'array!

- Esempio:

errore, fuori range: catastrofico: va a sovrascrivere su una variabile ma non rivelato!

(usare `-fno-stack-protector`):

```
{ ARRAY/array3_errato.cc }
```

Funzione con parametri di tipo array

- Una funzione può avere un parametro formale del tipo “array di T”
 - Es: `float sum(float v[], int n) { ... }`
 - tipicamente si omette la dimensione (“`float v[]`”, non “`float v[dim]`”)
 - tipicamente associato al numero di elementi effettivamente utilizzati (dimensione virtuale)
- Il corrispondente parametro attuale è un array di oggetti di tipo T
 - Es:
`float a[DIM]; int n;`
`(...)`
`x = sum(a, n);`
- Nel passaggio, gli elementi dell’array **non vengono copiati**
 - N.B. viene copiato **solo l’indirizzo del primo elemento**
 - Equivalente a passare gli elementi dell’array **per riferimento**
 - È possibile impedire di modificarli usando la parola chiave `const`

Nota

Con allocazione statica di array, tipicamente si definiscono array sufficientemente grandi, e poi se ne usa di volta in volta solo una parte.

Passaggio di parametri array costanti

- È possibile definire passaggi di array **in sola lettura** (passaggio di array costante)
 - Sintassi: **(const** tipo identificatore **[**, **...**)
 - Es: **int** print(**const int** v $\texttt{[]}$, **...**) **(...**) **{...**
- Passaggio di array: il contenuto dell'array non viene duplicato
 \Rightarrow evito possibile spreco di tempo CPU e memoria
- Non permette di modificare gli elementi di v!
 - Es: v[3] = 5; //ERRORE!
 - \Rightarrow passaggio di informazione **solo dalla chiamante alla chiamata**
 - \Rightarrow solo un input alla funzione
- Usato per passare array **in input** alla funzione
 - efficiente (no spreco di CPU e memoria)
 - evita errori
 - permette di individuare facilmente gli input della funzione

Esempi

- passaggio di vettori, i/o di vettori, norma 1, somma, concatenazione di vettori:
{ ARRAY/leggimanipolaarray.cc }

Array e funzioni ricorsive

- E frequente effettuare operazioni **ricorsive** su array
 - Tipicamente il parametro di ricorsione definisce il range dei sotto-array correntemente analizzati
- somma di array 1 :
 { ARRAY/array_rec1_nocomment.cc }
- ..., chiamate tracciate:
 { ARRAY/array_rec1.cc }
- somma di array 2 :
 { ARRAY/array_rec2_nocomment.cc }
- ..., chiamate tracciate:
 { ARRAY/array_rec2.cc }
- somma di array 3 :
 { ARRAY/array_rec3_nocomment.cc }
- ..., chiamate tracciate:
 { ARRAY/array_rec3.cc }

Esercizi proposti

Vedere file ESERCIZI_PROPOSTI.txt

Corso “Programmazione 1”

Capitolo 06: Gli Array

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 28 ottobre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Esempi

- passaggio di vettori, i/o di vettori, norma 1, somma, concatenazione di vettori:
{ ARRAY/leggimanipolaarray.cc }

Nota

Se arr è un parametro di tipo array di Type di una funzione

⇒ **sizeof(arr) = sizeof(Type *)**.

```
void addElement(int arr[], int &d, const int x, const int p, const int N) {
    cout << "Array_of_size:" << sizeof(arr)/sizeof(int) << endl;
    // Su macchina a 64bit stampa 2 indipendentemente da size di array passato!
    cout << "======" << endl;
    cout << "Adding" << x << "at_position" << p << endl;
    if (p >= 0 && p <= d && d < N) {
        for(int i = d; i > p; i--)
            arr[i] = arr[i-1];
        arr[p] = x;
        d++;
    }
}
```

Problema: ricerca di un elemento in un array

Quanti passi richiede in media il cercare un elemento in un array di N elementi?

- Con un array **generico**:

- $\approx N/2$ se l'elemento è presente, $\approx N$ se non è presente
- $\Rightarrow O(N)$ (un numero proporzionale ad N)
- ES: $N = 1.000.000 \Rightarrow \leq 1.000.000$ passi
- **Ricerca lineare:**
`{ SORT/linear_search.cc }`

- Su un array **ordinato**:

- $\leq \lceil \log_2(N) \rceil$ se l'elemento è presente, $\lceil \log_2(N) \rceil$ se non è presente
- $\Rightarrow O(\log_2(N))$ (un numero proporzionale al logaritmo di N)
- ES: $N = 1.000.000 \Rightarrow \leq 20$ passi
- **Ricerca binaria:**
`{ SORT/binary_search.cc }`
- ... **versione ricorsiva:**
`{ SORT/binary_search_rec.cc }`

\Rightarrow In molte applicazioni, è vitale mantenere un array ordinato

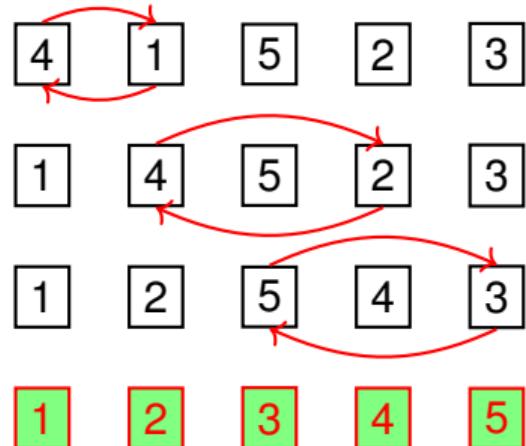
Metodi di ordinamento

Moltissimi metodi di ordinamento

- Selection sort
- Insertion sort
- Bubble sort
- Quick sort
- ...

Ordinamento per selezione: Selection Sort

- Cerco elemento più piccolo dell'array e lo scambio con il primo elemento dell'array.
- Cerco il secondo elemento più piccolo dell'array e lo scambio con il secondo elemento dell'array.
- Proseguo in questo modo fintanto che l'array non è ordinato.



Selection sort:

{ SORT/selection_sort.cc }

Ordinamento per selezione: Selection Sort

```
void selectionsort(int A[], int N) {  
    for (int i = 0; i < N - 1; i++) {  
        int min = i;  
        for(int j = i + 1; j < N; j++) // cerco elemento piu'  
            // piccolo nella parte di  
            // array ancora da ordinare  
        if (A[j] < A[min]) min = j;  
        swap(A[i], A[min]); // scambio elemento trovato  
            // con elemento dell'array  
            // ancora da ordinare  
    }  
}
```

Ordinamento per selezione: Selection Sort

Selection sort

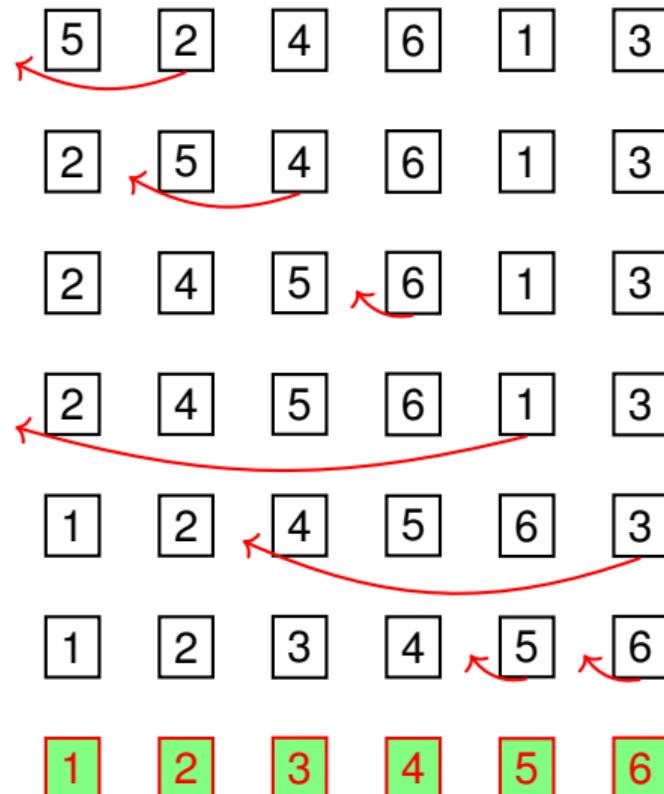
- Relativamente semplice:
- $O(N^2)$ passi in media
- ES: $N = 1.000.000 \Rightarrow \leq 1.000.000.000.000 = 10^{12}$ passi (!)

Ordinamento per inserzione: Insertion Sort

- È il metodo usato dai giocatori di carte per ordinare in mano le carte.
- Considero un elemento per volta e lo inserisco al proprio posto tra quelli già considerati (mantenendo questi ultimi ordinati).
 - L'elemento considerato viene inserito nel posto rimasto vacante in seguito allo spostamento di un posto a destra degli elementi più grandi.

Insertion sort:

{ SORT/insertion_sort.cc }



Ordinamento per inserzione : Insertion sort

```
void insertsort( int A[], int N) {  
    for(int i = N-1; i > 0; i--) // porto elemento  
        // piu' piccolo in A[0]  
    if (A[i] < A[i-1]) swap(A[i], A[i-1]);  
    for(int i = 2; i <= N-1; i++) {  
        int j = i;  
        int v = A[i];  
        while( v < A[j-1] ) {  
            A[j] = A[j-1]; j--;  
        }  
        A[j] = v;  
    }  
}
```

Ordinamento per inserzione: Insertion Sort

Insertion sort

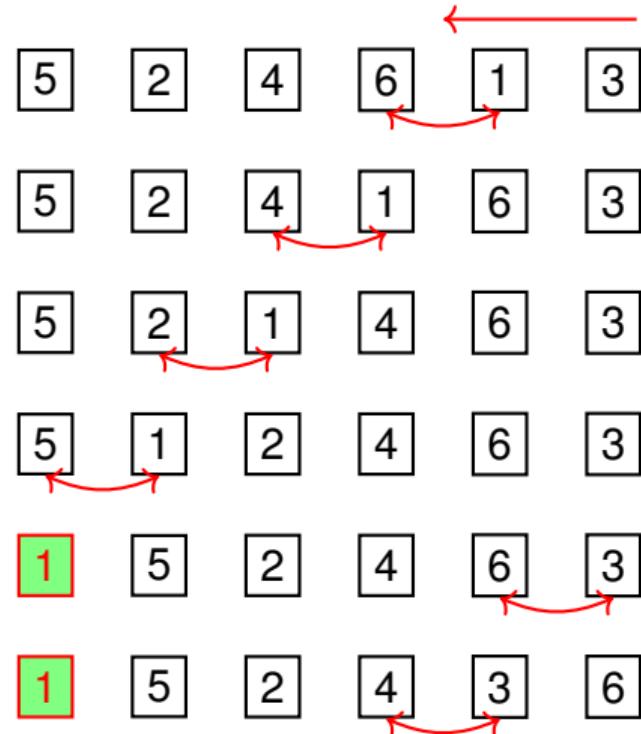
- Relativamente semplice:
- $O(N^2)$ passi in media
- ES: $N = 1.000.000 \Rightarrow \leq 1.000.000.000.000 = 10^{12}$ passi (!)

Ordinamento a bolle: Bubble Sort

- Si basa su scambi di elementi adiacenti se necessari, fino a quando non è più richiesto alcuno scambio e l'array risulta ordinato.

bubble sort:

{ SORT/bubblesort_nocomment.cc }

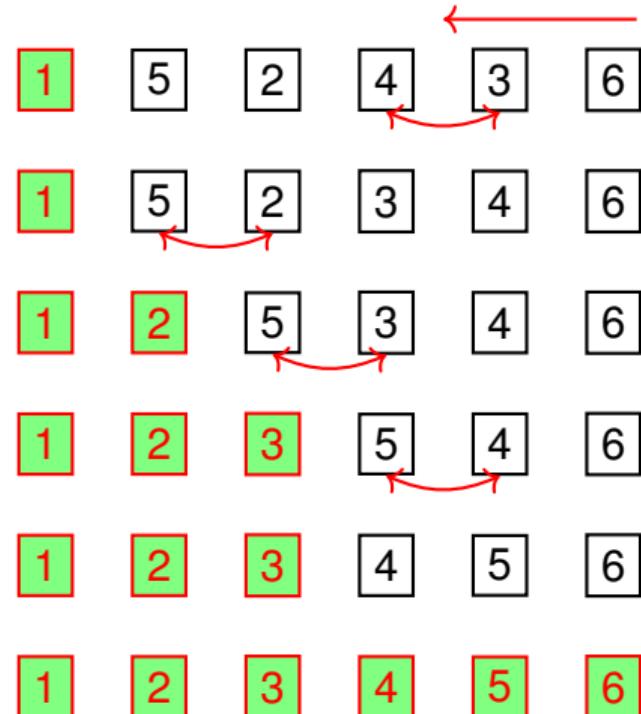


Ordinamento a bolle: Bubble Sort

- Si basa su scambi di elementi adiacenti se necessari, fino a quando non è più richiesto alcuno scambio e l'array risulta ordinato.

bubble sort:

{ SORT/bubblesort_nocomment.cc }



Metodi di ordinamento: Bubblesort

Bubblesort

- Relativamente semplice:
 - $O(N^2)$ passi in media
 - ES: $N = 1.000.000 \Rightarrow \leq 1.000.000.000.000 = 10^{12}$ passi (!)
-
- bubblesort semplice:
 { SORT/bubblesort_nocomment.cc }
 - ... con passi tracciati:
 { SORT/bubblesort.cc }
 - bubblesort ottimizzato:
 { SORT/bubblesort_opt.cc }

Ordinamento QuickSort

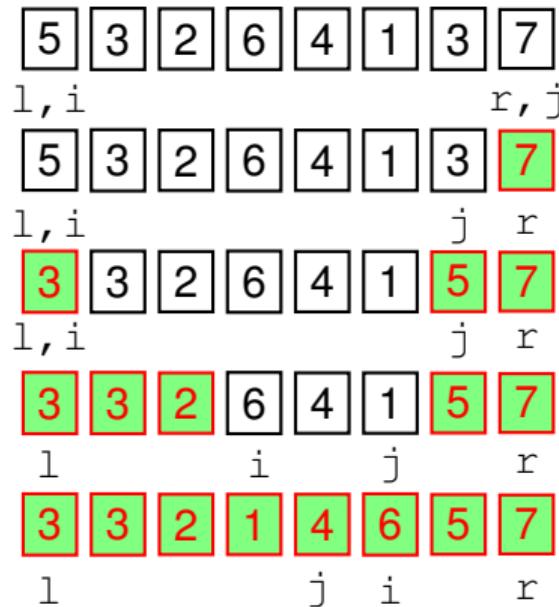
- È un algoritmo di ordinamento del tipo **divide et impera**.
- Si basa su un processo di **partizionamento** dell'array in modo che le seguenti tre condizioni siano verificate:
 - Per qualche valore di i , l'elemento $A[i]$ si trova al posto giusto.
 - Tutti gli elementi $A[0], \dots, A[i-1]$ sono minori od uguali ad $A[i]$.
 - Tutti gli elementi $A[i+1], \dots, A[N-1]$ sono maggiori od uguali ad $A[i]$.
- L'array è ordinato partizionando ed applicando ricorsivamente il metodo ai sotto array.

Ordinamento QuickSort (II)

- Scegliamo arbitrariamente un elemento (e.g. $A[r]$), che chiameremo pivot (o elemento di partizionamento).
- Scandiamo l'array dall'estremità sinistra fino a quando non troviamo un elemento $A[i] <= A[r]$.
- Scandiamo l'array dall'estremità destra fino a che non troviamo un elemento $A[j] >= A[r]$.
- Scambiamo $A[i]$ e $A[j]$, e iteriamo.
- Procedendo in questo modo si arriva ad una situazione in cui **tutti** gli elementi a sinistra di i sono minori di $A[r]$, mentre quelli a destra di j sono maggiori di $A[r]$.



Ordinamento QuickSort (III)



- Partizioniamo a partire da $A[0] = 5$.
 - Gli elementi dell'array precedenti ad $A[i]$ sono minori od uguali a 5.
 - Gli elementi dopo $A[j]$ sono maggiori od uguali a 5.
 - Ripartizioniamo i sotto array da 1 a $i-1$, e da $i+1$ a r .

Algoritmo QuickSort: (IV)

```
int partition(int A[],  
             int l, int r) {  
    int i = l-1, j = r, v = A[r];  
    while (true) {  
        while (A[++i] < v);  
        while (v < A[--j])  
            if (j == l) break;  
        if (i >= j) break;  
        swap(A[i], A[j]);  
    }  
    swap(A[i], A[r]);  
    return(i);  
}
```

```
void quicksort(int A[], int N) {  
    quicksort_aux(A, 0, N-1);  
}  
  
void quicksort_aux(int A[],  
                  int l,  
                  int r) {  
    if (r <= l) return;  
    int i = partition(A, l, r);  
    quicksort_aux(A, l, i-1);  
    quicksort_aux(A, i+1, r);  
}
```

Metodi di ordinamento: Quicksort

Quicksort

- Complesso
- $O(N \cdot \log_2(N))$ passi in media
- ES: $N = 1.000.000 \Rightarrow \leq 20.000.000 = 2 * 10^7$ passi

- quicksort semplice:
 { SORT/quicksort_nocomment.cc }
- ... con passi tracciati:
 { SORT/quicksort.cc }
- quicksort, con randomizzazione:
 { SORT/quicksort_rand.cc }

Nota

Quicksort algoritmo intrinsecamente ricorsivo!

Algoritmo di ordinamento: Shell Sort

- La lentezza dell'algoritmo Insertion Sort risiede nel fatto che le operazioni di scambio avvengono solo tra elementi contigui.
 - Esempio: se l'elemento più piccolo è in fondo all'array, occorrono N scambi per posizionarlo al posto giusto.
- Per migliorare questo algoritmo è stato pensato l'algoritmo Shell Sort.
- L'idea è quella di organizzare l'array in modo che esso soddisfi la proprietà per cui gli elementi aventi tra loro distanza h costituiscono una sequenza ordinata, indipendentemente dall'elemento di partenza.
 - Se si applica l'algoritmo con una sequenza di h che termina con 1, si ottiene un file ordinato.

```
void ShellSort(int A[],  
              int l, int r)  
{  
    int h;  
    for(h = 1; h <= (r-1)/9;  
        h = 3*h+1);  
    for( ; h > 0; h /= 3)  
        for(int i = l+h;  
            i <= r; i++) {  
            int j = i;  
            int v = A[i];  
            while((j >= l + h) &&  
                  (v < A[j-h])) {  
                A[j] = A[j-h];  
                j = j - h;  
            }  
            A[j] = v;  
        }  
}
```

Metodi di ordinamento: Shell Sort

Shell Sort

- L'implementazione proposta $O(N^{3/2})$ passi in media
- ES: $N = 1.000.000 \Rightarrow \leq 1.000.000.000 = 10^9$ passi (!)
- Usando sequenze particolari di h si possono ottenere prestazioni diverse (e.g. $O(N \cdot \log_2(N)^2)$).

Metodi di ordinamento: Algoritmi a confronto

Algoritmo	$O(..)$
Selection sort	$O(N^2)$
Insertion sort	$O(N^2)$
Bubblesort sort	$O(N^2)$
Quick sort	$O(N \cdot \log_2(N))$
Shell sort	$O(N^{3/2})$
Merge sort*	$O(N \cdot \log_2(N))$

* Da cercare e implementare come esercizio

Algoritmi di sorting a confronto:
{ SORT/sorting_all.cpp }

Altre operazioni su array ordinati

Fusione ordinata di due array ordinati (merging)

- ES: $\text{merge}([1\ 3\ 4\ 8], [2\ 3\ 5\ 6]) \Rightarrow [1\ 2\ 3\ 3\ 4\ 6\ 8]$
- $O(N_1 + N_2)$
- merging:
{ SORT/merge.cc }

Inserimento di un elemento in un array ordinato

- ES: $\text{insert}, 5, [1\ 3\ 4\ 8]) \Rightarrow [1\ 3\ 4\ 5\ 8]$
- $O(N)$
- Equivalente a $\text{merge}([5], [1\ 3\ 4\ 8])$

Esercizi proposti

Vedere file ESERCIZI_PROPOSTI.txt

Corso “Programmazione 1”

Capitolo 06: Gli Array

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 2 novembre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Array multidimensionali

- In C++ è possibile dichiarare array i cui elementi siano a loro volta degli array, generando degli **array multidimensionali** (matrici)
- Sintassi:

```
tipo id[dim1] [dim2] ... [dimN];
```

```
tipo id[dim1] [dim2] ... [dimN]={lista_valori};
```

```
tipo id[] [dim2] ... [dimN]={lista_valori};
```

⇒ un array multidimensionale $dim_1 \cdot \dots \cdot dim_n$ può essere pensato come un array di dim_1 array multidimensionali $dim_2 \cdot \dots \cdot dim_n$

- Esempio:

```
int MAT[2][3];
```

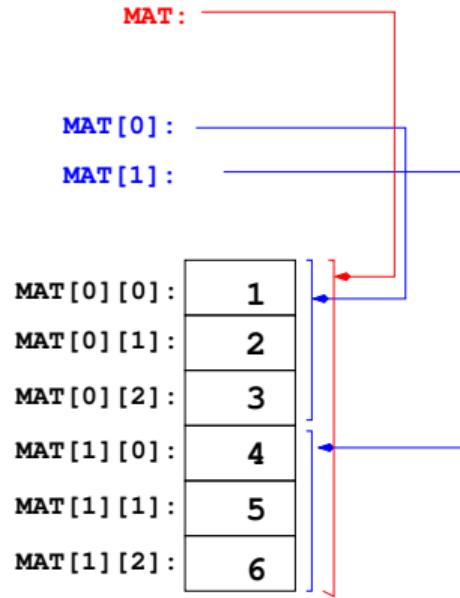
```
int MAT[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

```
int MAT[][3] = {{1, 2, 3}, {4, 5, 6}};
```

- Le **dimensioni** devono essere valutabili durante la compilazione
- Elementi mancanti vengono sostituiti da zeri

Struttura di un array bidimensionale (statico)

```
int MAT[2][3] = {{1,2,3},{4,5,6}};
```



Esempio matrice statica:

```
{ MATRICI/matrix_stata.cc }
```

Esempi: inizializzazione di array bidimensionali

- con inizializzazione:

```
{ MATRICI/def_mat1.cc }
```

- con inizializzazione parziale:

```
{ MATRICI/def_mat2.cc }
```

- con inizializzazione parziale (2):

```
{ MATRICI/def_mat3.cc }
```

- con inizializzazione, senza valore iniziale:

```
{ MATRICI/def_mat4.cc }
```

- inizializzazione: errore:

```
{ MATRICI/def_mat5.cc }
```

Esempi: passaggio di array bidimensionali a funzioni

- solo una dimensione fissa, utilizzate in parte:
 { MATRICI/matrix3.cc }
- Err: (è necessario passare la dimensione):
 { MATRICI/matrix3_err.cc }

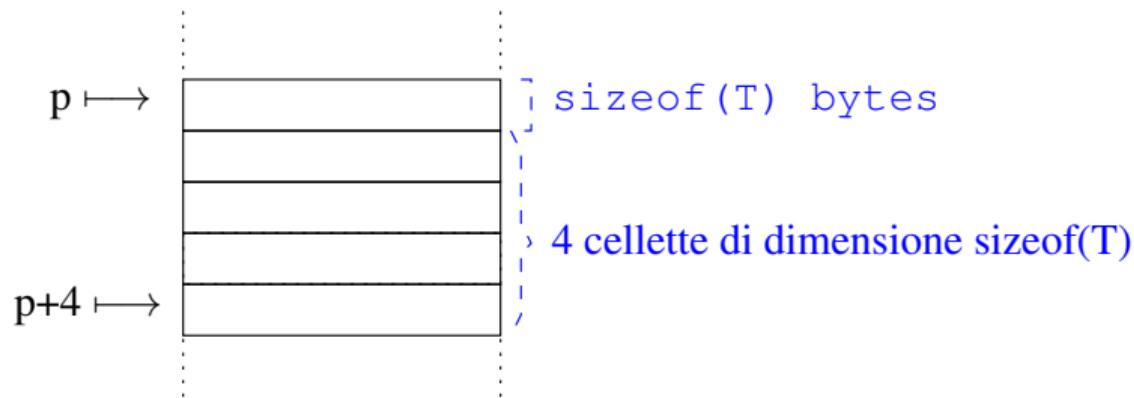
Aritmetica di Puntatori ed Indirizzi (richiamo)

Gli indirizzi e i puntatori hanno un'aritmetica:

se p è di tipo $*T$ e i è un intero, allora:

- $p+i$ è di tipo $*T$ ed è l'indirizzo di un oggetto di tipo T che si trova in memoria dopo i posizioni di dimensione $\text{sizeof}(T)$
- analogo discorso vale per $p++$, $++p$, $p--$, $--p$, $p+=i$, ecc.

$\Rightarrow i$ viene implicitamente moltiplicato per $\text{sizeof}(T)$

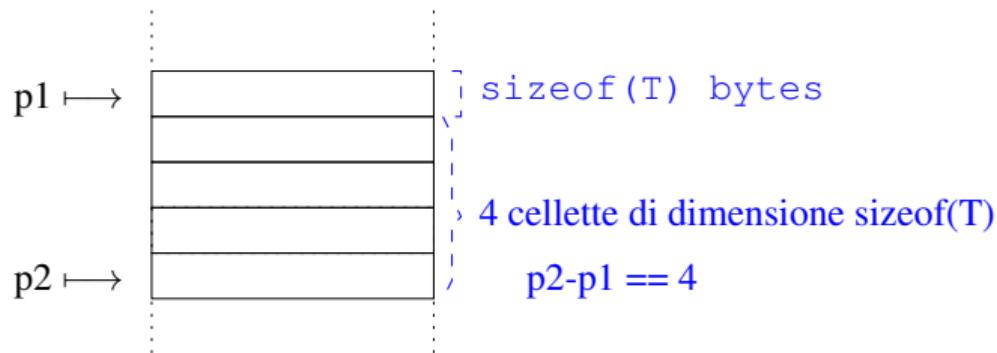


Aritmetica di Puntatori ed Indirizzi (richiamo) II

se $p1$, $p2$ sono di tipo $*T$, allora:

- $p2 - p1$ è un intero ed è il numero di posizioni di dimensione $\text{sizeof}(T)$ per cui $p1$ precede $p2$ (negativo se $p2$ precede $p1$)
- si possono applicare operatori di confronto $p1 < p2$, $p1 \geq p2$, ecc.

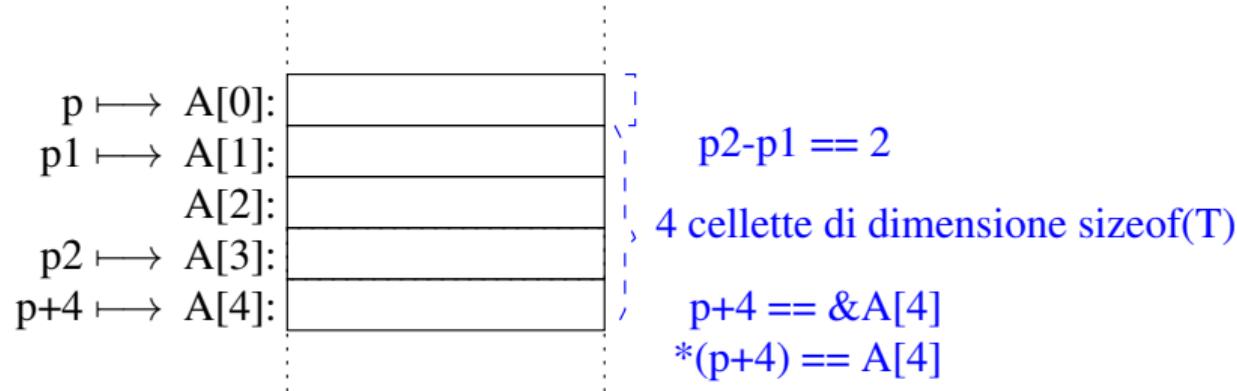
$\Rightarrow p2 - p1$ viene implicitamente diviso per $\text{sizeof}(T)$



Esempio di operazioni aritmetiche su puntatori (richiamo):
{ ARRAY_PUNT/aritmetica_punt.cc }

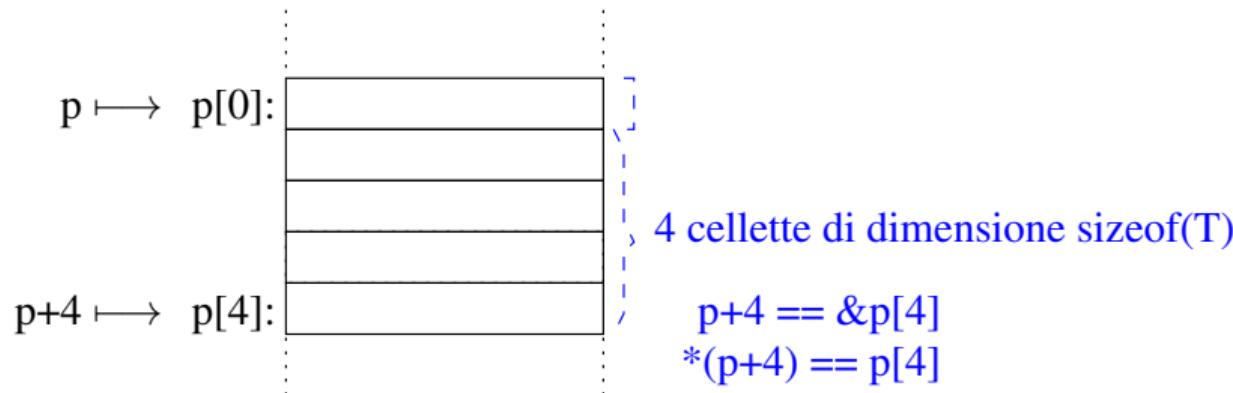
Puntatori ad elementi di un array

- Se un puntatore p punta al primo elemento di un array A , l'espressione $p+i$ è l'indirizzo dell' i -esimo elemento dell'array A
 \Rightarrow valgono $p+i == \&(A[i])$ e $*(p+i) == A[i]$
- Se due puntatori dello stesso tipo p_1, p_2 puntano ad elementi di uno stesso array, p_2-p_1 denota il numero di elementi compresi tra p_1 e p_2 (p_2-p_1 negativo se p_2 precede p_1)



Array e puntatori

- Il nome di un array è (implicitamente equivalente a) una costante puntatore al primo elemento dell'array stesso
- se p è il nome di puntatore o di un array, le espressioni $\&p[i]$ e $p+i$ sono equivalenti, come lo sono $p[i]$ e $*(p+i)$



Esempi

- relazione tra array e puntatori:

{ ARRAY_PUNT/punt_vett.cc }

- idem, con doppio avanzamento:

{ ARRAY_PUNT/punt_vett1.cc }

- scansione array con puntatori:

{ ARRAY_PUNT/scansione_array.cc }

- ..., variante:

{ ARRAY_PUNT/scansione_array2.cc }

Passaggio di Array tramite Puntatore

- Nelle chiamate di funzioni viene passato solo l'indirizzo alla prima locazione dell'array,
⇒ un parametro formale array può essere specificato usando indifferentemente la notazione degli array o dei puntatori
- Le seguenti scritture sono equivalenti (array semplici):

```
int f(int arr[dim]);  
int f(int arr[]);  
int f(const int *arr);
```

- Le seguenti scritture sono equivalenti (array multi-dimensionali):

```
int f(int arr[dim1][dim2]...[dimN]);  
int f(int arr[] [dim2]...[dimN]);  
int f(const int *arr[dim2]...[dimN]);
```

Esempi

- array passati come tali (richiamo):
 { ARRAY_PUNT/concatena_array.cc }
- ... passati come puntatori:
 { ARRAY_PUNT/concatena_array1.cc }
- ... passati e manipolati come puntatori:
 { ARRAY_PUNT/concatena_array2.cc }
- passaggio con par. attuale puntatore, par. formale array:
 { ARRAY_PUNT/concatena_array3.cc }

Restituzione di un Array

Problema importante

Una funzione può restituire un array (allocato staticamente)?

- Sì, ma solo se è allocato staticamente **esternamente** alla funzione!
(es. parametro formale, array globale)
- No, se è allocato staticamente **internamente** alla funzione!

- corretta, restituisce array parametro formale:
 { ARRAY_PUNT/restituzione_array.cc }
- corretta, restituisce array globale:
 { ARRAY_PUNT/restituzione_array1.cc }
- compila, ma fa disastri:
 { ARRAY_PUNT/err_restituzione_array2.cc }

Esercizi proposti

Vedere file ESERCIZI_PROPOSTI.txt

Corso “Programmazione 1”

Capitolo 07: Stringhe e File di Testo

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 3 novembre 2020

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Outline

1 Stringhe

2 Argomenti da linea di comando

3 I/O con File di Testo

Stringhe (C)

- Una stringa C è un array di **char**, il cui ultimo elemento è il carattere nullo ('\0')
 - Esempio: **char** stringa[] = "Ciao";
 - Equivalente a **char** stringa[] = {'C', 'i', 'a', 'o', '\0'};
- ⇒ contiene 5 elementi: i 4 caratteri della costante stringa e il carattere nullo che viene inserito automaticamente
- ⇒ La dimensione dell'array deve essere maggiore di almeno 1 rispetto al numero di caratteri che si vuole rappresentare

Nota:

dato che negli array non è definito l'assegnamento, l'uso di una costante stringa per specificare il valore di una stringa è permesso **solo nell'inizializzazione**

- L'esempio di cui sopra:
{ STRINGS/strings2.cc }

Input e Output di Stringhe

- Gli operatori di I/O `>>`, `<<` operano direttamente su stringhe!
- L' **operatore di ingresso** `>>`:
 1. legge caratteri da `cin`
 2. li memorizza in sequenza finché non incontra una spaziatura (che non viene letta)
 3. memorizza '`\0`' nella stringa dopo l'ultimo carattere letto
 4. termina l'operazione
- L' **operatore di uscita** `<<` scrive in sequenza su `cout` i caratteri della stringa, fino al primo '`\0`' (che non viene scritto)

Esempio

Legge una stringa di al più 255 caratteri e la stampa:

```
char buffer[256];
cin >> buffer;
cout << buffer;
```

Esempi

- input/output di stringhe, usare `./a.out < inputs/in: { STRINGS/strings3.cc }`
- variante, con `cin`-loop:
`{ STRINGS/strings3_cinloop.cc }`

Alcune funzioni utili della libreria <iostream> (Ripasso)

- `cin.eof()`: ritorna un valore diverso da 0 se lo stream `cin` ha raggiunto la sua fine (End Of File)
 - va usato sempre dopo almeno un'operazione di lettura
 - richiede un separatore dopo l'ultimo elemento letto
- `cin.fail()`: ritorna un valore diverso da 0 se lo stream `cin` ha rilevato uno stato di errore (e.g. stringa per `int`) o un end-of-file
 - non necessariamente usato dopo almeno un'operazione di lettura
 - non richiede un separatore dopo l'ultimo elemento letto
- `cin.clear()`: ripristina lo stato normale dello stream `cin` dallo stato di errore
- uso di `cin.eof()` – usare `inputs/in1` e `inputs/in1.bis`:
`{ STRINGS/converti.cc }`
- uso di `cin.fail()` – usare `inputs/in1` e `inputs/in1.bis`:
`{ STRINGS/converti1.cc }`
- uso di `cin.clear()` – usare `inputs/in1.tris`:
`{ STRINGS/converti2.cc }`

Alcune funzioni utili della libreria <iostream> II

Nelle funzioni seguenti `s` è una stringa, `c` è un carattere e `n` un intero:

- `cin.getline(s, n)`: legge da `cin` una riga in `s` fino a capo linea, per un massimo di `n-1` caratteri (lo '`\n`' non viene letto)
 - restituisce (un oggetto equivalente a) 0 se incontra `eof`
- `cin.get(c)`: legge da `cin` in `c` un singolo carattere (spaziature comprese), restituisce `c` ('`\0`' se `c` è `eof`)
- `cout.put(c)`: scrive su `cout` il singolo carattere `c`
- uso di `cin.getline(char *, int)`:
{ STRINGS/strings4_while.cc }
- uso di `cin.get` (usare inputs/silvia.txt come input):
{ STRINGS/agosti.cc }
- Esempio uso di `cin.put`:
{ STRINGS/strings7.cc }

Alcune funzioni utili della libreria <cstring>

Nelle funzioni che seguono `s` e `t` sono stringhe e `c` è un carattere:

- `strlen(s)`: restituisce la lunghezza di `s`
- `strchr(s, c)`: restituisce un puntatore alla prima occorrenza di `c` in `s`, oppure `NULL` se `c` non si trova in `s`
- `strrchr(s, c)`: come sopra ma per l'ultima occorrenza di `c` in `s`
- `strstr(s, t)`: restituisce un puntatore alla prima occorrenza della sottostringa `t` in `s`, oppure `NULL` se `t` non si trova in `s`
- `strcpy(s, t)`: copia `t` in `s` e restituisce `s`
- `strncpy(s, t, n)`: copia `n` caratteri di `t` in `s` e restituisce `s`
- `strcat(s, t)`: concatena `t` al termine di `s` e restituisce `s`
- `strncat(s, t, n)`: concatena `n` caratteri di `t` al termine di `s` e restituisce `s`
- `strcmp(s, t)`: restituisce un valore negativo, nullo o positivo se `s` è alfabeticamente minore, uguale o maggiore di `t`

Esempi

- `strlen`:
 { `STRINGS/strings13.cc` }
- `strchr`, `strrchr`, `strstr` (ricerca di caratteri e stringhe in una stringa):
 { `STRINGS/strings14.cc` }
- `strcpy`:
 { `STRINGS/strings15.cc` }
- `strncpy`:
 { `STRINGS/strings16.cc` }

Esempi II

- `strcat` (attenzione, c'è un errore, dove?):
 { `STRINGS/strings17.cc` }
- **versione corretta:**
 { `STRINGS/strings17_correct.cc` }
- (**compilare con `-fno-stack-protector`**) **effetto catastrofico:**
 { `STRINGS/strings17_catastrophic.cc` }
- `strncat` (può dare errore a seconda di opzioni di compilazione):
 { `STRINGS/strings18.cc` }
- `strcmp`:
 { `STRINGS/strings19.cc` }

Esercizi proposti

Vedere file `ESERCIZI_PROPOSTI.txt`

Argomenti da linea di comando

- In C++ è possibile passare ai programmi argomenti (es. valori numerici, nomi di file,...) **direttamente da linea di comando**

```
>./a.out 1000 22.5 miofile
```

- Possibile tramite due **parametri formali predefiniti** della funzione `main`:

```
int main (int argc, char * argv[]) {}
```

- l'intero `argc`, in cui viene automaticamente copiato il numero delle parole della riga di comando (“./a.out” o analogo inclusa)
- l'array di puntatori a caratteri (stringhe) `argv` in cui vengono automaticamente copiate le parole della linea di comando

Nota

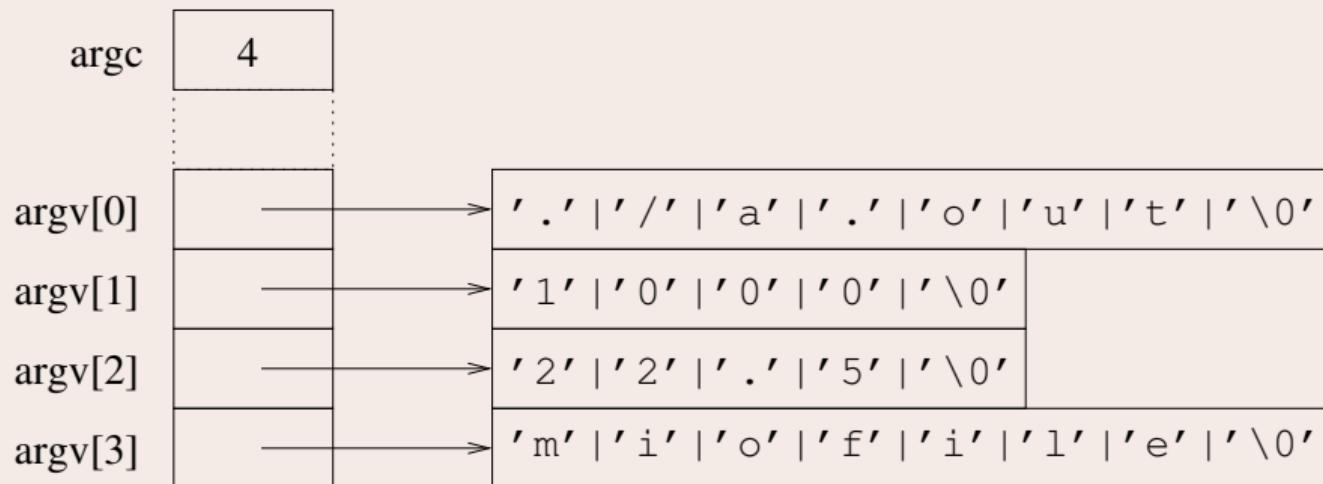
Gli argomenti sono **stringhe**: se rappresentano numeri, devono essere convertiti tramite le funzioni `atoi` o `atof` della libreria `<cstdlib>`

Argomenti da linea di comando II

```
int main (int argc, char * argv[])
```

```
{...}
```

```
>./a.out 1000 22.5 miofile
```



Esempi

- Esempio generico di uso di argc&argv:

{ STRINGS/argcargv.cc }

- .. con parametri numerici:

{ STRINGS/ival.cc }

Stream e I/O su File di Testo

- In C++ sono possibili operazioni di I/O **direttamente da file di testo** (senza usare <, >) tramite la libreria `<fstream>`
- È possibile **definire stream**, a cui associare (i nomi di) file di testo.
- Lo stream viene **aperto** e associato al nome di un file tramite il comando `open`, in tre possibili modalità
 - **lettura** da file,
 - **scrittura** su file,
 - **scrittura a fine file (append)**.
- Lo stream può essere utilizzato per tutte le operazione di lettura [resp. scrittura] a seconda della modalità di apertura
- Uno stream, quando è stato utilizzato, può essere chiuso mediante la funzione `close`

`fstream` “eredita” da `iostream` sostanzialmente tutti i suoi operatori e funzioni di lettura e scrittura, nelle rispettive modalità
(Es: `<<, >>, get, put, getline, eof, fail, clear, ...`)

Operazioni su Stream: Sintassi

- **Definizione di uno stream:**

- **Sintassi:** `fstream nomestream;`
- **Esempio:** `fstream myin, myout, myapp;`

- **Apertura di uno stream:**

- **Sintassi:** `nomestream.open (nomefile, modo);`
- **Es:**

```
myin.open ("ingresso.txt", ios::in); //lettura  
myout.open ("uscita.txt", ios::out); //scrittura  
myapp.open ("uscita2.txt", ios::out|ios::app); //app.
```

- **Utilizzo di uno stream:**

- **Sintassi:** analoghe a `cin` e `cout`
 - **Es:**
- ```
myin >> a; myout << x; myin.get (c); myapp.put (c);...
```

- **Chiusura di uno stream:**

- **Sintassi:** `nomestream.close();`
- **Es:** `myin.close(); myout.close();`

# Apertura di un file

- Apertura in modalità **lettura** (`ios::in`):
  - il file associato deve già essere presente
  - il puntatore si sposta all'inizio dello stream
- Apertura in modalità **scrittura** (`ios::out`):
  - il file associato se non è presente viene creato
  - il puntatore si posiziona all'inizio dello stream (sovrascrivendo il file)
- Apertura in modalità **append** (`ios::out | ios::app`):
  - il file associato se non è presente viene creato
  - il puntatore si posiziona alla fine dello stream

# Chiusura di uno stream

- Alla fine del programma tutti gli stream aperti vengono automaticamente chiusi
- Una volta chiuso,
  - uno stream può essere riaperto in qualunque modalità e associato a qualunque file
  - uno stream per essere utilizzato deve essere riaperto in qualunque modalità e associato a qualunque file

## Nota:

È buona prassi di programmazione **chiudere** ogni stream aperto quando non più necessario (il sistema operativo ha un limite sul numero di file che possono essere aperti per un programma, vedi `ulimit -a` su bash).

# Uso di `fstream` con `argc` e `argv`

- È desiderabile poter passare i nomi dei file al programma:

- Es: `> ./a.out pippo pluto`

⇒ nomi dei file passati tramite `argc` e `argv`:

```
int main (int argc, char * argv[])
{
 fstream myin,myout;
 myin.open(argv[1],ios::in);
 myout.open(argv[2],ios::out);
```

- È necessario gestire l'errore utente e la mancata apertura:

```
if (argc!=3) {
 cerr << "Usage: ./a.out <source> <target>\n";
 exit(0);
}
if (myin.fail()) {
 cerr << "Il file " << argv[1] << " non esiste\n";
 exit(0);
}
```

# Esempi

- esempio di uso di `fstream` (usa `inputs/in1` e `inputs/in1.bis`):  
    { `IO_SU_FILES/converti1.cc` }
- come sopra, con `cin` loop:  
    { `IO_SU_FILES/converti2.cc` }
- effettua una copia di un file :  
    { `IO_SU_FILES/copiafile.cc` }
- appende un file ad un altro file:  
    { `IO_SU_FILES/appendifile.cc` }

# Esercizi proposti

**Vedere file `ESERCIZI_PROPOSTI.txt`**

# Corso “Programmazione 1”

## Capitolo 08: Gestione Dinamica della Memoria

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it  
**Stefano Berlato** - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 9 novembre 2020

# Terms of Use and Copyright

## USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

## SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

## COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

# Outline

- 1 Allocazione e Deallocazione Dinamica
- 2 Array e Stringhe Allocati Dinamicamente
- 3 Array Multidimensionali Allocati Dinamicamente

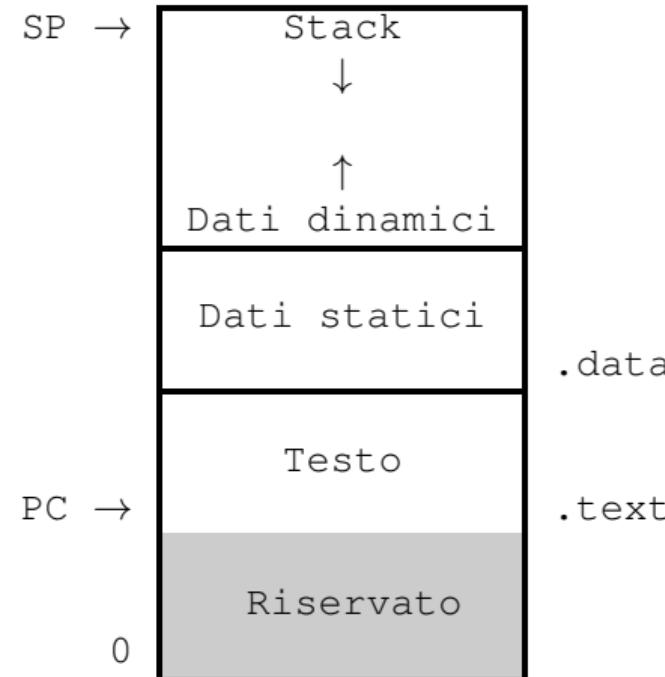
# Uso Dinamico della Memoria

- L'allocazione statica obbliga a definire la struttura e la dimensione dei dati a priori (compiler time)
  - non sempre questo è accettabile e/o conveniente
  - Esempio: dimensione di un array fissa e stabilità a priori (`int a[100];`)
- In C++ è possibile gestire la memoria anche **dinamicamente**, ovvero **durante l'esecuzione del programma**
- Memoria allocata nello **store** (**heap**), un'area esterna allo stack
- L'accesso avviene tramite **puntatori**
- L'allocazione/deallocazione è gestita dagli operatori **`new`** e **`delete`**

# Modello di gestione della memoria per un programma (ripasso)

Area di memoria allocata ad un'esecuzione di un programma:

- **Area programmi e costanti:** destinata a contenere le **istruzioni** (in linguaggio macchina) del programma
- **Area dati statici:** destinata a contenere variabili **globali** o **allocate staticamente** e le **costanti** del programma.
- **Area heap:** destinata a contenere le variabili dinamiche (di dimensioni non prevedibili a tempo di compilazione) del programma.
- **Area stack:** destinata a contenere le **variabili locali** e i **parametri formali** delle funzioni del programma.



# Allocazione: l'operatore **new**

- Sintassi:

- **new** tipo;
- **new** tipo (valore); (con inizializzazione del valore)
- **new** tipo [dimensione]; (per gli array)

dove dimensione può essere un'espressione **variabile** e valore deve essere un valore costante di tipo tipo

- Esempio:

```
int *p, *q;
char *stringa;

p = new int;
q = new int (5); // Assegna valore 5 all'area di memoria
stringa = new char[3*i];
```

## Allocazione: l'operatore **new** - II

- L'operatore **new/new** [dimensione]:

1. alloca un'area di memoria adatta a contenere un oggetto (o dimensione oggetti) del tipo specificato
2. la inizializza a valore (se specificato)
3. ritorna l'**indirizzo** (del primo elemento) di tale area  
    ⇒ tipicamente assegnato ad un puntatore

# Deallocazione: l'operatore **delete**

- Sintassi:

- **delete** indirizzo;
- **delete**[] indirizzo; (per gli array)

dove il valore dell'espressione `indirizzo` deve essere l'indirizzo di una celletta precedentemente allocata da una chiamata a **new**

- Esempio:

```
int *p;
char *stringa;

p = new int;
stringa = new char[30];
delete p;
delete[] stringa;
```

## Deallocazione: l'operatore **delete** - II

- L'operatore **delete/delete []** dealloca l'area di memoria precedentemente allocata a partire dall'indirizzo specificato
  - se l'indirizzo non corrisponde ad una chiamata a **new**  $\Rightarrow$  errore
  - un'area allocata da **new** [resp. **new [ . . . ]**] deve essere deallocata con **delete** [resp. **delete []**] (altrimenti comp. non specificato)
- Al termine del programma anche la memoria allocata con **new** viene automaticamente deallocata

# Nota su deallocazione

Deallocare un'area di memoria:

- significa che quell'area non è più “riservata”  
    ⇒ può essere ri-allocata
- non significa che il suo contenuto venga cancellato!  
    ⇒ valore potenzialmente ancora accessibile per un po' di tempo (**non noto a priori!!!**)  
    ⇒ **facile** non accorgersi di situazione di errore!!!

# Esempi: **new** e **delete** su variabili semplici

- Esempio con **new**:

```
{ ALLOC_DINAMICA/new1.cc }
```

- variante:

```
{ ALLOC_DINAMICA/new2.cc }
```

- ... con inizializzazione:

```
{ ALLOC_DINAMICA/new3.cc }
```

- esempio di allocazione e deallocazione:

```
{ ALLOC_DINAMICA/newdelete1.cc }
```

- tentativo di deallocazione di variabile statica:

```
{ ALLOC_DINAMICA/newdelete2.cc }
```

- indipendenza dal nome del puntatore:

```
{ ALLOC_DINAMICA/newdelete3.cc }
```

# Durata di un'Allocazione Dinamica

- Un oggetto creato dinamicamente resta allocato finché:
  - non viene esplicitamente deallocated con l'operatore `delete`, oppure
  - il programma non termina
- La memoria allocata con `new` non esplicitamente deallocated con `delete`, può risultare non più disponibile per altri programmi
  - ⇒ spreco di memoria ([memory leak](#))
  - ⇒ degrado delle prestazioni della macchina

## Regola aurea

In un programma, si deve sempre esplicitamente deallocate tutto quello che si è allocato dinamicamente non appena non serve più.

# Gestione dinamica della memoria: pro e contro

- Pro:

- Gestione efficiente della memoria: alloca solo lo spazio necessario
- Permette la creazione di strutture dati dinamiche (liste, alberi, ...)

- Contro:

- Molto più difficile da gestire
- Facile introdurre errori e/o memory leaks

## Nota

- Esistono strumenti a supporto dell'identificazione di memory leaks:
  - Open source (e.g. valgrind, gperftool, -fsanitize=...)
  - ... e commerciali (e.g. Parasoft Insure++, IBM Rational Purify)

# Allocazione dinamica di Array

- Consente di creare a run-time array di dimensioni diverse a seconda della necessità
- Un array dinamico è un **puntatore** al primo elemento della sequenza di celle

```
int n;
cin >> n;
int *a = new int[n]; //allocazione dell'array
for (int i=0; i<n; i++) {
 cout << endl << i+1 << ":";
 cin >> a[i]; };
delete[] a; //deallocazione dell'array
```

- Esempio di cui sopra esteso:  
 { ALLOC\_DINAMICA/prova.cc }
- **allocazione dinamica array + inizializzazione: non più ammessa:**  
 { ALLOC\_DINAMICA/prova5.cc }

# Allocazione dinamica di Stringhe

- Consente di creare a run-time stringhe di dimensioni diverse
- Una stringa dinamica è un puntatore al primo elemento della sequenza di caratteri, terminata da '\0'
- L'I/O di una stringa dinamica è gestita automaticamente dagli operatori >>, <<
- Tutte le primitive su stringhe in <cstring> applicano anche alle stringhe dinamiche

```
char * sc, *sb = new char [20];
cin >> sb;
sc = new char[strlen(sb)+1];
strcpy(sc, sb);
cout << sc;
```

- Esempio di cui sopra esteso:  
{ ALLOC\_DINAMICA/prova2.cc }

## Fallimento di `new`

- L'esecuzione di una `new` può **non andare a buon fine** (memoria destinata al programma esaurita)
  - in tal caso lo standard C++ prevede che, se non diversamente specificato, `new` richieda al s.o. di abortire il programma.
- Soluzione: usare "`new (nothrow)`"
  - Con l'opzione "`nothrow`", `new` non abortisce ma restituisce "`NULL`" **in caso di impossibilità ad allocare** la memoria richiesta.
  - Esempio:

```
char *p = new (nothrow) char[mymax];
...
if (p!=NULL) ...
```

# Esempi

Suggerimento: aprire shell con comando “top” attivo

- **allocazione eccessiva:**  
    { ALLOC\_DINAMICA/prova3.cc }
- ... con deallocazione:  
    { ALLOC\_DINAMICA/prova3\_bis.cc }
- deallocazione non dipende dal nome del puntatore!:  
    { ALLOC\_DINAMICA/prova3\_tris.cc }
- **delete []** richiede l'indirizzo del primo elemento allocato!:  
    { ALLOC\_DINAMICA/prova3\_err.cc }
- uso di **new (nothrow)**:  
    { ALLOC\_DINAMICA/prova4\_nothrow.cc }

# Corso “Programmazione 1”

## Capitolo 08: Gestione Dinamica della Memoria

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it  
**Stefano Berlato** - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 10 novembre 2020

# Terms of Use and Copyright

## USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

## SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

## COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

# Esempi

Suggerimento 1: aprire shell con comando “top” attivo

Suggerimento 2: su bash provare ad eseguire con (ulimit -v 5000000; ./a.out)

- **allocazione eccessiva:**  
    { ALLOC\_DINAMICA/prova3.cc }
- ... **con deallocazione:**  
    { ALLOC\_DINAMICA/prova3\_bis.cc }
- **deallocazione non dipende dal nome del puntatore!:**  
    { ALLOC\_DINAMICA/prova3\_tris.cc }
- **delete []** richiede l'indirizzo del primo elemento allocato!:  
    { ALLOC\_DINAMICA/prova3\_err.cc }
- **uso di new (nothrow):**  
    { ALLOC\_DINAMICA/prova4\_nothrow.cc }

# Restituzione di Array II<sup>1</sup>

Una funzione può restituire un array se allocato **dinamicamente** al suo interno.

```
int *times(int a[], ...) {
 int * b = new int[10];
 ...
 return b;
}
```

```
int v[10] = {1,2,3,4,5,6,7,8,9,10};
int * w = times(v, ...);
```

- **versione non corretta, vedi Ch. 6:**  
    { ALLOC\_DINAMICA/err\_restituzione\_array2.cc }
- **versione corretta:**  
    { ALLOC\_DINAMICA/restituzione\_array.cc }

<sup>1</sup>Si veda per confronto la slide omonima in Cap. 06.

# Responsabilità di allocazione/deallocazione dinamica

Quando si usa allocazione dinamica di un dato (e.g. di un array) che viene passato tra più di una funzione, il programmatore deve:

- decidere **quale funzione ha la responsabilità di allocare il dato**
  - rischio di mancanza di allocazione  $\Rightarrow$  segmentation fault
  - rischio di allocazioni multiple  $\Rightarrow$  memory leak
- decidere **quale funzione ha la responsabilità di dealloccarlo**
  - rischio di mancanza di deallocazione  $\Rightarrow$  memory leak
  - rischio di deallocazioni multiple  $\Rightarrow$  segmentation fault
- adeguare **il passaggio di parametri delle funzioni** in tal senso.
  - rischio di mancanza di allocazione  $\Rightarrow$  segmentation fault

## Nota importante

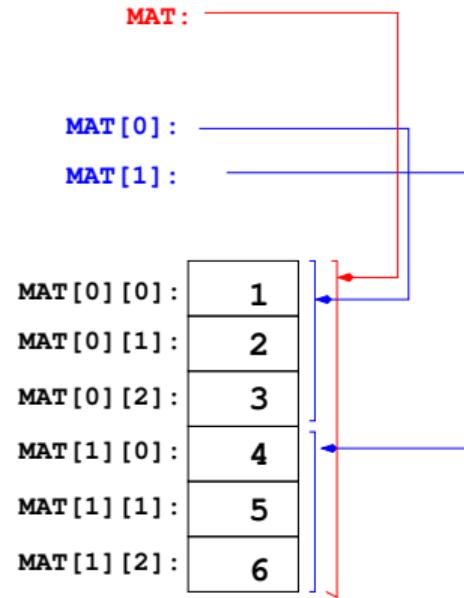
È fondamentale **concordare preventivamente la responsabilità dell'allocazione e deallocazione** quando il codice è sviluppato in team!

# Esempi

- nessuna allocazione, passaggio per valore:  
  { ALLOC\_DINAMICA/responsabilita1\_err.cc }
- allocazione esterna alla funzione get, passaggio per valore:  
  { ALLOC\_DINAMICA/responsabilita1.cc }
- allocazione interna alla funzione get, passaggio per valore:  
  { ALLOC\_DINAMICA/responsabilita2\_err.cc }
- allocazione interna alla funzione get, passaggio per riferimento:  
  { ALLOC\_DINAMICA/responsabilita2.cc }
- doppia allocazione, passaggio per riferimento:  
  { ALLOC\_DINAMICA/responsabilita2\_memleak.cc }
- deallocazione interna alla funzione print (insensata e pericolosa)!:  
  { ALLOC\_DINAMICA/responsabilita3.cc }
- deallocazione interna alla funzione print (insensata e pericolosa)!:  
  { ALLOC\_DINAMICA/responsabilita3\_2delete.cc }
- funzione di deallocazione esplicita:  
  { ALLOC\_DINAMICA/responsabilita4.cc }

# Struttura di un array bidimensionale (statico)

```
int MAT[2][3] = {{1,2,3},{4,5,6}};
```



Esempio di allocazione statica (da esempi su “MATRICI”):  
{ ALLOC\_DINAMICA/matrix\_sta.cc }

# Allocazione dinamica di un array multidimensionale

- In C++ non è possibile allocare direttamente un array multi-dimensionale in modo dinamico

⇒ array multidimensionali e puntatori sono oggetti **incompatibili**.

```
int * MAT1 = new int[2][3]; // ERRORE
int ** MAT2 = new int[2][3]; // ERRORE
```

- “`new int[2][3]`” restituisce l’indirizzo di 2 oggetti consecutivi di tipo “`int[3]`”, incompatibili sia con “`int *`” che con “`int **`”

Esempio di cui sopra, espanso:

```
{ ALLOC_DINAMICA/matrix_din_err.cc }
```

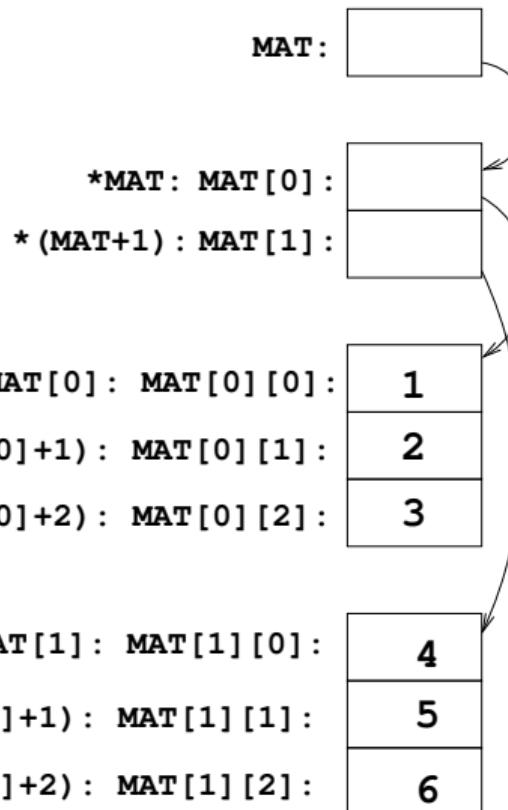
# Array dinamici multidimensionali

- In C++ si possono definire array dinamici multidimensionali come **array dinamici di array dinamici** ...
- Tipo base: puntatore di puntatore ...
- Gli operatori “[ ]” funzionano come nel caso statico
  - `MAT[ i ]` equivalente a `* (MAT+i)`,
  - `MAT[ i ] [ j ]` equivalente a `* ( (* (MAT+i)) +j )`,
- L'allocazione richiede un ciclo (o più)

```
int ** M; // puntatore a puntatori a int
M = new int *[dim1]; // array dinamico di puntatori
for (int i=0; i<dim1; i++)
 M[i] = new int [dim2]; // allocazione di ciascun array
```

# Struttura di un array bidimensionale (dinamico)

```
int dim1=2, dim2=3;
int ** MAT = new int *[dim1];
for (int i=0;i<dim1;i++)
 M[i] = new int[dim2];
```



Esempio di cui sopra, espanso:  
{ ALLOC\_DINAMICA/matrix\_din.cc }

# Esempi: allocazione e gestione di matrici dinamiche

- Operazioni matriciali su matrice dinamica:  
    { ALLOC\_DINAMICA/matrix.cc }
- idem, con il nuovo tipo “matrix” (uso di **typedef**):  
    { ALLOC\_DINAMICA/matrix\_typedef.cc }
- come sopra, con unica funzione di allocazione matrice:  
    { ALLOC\_DINAMICA/matrix\_v2\_typedef.cc }

# Esempi: deallocazione di matrici dinamiche

Suggerimento 1: aprire shell con comando “top” attivo

Suggerimento 2: su bash provare ad eseguire con (ulimit -v 5000000; ./a.out)

- **allocazione senza deallocazione di matrici dinamiche:**

{ ALLOC\_DINAMICA/matrix2.cc }

- .. con deallocazione mediante **delete []**:

{ ALLOC\_DINAMICA/matrix3.cc }

- ... con deallocazione completa di matrici:

{ ALLOC\_DINAMICA/matrix4.cc }

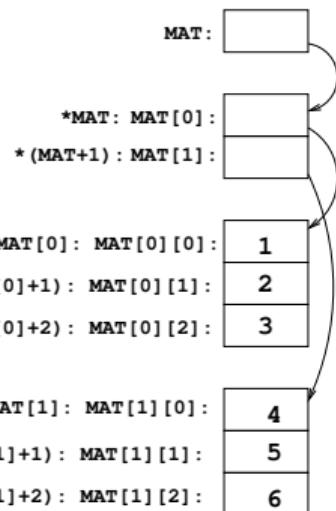
⇒ Anche la deallocazione richiede un ciclo (o più)

# Array bidimensionali dinamici vs. statici

Sebbene concettualmente simili, gli array multidimensionali dinamici e statici sono sintatticamente oggetti diversi e non compatibili  
(uno è un “`int **`”, l’altro un “`int * const *`”)

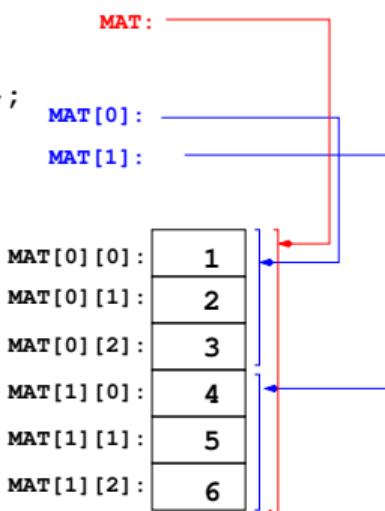
## Array bidimensionale dinamico

```
int dim1=2, dim2=3;
int ** MAT = new int *[dim1];
for (int i=0;i<dim1;i++)
 M[i] = new int[dim2];
```



## Array bidimensionale statico

```
int MAT[2][3] = ;
{ {1,2,3}, {4,5,6} };
```



## Array bidimensionali dinamici vs. statici II

```
void print_matrix_dim(float ** a, ...) {...}
void print_matrix_sta(float a[][][d2a], ...) {...}

float A[d1a][d2a] = {{1,2,3},{4,5,6}};
float ** B;
B = read_matrix(d1b, d2b);
// B = A; // errore
// print_matrix_dim (A, d1a, d2a); // errore
print_matrix_dim(B, d1b, d2b);
print_matrix_sta(A, d1a, d2a);
// print_matrix_sta(B, d1b, d2b); // errore
```

Esempio di cui sopra, espanso:

{ ALLOC\_DINAMICA/matrix\_stavsdin.cc }

# Corso “Programmazione 1”

## Capitolo 09: Le Strutture

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it  
**Stefano Berlato** - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 16 novembre 2020

# Terms of Use and Copyright

## USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

## SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

## COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

# Outline

- 1 Le Strutture
- 2 Operazioni su Strutture
- 3 Strutture Ricorsive
- 4 Array ordinati di Strutture

- Una struttura è una **collezione ordinata di elementi non omogenei**
  - Gli elementi sono detti **membri** o **campi**
  - Ciascun campo ha uno specifico **tipo**, **nome** e **valore**
- Permette di definire **nuovi tipi** di **oggetti aggregati**
  - La struttura può essere utilizzata come un oggetto unico
  - I campi possono essere utilizzati singolarmente
- Ciascun campo può essere a sua volta un tipo struttura

# Definizione di un tipo **struct**

- Viene definito un nuovo **tipo** aggregato

- Sintassi:

```
struct new_struct_id {
 tipo1 campo1;
 ...
 tipoN campoN;
};
new_struct_id var_id;
```

- Esempio:

```
struct complex { // definizione del tipo "complex"
 double re; // campo "reale"
 double im; // campo "immaginario"
};
```

```
complex c,c1; // definizione di variabili di tipo "complex"
```

# Alcuni esempi di strutture annidate

```
struct data {
 int giorno, mese, anno;
};

struct persona { // struttura annidata
 char nome[25], cognome[25];
 char comune_nascita[25];
 data data_nascita;
 enum { F, M } sesso;
};

struct studente { // struttura ulteriormente annidata
 persona generalita;
 char matricola[10];
 int anno_iscrizione;
};
```

# Inizializzazione di variabili di tipo **struct**

- Una variabile di tipo **struct** viene inizializzata con liste ordinate dei valori dei campi rispettivi
  - Devono combaciare per ordine e tipo
  - Eventuali valori mancanti vengono iniziati allo zero del tipo

```
struct data {
 int giorno, mese, anno;
};
struct persona {
 char nome[25], cognome[25];
 char comune_nascita[25];
 data data_nascita;
 enum { F, M } sesso;
};
persona x = {"Paolo", "Rossi", "Trento",
 {21,10,1980}, M };
```

# Accesso ai campi di una **struct**

- Se `s` è una **struct** e `field` è un identificatore di un suo campo, allora `s.field` denota il campo della **struct**
  - `s.field` è un'espressione dotata di indirizzo!  
⇒ può essere letta con `>>`, assegnata, passata per riferimento, ecc.
- Se `ps` è un **puntatore** ad una struttura avente `field` come campo, allora è possibile scrivere `ps->field` al posto di `(*ps).field`
  - zucchero sintattico
  - uso molto frequente

## Esempio

```
struct complex { double re, im; };
complex c; complex *pc = &c;
c.re = 2.5; pc->im = 3;
cin >> c.re >> c.im;
swap(c.re,c.im);
```

# Esempi

- Operazioni di base sui membri di **struct** :  
  { STRUCT/struct.cc }
- ... con inizializzazione:  
  { STRUCT/struct1.cc }

# Assegnazione di Strutture

- A differenza degli array, l'assegnazione tra **struct** è definita
- L'assegnazione di **struct** avviene **per valore**
  - Vengono copiati tutti i valori dei membri
  - Se un campo è un array statico, viene copiato per intero!

⇒ La copia di **struct** può essere computazionalmente onerosa!

```
persona x, y = {"Paolo", "Rossi", "Trento",
 {21,10,1980}, M };
```

```
x=y; // vengono copiate tutte le stringhe!
```

# Passaggio di Strutture a Funzioni

- A differenza degli array, le strutture:
  - Possono essere passate per valore ad una funzione
  - Possono essere restituire da una funzione tramite **return**
- Entrambe le operazioni comportano una **copia** dei valori dei membri (array compresi!)
- $\Rightarrow$  **Entrambe le operazioni possono essere computazionalmente onerose!**
- $\Rightarrow$  Quando possibile, è preferibile utilizzare passaggio per riferimento (con **const**)

```
void stampa_persona (persona p) {...}
void stampa_personal (const persona & p) {...}

persona x,y = {"Paolo", "Rossi", "Trento",
 {21,10,1980}, M };
stampa_persona(y); // viene fatta una copia
stampa_personal(y); // non viene fatta alcuna copia
```

- Assegnazione e passaggio di **struct** :  
{ STRUCT/struct3.cc }

# Assegnazione di array statici tramite **struct**

Per gestire gli **array statici** in modo che possano essere copiati, si può “incapsulare” un tipo array come membro di una **struct**

```
struct int_array { int ia[3]; };
int_array sa, sb;
sa.ia[0]=1;
sa.ia[1]=2;
sa.ia[2]=3;
sb = sa; // l'array viene copiato!
```

- Esempio di cui sopra, esteso:  
{ STRUCT/struct\_array.cc }
- ... con inizializzazione:  
{ STRUCT/struct\_array1.cc }

# Assegnazione di array dinamici tramite **struct**

Nel caso di **array dinamici**, viene copiato solo il puntatore

```
struct int_array { int * ia; };
sa.ia = new int[3];
sb = sa; // viene copiato il puntatore,
 // sa.ia e sb.ia sono lo stesso array!
```

- Esempio di cui sopra, esteso:  
{ STRUCT/struct\_arraypunt.cc }

# Strutture ricorsive

- La seguente definizione non è lecita:

```
struct S {
 int value;
 S next; //definizione circolare!
};
```

- La seguente definizione è lecita:

```
struct S {
 int value;
 S *next;
};
```

- Ogni puntatore occupa lo stesso spazio di memoria indipendentemente dal suo tipo.
- Molto importante per strutture dati dinamiche!

# Strutture mutualmente ricorsive

- La seguente definizione non è lecita:

```
struct S1
{ int value;
 S2 *next; }; // S2 ancora indefinito
struct S2
{ int value;
 S1 *next; };
```

- La seguente definizione è lecita:

```
struct S2; // dichiarazione di S2
struct S1
{ int value;
 S2 *next; }; // Ok!
struct S2 // definizione di S2
{ int value;
 S1 *next; };
```

# Esempi

- **struct** ricorsiva, non corretta:

```
{ STRUCT/rec_struct_err.cc }
```

- **struct** ricorsiva, corretta:

```
{ STRUCT/rec_struct.cc }
```

- **struct** mutualmente ricorsive, non corrette:

```
{ STRUCT/mutrec_struct_err.cc }
```

- **struct** mutualmente ricorsive, corrette:

```
{ STRUCT/mutrec_struct.cc }
```

# Uso di Array Ordinati di Strutture

- È frequente il dover gestire **archivi ordinati** di oggetti complessi (ES: persone, articoli, libri, ecc. )
  - Elemento base dei **sistemi informativi** (ES: archivi, inventari, rubriche, anagrafi, ecc.)
  - Ordinamento usa qualche campo specifico (**chiave**)
- Tipicamente utilizzate strutture di dati dinamiche ad hoc (ES: alberi di ricerca binaria)
- Esempio di archivio semplificato: **array ordinato di persone**

# Esempio 1: Array Ordinato di Persone

- Array ordinato di strutture “persona”:

```
persona persone [NmaxPers];
```

- Ordinamento e ricerca usano `strcmp` sul campo `cognome`:

```
if (strcmp(p[i].cognome, cognome) < 0) ...
```

- Array ordinato di `struct`, bubblesort, ricerca binaria:

```
{ STRUCT/persone.cc }
```

- Problema: ogni swap effettua 3 copie tra `struct`!

⇒ molto inefficiente!

## Esempio 2: Array Ordinato di Puntatori a Persone

- Array ordinato di **puntatori** a strutture “persona”:  
`persona * persone [NmaxPers];`
- Ordinamento e ricerca usano **strcmp** sul campo **cognome**:  
`if (strcmp(p[i]->cognome, cognome) < 0) ...`
- Array ordinato di puntatori a **struct**, bubblesort, ricerca binaria:  
`{ STRUCT/persone2.cc }`
- Importante: **ogni swap effettua 3 copie tra puntatori**  
 $\Rightarrow$  efficiente!

## Es. 3: Doppio Array Ordinato di Puntatori a Persone

- Richiesta: poter effettuare ricerca sia per nome che per cognome.
- Idea: 2 array di puntatori a strutture “persona”, ordinati rispettivamente per nome e cognome

```
persona * nomi [NmaxPers];
persona * cognomi [NmaxPers];
```

- Ordinamento e ricerca usano `strcmp` sul campo cognome e nome rispettivamente:  
`if (strcmp(p[i]->cognome, cognome) < 0) ...`  
`if (strcmp(p[i]->nome, nome) < 0) ...`
- Array ordinato di puntatori a **struct**, bubblesort, ricerca binaria:  
`{ STRUCT/persone3.cc }`
- Importante: Le **struct** sono condivise tra i due array, ogni swap effettua 3 copie tra puntatori  
⇒ efficiente!

# Esercizi proposti

**Vedere file `ESERCIZI_PROPOSTI.txt`**

# Corso “Programmazione 1”

## Capitolo 10: Strutturazione di un Programma

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it  
**Stefano Berlato** - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 18 novembre 2020

# Terms of Use and Copyright

## USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

## SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

## COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

# Outline

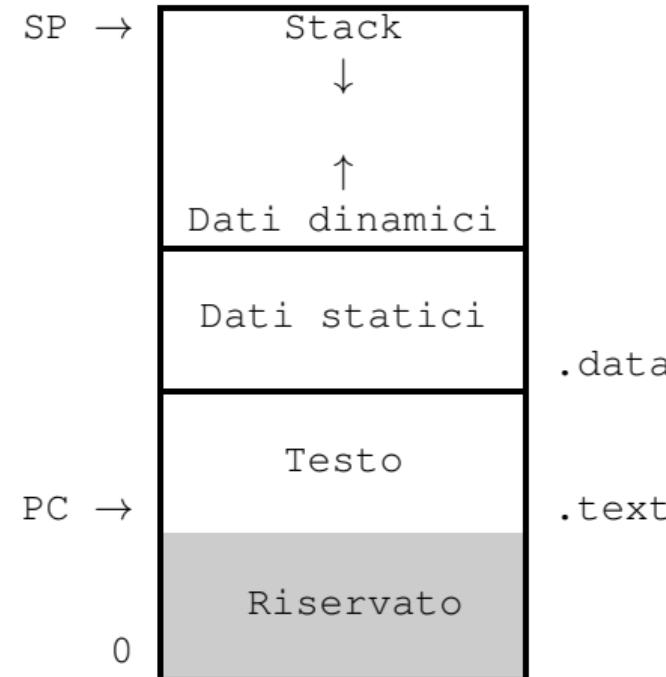
1 Modello di Gestione della Memoria in un Programma

2 Programmazione su File Multipli

# Modello di gestione della memoria per un programma (ripasso)

Area di memoria allocata ad un'esecuzione di un programma:

- **Area programmi e costanti:** destinata a contenere le **istruzioni** (in linguaggio macchina) del programma
- **Area dati statici:** destinata a contenere variabili **globali** o **allocate staticamente** e le **costanti** del programma.
- **Area heap:** destinata a contenere le variabili dinamiche (di dimensioni non prevedibili a tempo di compilazione) del programma.
- **Area stack:** destinata a contenere le **variabili locali** e i **parametri formali** delle funzioni del programma.



# Scope, Visibilità e Durata di una definizione

La definizione di un oggetto (variabile, costante, tipo, funzione) ha tre caratteristiche:

- Scope o ambito
- Visibilità
- Durata

# Scope di una definizione

È la porzione di codice in cui è attiva una definizione

- **Scope globale**: definizione attiva a livello di file
- **Scope locale**: definizione attiva localmente
  - ad una funzione
  - ad un blocco di istruzioni

```
const float pi=3.1415; // scope globale
int x; // scope globale
int f(int a, double x); // scope locale
{ int c; // scope locale
 ...
}
int main()
{ char pi; // scope locale
 ...
}
```

# Visibilità di una definizione

Stabilisce quali oggetti definiti sono visibili da un punto del codice

- In caso di funzioni:
  - una definizione globale è visibile a livello locale, ma non viceversa
  - una definizione omonima locale maschera una definizione globale
- in caso di blocchi annidati
  - una definizione esterna è visibile a livello interno, ma non viceversa
  - una definizione omonima interna maschera una definizione esterna

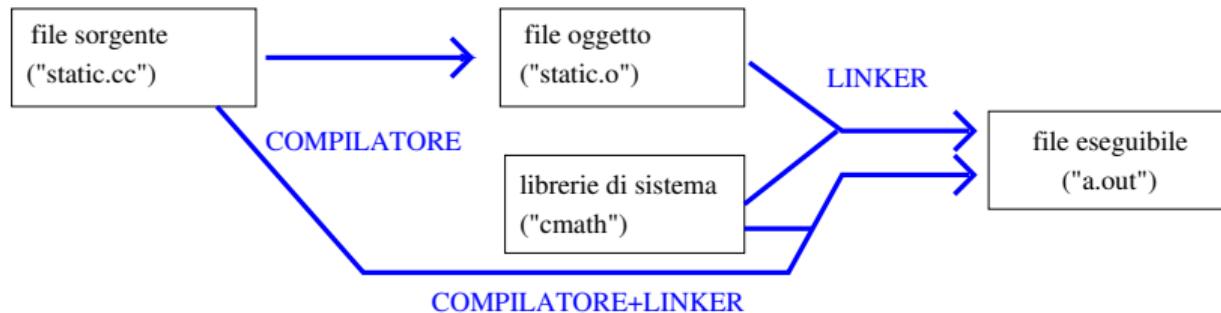
```
1. const float pi=3.1415; // sono visibili:
2. int x; // pi(1)
3. int f(int a, double x)
4. { int c; // pi(1), a(3), x(3)
5. ...} // pi(1), a(3), x(3), c(4)
6. int main()
7. { char pi; // x(2), f(3),
8. ... } // x(2), f(3), pi(7)
```

# Durata di una definizione

Stabilisce il periodo in cui l'oggetto definito rimane allocato in memoria

- **Globale o Statico**: oggetto globale o dichiarato con **static**
  - dura fino alla fine dell'esecuzione del programma
  - memorizzato nell'**area dati statici**
- **Locale o automatico**: oggetti locali a un blocco o funzione
  - hanno sempre scope locale
  - durano solo il periodo di tempo necessario ad eseguire il blocco o funzione in cui sono definiti
  - memorizzato nell'**area stack**
- **Dinamico**: oggetti allocati e deallocati da **new/delete**
  - hanno scope determinato dalla raggiungibilità dell'indirizzo.
  - durata gestita dalle chiamate a **new** e **delete**: durano fino alla deallocazione con **delete** o alla fine del programma
  - dimensione non prevedibile a tempo di compilazione
  - memorizzati nell'**area heap**

# Compilazione (un unico file) - recap



- File sorgente tradotto in un file oggetto dal compilatore
  - Es: `g++ -c static.cc`
  - Imp: file oggetto illeggibile ad un essere umano e non stampabile!
- File oggetto collegato (linked) a librerie di sistema dal linker, generano un file eseguibile (default `a.out`, opzione `-o <nome>`)
  - Es: `g++ static.o`
  - Es: `g++ static.o -o static`
  - File incomprensibili agli umani, ma eseguibili da una macchina
- Compilazione e linking possibile in un'unica istruzione
  - Es: `g++ static.cc`

# Lo specificatore **static**

Lo specificatore **static** applicato ad una **variabile locale** forza la durata della variabile oltre la durata della funzione dove è definita

- la variabile è allocata nell'**area dati statici**
  - un'eventuale inizializzazione nella dichiarazione viene eseguita una sola volta all'atto dell'inizializzazione del programma
  - il valore della variabile viene “ricordato” da una chiamata all'altra della funzione
  - potenziali sorgenti di errori  $\Rightarrow$  **vanno usate con molta cautela!**
- 
- Esempio di uso di variabile **static** locale:  
  { PROG\_FILE\_MULTIPLI/static.cc }
  - Esempio di uso di variabile **static** locale anziché globale:  
  { PROG\_FILE\_MULTIPLI/fibonacci.cc }

# Lo specificatore **static** II

## Nota

Lo specificatore **static** applicato ad un **oggetto di scope globale** (es. funzioni, variabili, costanti globali) ha l'effetto di restringere la visibilità dell'oggetto al solo file in cui occorre la definizione

- concetto molto importante nella **programmazione su più file**  
(vedi slide successive)

# Lo specificatore **extern**

Lo specificatore **extern** consente di **dichiarare** e poi utilizzare in un file oggetti (globali) che sono **definiti** in un altro file

- consente al compilatore di
  - verificare la coerenza delle espressioni contenenti tali oggetti
  - stabilire le dimensioni delle corrispondenti aree di memoria
- l'oggetto dichiarato deve essere definito in un altro file
- il linker associa gli oggetti dichiarati alle corrispondenti definizioni

## • Esempio di uso di **extern**:

{ PROG\_FILE\_MULTIPLI/extern.cc                            }  
                                                                  }  
                                                                  PROG\_FILE\_MULTIPLI/extern\_main.cc

# Dichiarazione vs Definizione

## Nota

- Un oggetto può essere **dichiarato** quante volte si vuole ...  
... ma può essere **definito** una volta sola
- Un oggetto dichiarato più volte deve essere dichiarato sempre nello stesso modo
- Ogni **definizione** è anche una implicita **dichiarazione**

# Programmazione su file multipli

I programmi possono essere organizzati su file multipli

- Organizzazione **modulare**
  - Ogni file raggruppa un **insieme di funzionalità** (modulo)
  - Compilazione separata di ogni modulo e linking file oggetto
- Moltissimi vantaggi:
  - Rapidità di compilazione
  - Programmazione condivisa tra più persone/team
  - Riutilizzo del codice in più programmi
  - Produzione di librerie
  - Utilizzo di librerie prodotte da altri
  - Mantenibilità del codice
  - ...

# Organizzazione di un programma su file multipli

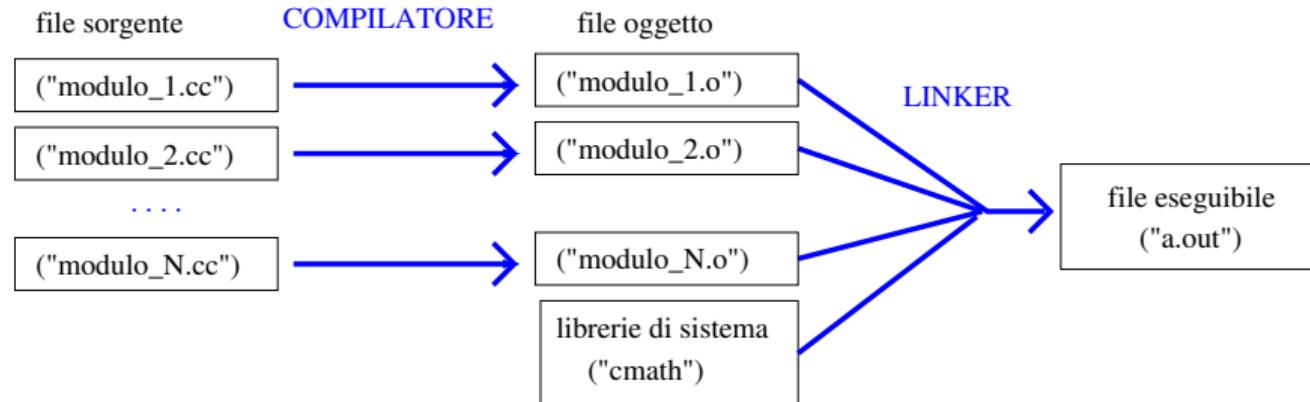
- Un programma viene usualmente ripartito su  $2N + 1$  file
  - Un file `file_main.cc` contenente la definizione della funzione `main()`
  - $N$  coppie di file `modulo_i.h` e `modulo_i.cc`, una per ogni modulo `modulo_i` che si vuole realizzare separatamente
  - Tutti i file “`.cc`” devono venire compilati e i risultanti file oggetto linkati

# Organizzazione di un programma su file multipli - II

- Ogni file “.cc” che utilizzi funzioni/tipi/costanti/variabili globali definiti in `modulo_i` deve inizialmente contenere l’istruzione: `#include "modulo_i.h"`
- `modulo_i.h` contiene gli **header** delle funzioni di `modulo_i`
  - può contenere definizioni di tipo, costanti, variabili globali, ecc.
  - per evitare di venire caricato ripetutamente deve utilizzare **guardie di compilazione**:

```
#ifndef MODULO_I_H // Esiste anche direttiva
#define MODULO_I_H // #pragma once
...
...
#endif // Non e' standard e non e' robusto
 // come approccio (e.g. link,
 // copie di file, ...)
```
- `modulo_i.cc` contiene le **definizioni** delle funzioni di `modulo_i`
  - contenere l’istruzione: `#include "modulo_i.h"`
  - può contenere **funzioni ausiliarie inaccessibili all'esterno (static)**

# Compilazione (su più files) - recap



- File sorgente tradotti nei rispettivi **file oggetto** uno alla volta
- File oggetto collegato (linked) a librerie di sistema dal **linker**, generano un **file eseguibile (default a.out)**
  - **Es:** `g++ modulo_1.o modulo_2.o ... modulo_N.o`
- Compilazione e linking possibile in un unica istruzione
  - **Es:** `g++ modulo_1.cc modulo_2.cc ... modulo_N.cc`

## Schema: Programma su un solo file

disney.cc

```
#include <iostream>
int pluto() {...}; // funzione ausiliaria
 // non chiamata dal main()
void topolino() { ... };
void paperino() { ... };

int main() {
...
switch(scelta) {
 case 1: topolino(); break;
 case 2: paperino(); break;
...
}
```

## Schema: Programma su file multipli - I

disney.h

```
#ifndef DISNEY_H
#define DISNEY_H
void topolino();
void paperino();
#endif
```

disney.cc

```
#include <iostream>
#include "disney.h"
static int pluto() {...}; // funzione ausiliaria non chiamata
 // dal main().
 // Header non in disney.h!
void topolino() { ... };
void paperino() { ... };
```

## Schema: Programma su file multipli - II

disney\_main.cc

```
#include <iostream>
#include "disney.h"

int main() {
 ...
 switch(scelta) {
 case 1: topolino(); break;
 case 2: paperino(); break;
 ...
 }
}
```

# Organizzazione di un programma su file multipli: Esempi

- Programma su un solo file:

{ PROG\_FILE\_MULTIPLI/matrix\_v2\_typedef.cc }

- Stesso programma organizzato in file multipli:

{ PROG\_FILE\_MULTIPLI/matrix.h  
PROG\_FILE\_MULTIPLI/matrix.cc  
PROG\_FILE\_MULTIPLI/matrix\_main.cc }

# Bad programming practices

## Cosa non fare e come non organizzare un programma su file multipli

- Includere i file “.h” (e compilare tutti i file, “.h” inclusi)
- NON includere i file “.cc” (e compilare solo il file chiamante) !!!
  - impedirebbe uso multiplo (non hanno guardie di compilazione )
  - romperebbe la modularità
  - impedirebbe la compilazione separata
- $\Rightarrow$  naturale sorgente di errori, **evitare tassativamente**

disney\_main.cc

```
#include "disney.cc" // NNOO!!!!
```

...

disney\_main.cc

```
#include "disney.h" // SI!
```

...

# Esercizi proposti

**Vedere file ESERCIZI\_PROPOSTI.txt**

# Corso “Programmazione 1”

## Capitolo 11: Liste Concatenate

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it  
**Stefano Berlato** - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 23 novembre 2020

# Terms of Use and Copyright

## USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

## SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

## COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored by Marco Roveri.

# Introduzione

- Quando dobbiamo scandire una collezione di oggetti in modo sequenziale e non sequenziale, un modo conveniente per rappresentarli è quello di organizzare gli oggetti in un array.
- Esempi:
  - `{1, 2, -3, 5, -10}` è una sequenza di interi.
  - `{'a', 'd', '1', 'F'}` è una sequenza di caratteri.
- Una soluzione alternativa all'uso di array per rappresentare collezioni di oggetti quando l'accesso non sequenziale non è un requisito, consiste nell'uso delle cosiddette **liste concatenate**.
- In una lista concatenata i vari elementi che compongono la sequenza di dati sono rappresentati in zone di memoria che possono anche essere distanti fra loro (al contrario degli array, in cui gli elementi sono consecutivi).
- In una lista concatenata, ogni elemento contiene informazioni necessarie per accedere all'elemento successivo.

# Liste concatenate

- Una lista concatenata è un insieme di oggetti, dove ogni oggetto è inserito in un nodo contenente anche un link ad un altro nodo.

```
// Lista concatenata di interi
struct nodo {
 int dato;
 nodo * next;
}
```



## Liste concatenate (II)

- Per il nodo finale si possono adottare diverse convenzioni:
  - Punta ad un **link nullo** che non punta a nessun nodo (e.g. `<val> = NULL`)



- Punta ad un **nodo fittizio** che non contiene alcun nodo (e.g. `<val> = (nodo *) 300`)



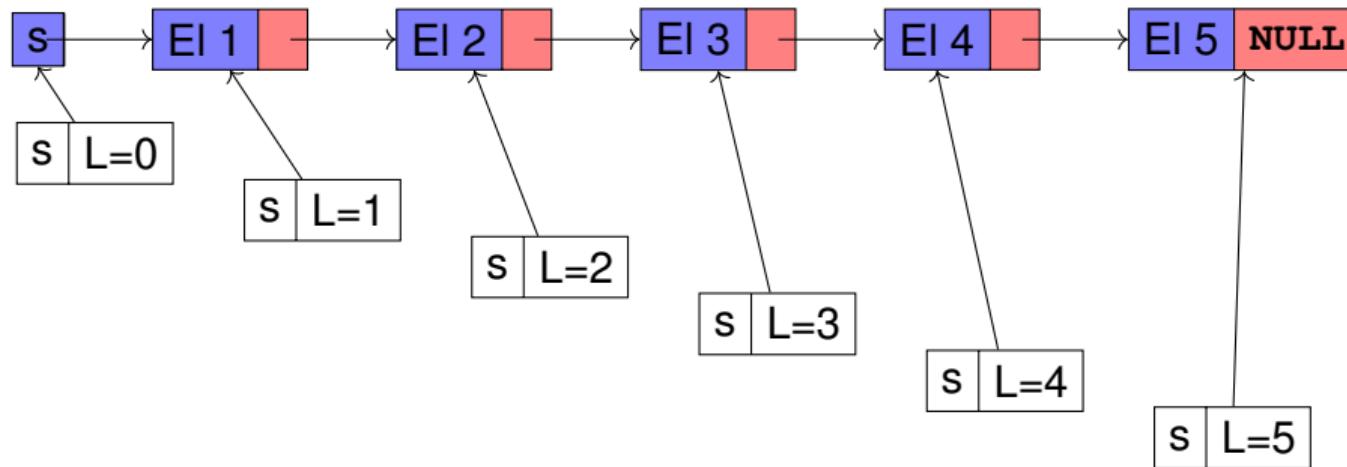
- Punta indietro al primo della lista, creando una **lista circolare**



# Operazioni su liste concatenate

- Calcolo della lunghezza di una lista concatenata.
- Inserimento di un elemento in una lista concatenata, aumentando la lunghezza di una unità.
- Cancellazione di un elemento in una lista concatenata, diminuendo la lunghezza di una unità.
- Rovesciamento di una lista.
- Append: concatenazione di due liste.

# Calcolo lunghezza di una lista



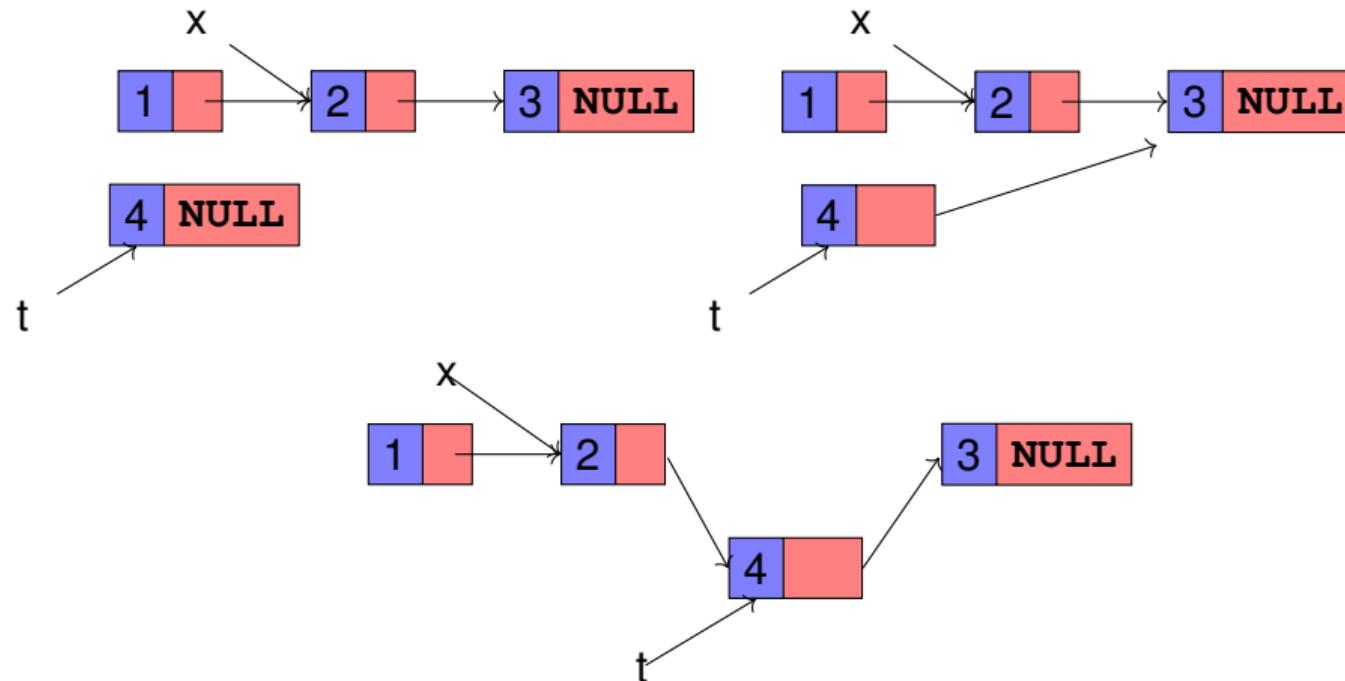
## Calcolo lunghezza di una lista (II)

```
// lista con terminatore NULL
int length (nodo * s) {
 int l = 0;
 for(; s != NULL; s = s->next) l++;
 return l;
}
```

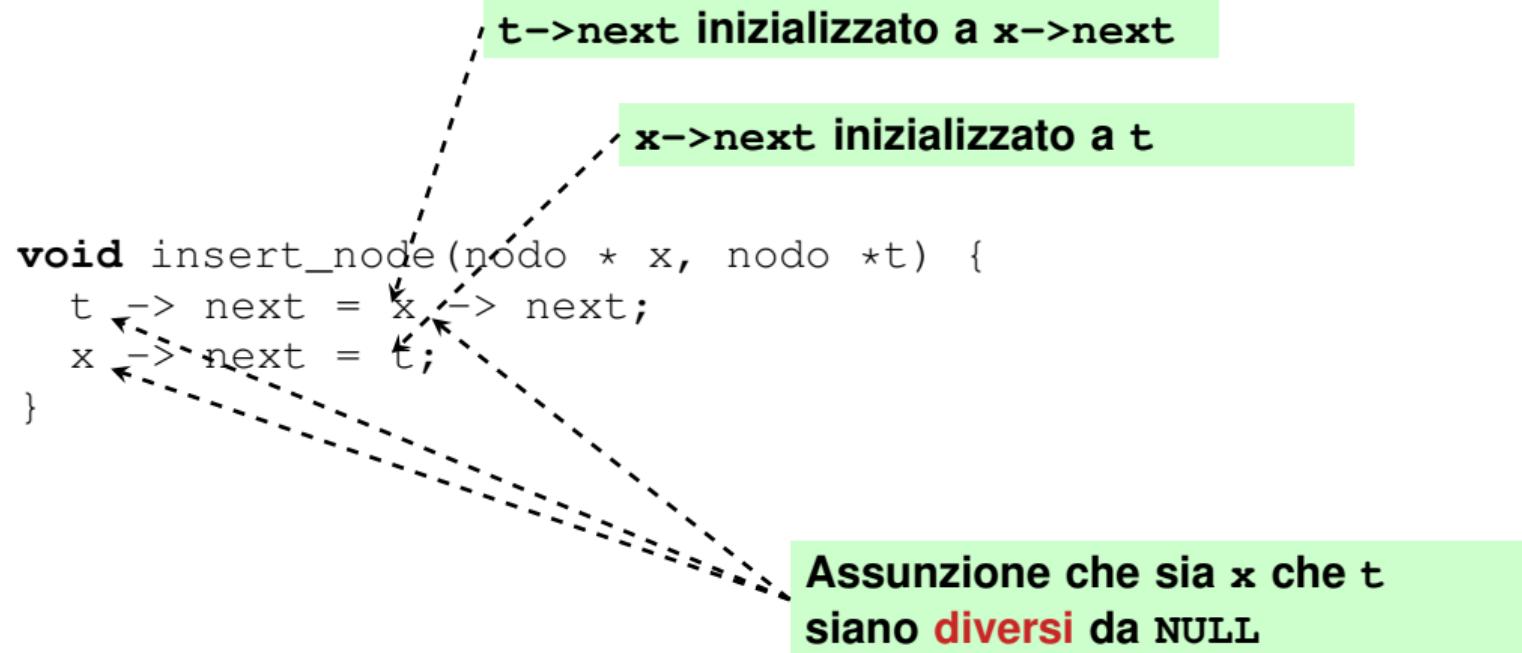
```
// lista circolare, x primo elemento
int length (nodo * s, nodo * x) {
 int l = 0;
 if (s != NULL) {
 l = 1;
 for(s = s->next; s != x; s = s->next) l++;
 }
 return l;
}
```

# Inserimento di un elemento

- Per inserire un nodo  $t$  in una lista concatenata nella posizione successiva a quella occupata da un dato nodo  $x$ , poniamo  $t \rightarrow \text{next} = x \rightarrow \text{next}$ , e quindi  $x \rightarrow \text{next} = t$ .



## Inserimento di un elemento (II)



## Inserimento di un elemento (II)

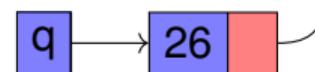
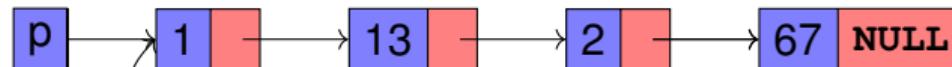
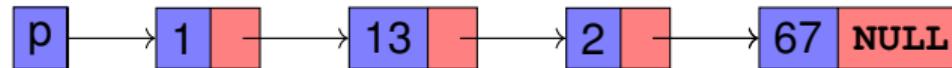
```
int main () {
 nodo * x = new nodo; ← Allocazione di un nodo per
 cout << "Inserire_numero:_"; memorizzare primo elemento
 cin >> x->dato;
 x->next = NULL;
 for (int i = 0; i < 10; i++){
 nodo * t = new nodo; ← Allocazione di un nuovo nodo per
 cout << "Inserire_un_numero:_"; memorizzare i-esimo elemento
 cin >> t->dato;
 t->next = NULL; ← Campo next di t inizializzato a
 insert_node(x, t); ← NULL
 }
 for (nodo *s = x; s != NULL; s=s->next)
 cout << "valore_=_" << s->dato << endl;
}
```

Manca deallocazione della lista!!!

Variabile temporanea per scorrere lista

# Inserimento di un elemento in testa

- Si vuole inserire un nuovo elemento 26 in testa alla lista!



```
node *q = new nodo;
q->dato = 26;
```

```
q->next = p;
```

```
p = q;
```

## Inserimento di un elemento in testa (II)

- Se dobbiamo inserire un elemento in testa della lista:
  - `void insert_first(nodo * s, int v);`
  - `void insert_first(nodo * &s, int v);`
  - `nodo * insert_first(nodo * s, int v);`
- La seconda e la terza sono le uniche possibili dichiarazioni corrette.
  - Nella seconda side effect sull'argomento.
  - Nella terza lista nuova ritornata dalla funzione.

## Inserimento di un elemento in testa (III)

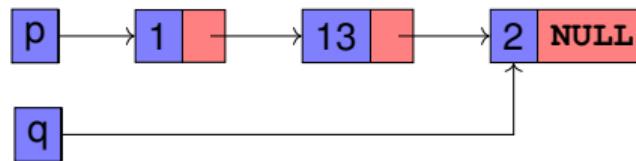
```
void insert_first(node * s, int v) {
 node * n = new node;
 n->dato = v;
 n->next = s;
 s = n;
}

void insert_first(node*&s, int v) {
 node * n = new node;
 n->dato = v;
 n->next = s;
 s = n;
}

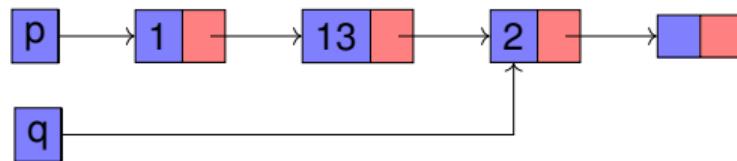
node * insert_first(node*s, int v) {
 node * n = new node;
 n->dato = v;
 n->next = s;
 return n;
}
```

# Inserimento di un elemento in coda

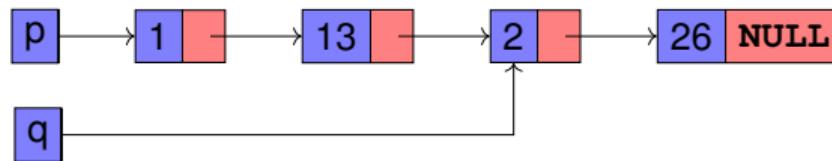
- Si vuole inserire un nuovo elemento 26 in coda alla lista!



```
node *q = p;
while (q->next != NULL)
 q = q->next;
```



```
q->next = new node;
```



```
q->next->dato = 26;
q->next->next = NULL;
```

Nota:

Questo ragionamento è corretto solo se la lista non è vuota!!!

## Inserimento di un elemento in coda (II)

```
void insert_last(nodo * & p, int n) {
 nodo * r = new nodo; ----->
 r->dato = n;
 r->next = NULL;
 if (p != NULL) {
 nodo * q = p;
 while (q->next != NULL) { ----->
 q = q->next; ----->
 }
 q->next = r; ----->
 }
 else {
 p = r; ----->
 }
}
```

Allocazione del nuovo nodo

Se la lista non è vuota, cerco in q il puntatore all'ultimo elemento

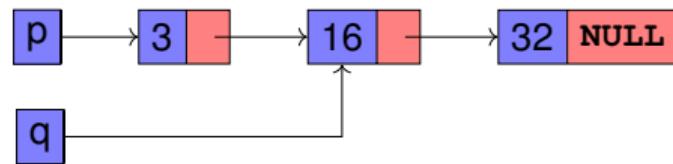
q è garantito essere diverso da NULL

Memorizzo in q->next il nuovo nodo r allocato in precedenza

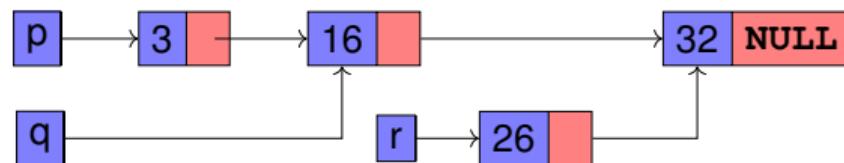
Se la lista è vuota, p punta al nuovo nodo allocato: p è passato per riferimento

# Inserimento di un elemento in lista ordinata

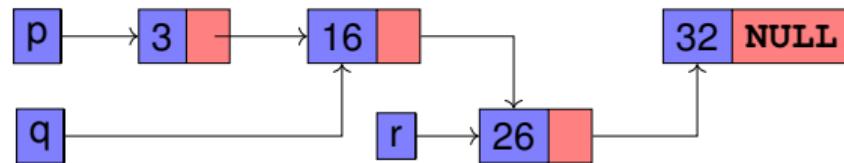
- Si vuole inserire un nuovo elemento 26 in una lista ordinata mantenendo ordinamento!



```
node *q = p;
while (q->next->dato <= 26)
 q = q->next;
```



```
node *r = new node;
r->dato = 26;
r->next = q->next;
```



```
q->next = r;
```

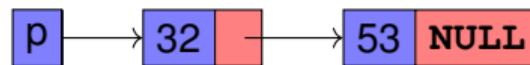
Nota:

Devono essere considerati alcuni casi limite!!!

# Inserimento di un elemento in lista ordinata (II)

- **Primo caso limite:** inserimento in testa

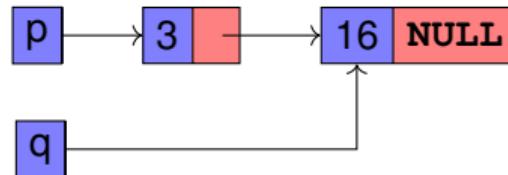
- perchè la lista è vuota:  $p == \text{NULL}$
- perchè tutti gli elementi hanno un valore maggiore



```
if ((p == NULL) || (p->dato >= 26))
 insert_first(p, 26);
```

- **Secondo caso limite:** inserimento in coda

- perchè tutti gli elementi hanno un valore minore



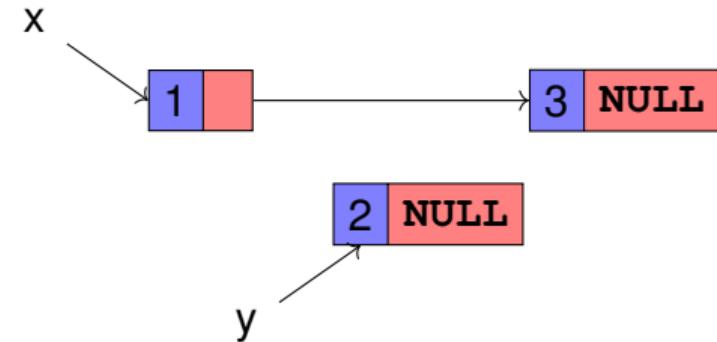
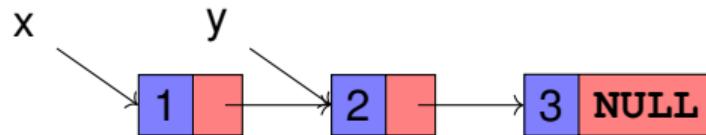
```
node *q = p;
while (q->next != NULL &&
 q->next->dato <= 26)
 q = q->next;
```

## Inserimento di un elemento in lista ordinata (III)

```
void insert_order(nodo * &p, int inform) {
 if ((p==NULL) || (p->dato >= inform)) {
 insert_first(p, inform);
 }
 else {
 nodo* q=p;
 while ((q->next != NULL) &&
 (q->next->dato <= inform)) {
 q=q->next;
 }
 nodo* r = new nodo;
 r->dato = inform;
 r->next = q->next;
 q->next = r;
 }
}
```

## Rimozione di un elemento

- Per rimuovere un nodo  $y$  in una lista concatenata nella posizione successiva a quella occupata da un dato nodo  $x$ , cambiamo  $x \rightarrow \text{next}$  a  $y \rightarrow \text{next}$ .



## Rimozione di un elemento (II)

```
node * remove_element (node **x) {
 node * y = x->next;
 x->next = y->next;
 y->next = NULL;
 return y;
}
```

y inizializzato a x->next

x->next punta a y->next

Assunzione che x, x->next (e quindi anche y) siano diversi da NULL

y ritornato per ad esempio essere deallocated!

## Rimozione di un elemento (II)

```
int main () {
 nodo * x = new nodo;
 cout << "Inserire_numero:_";
 cin >> x->dato
 x->next = NULL;
 for (int i = 0; i < 10; i++) {
 nodo * t = new nodo;
 cout << "Inserire_un_numero:_";
 cin >> t->dato
 t->next = NULL;
 insert_node(x, t);
 }
 for (int i = 0; i < 10; i++) {
 nodo * t = remove_element(x);
 cout << "valore_=_" << t->dato << endl;
 delete t; <----- Deallocazione del nodo rimosso
 }
 delete x; <----- Deallocazione del nodo x iniziale
}
```

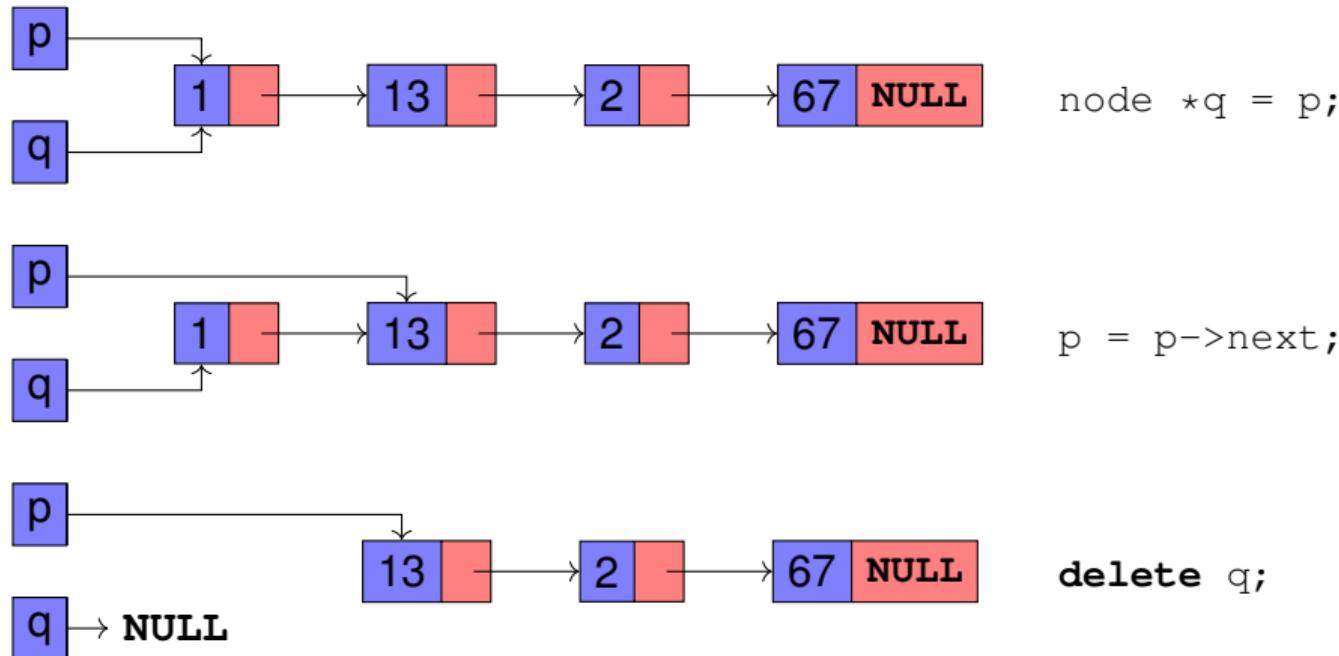
Variabile temporanea per memorizzare nodo rimosso!

Deallocazione del nodo rimosso

Deallocazione del nodo x iniziale

# Rimozione di un elemento in testa

- Si vuole eliminare primo elemento della lista!



## Rimozione di un elemento in testa (II)

- Se dobbiamo rimuovere un elemento in testa della lista:

- **void** remove\_first(nodo \* s);
- **void** remove\_first(nodo \* &s);
- nodo \* remove\_first(nodo \* s);

- La seconda e la terza sono le uniche possibili dichiarazioni corrette.

- Nella seconda side effect sull'argomento.
- Nella terza lista nuova ritornata dalla funzione.

Nota:

Il nodo rimosso deve essere deallocated!!!

## Rimozione di un elemento in testa (III)

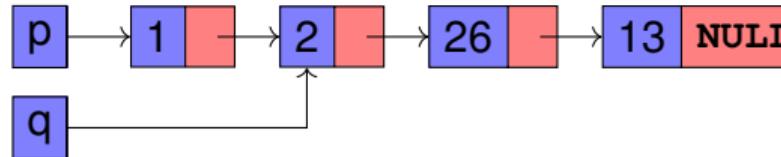
```
void remove_first(nodo * s) {
 nodo * n = s;
 if (s != NULL) {
 s = s->next;
 delete n;
 }
}

void remove_first(nodo * & s) {
 nodo * n = s;
 if (s != NULL) {
 s = s->next;
 delete n;
 }
}

nodo * remove_first(nodo * s)
{
 nodo * n = s;
 if (s != NULL) {
 s = s->next;
 delete n;
 }
 return s;
} // attenzione a come invocata
```

# Rimozione di un elemento particolare

- Si vuole cercare un nodo che contiene nel campo `dato` un valore ed eliminarlo!

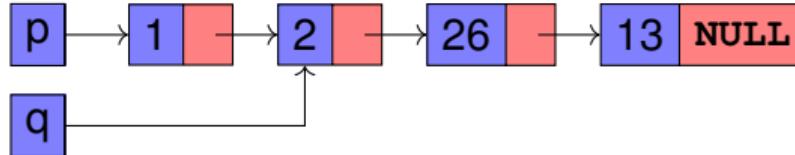


`nodo* q=p;`

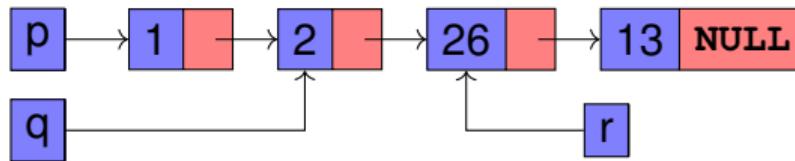
```
while (q->next !=NULL) {
 if (q->next->dato==26) {
 }
 q=q->next;
}
```

# Rimozione di un elemento particolare

- Si vuole cercare un nodo che contiene nel campo `dato` un valore ed eliminarlo!



```
nodo* q=p;
```

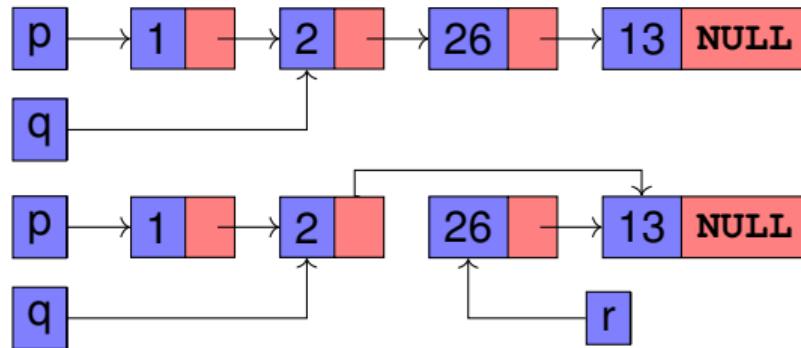


```
while (q->next !=NULL) {
 if (q->next->dato==26) {
 node *r = q->next;
```

```
 }
 q=q->next;
}
```

# Rimozione di un elemento particolare

- Si vuole cercare un nodo che contiene nel campo `dato` un valore ed eliminarlo!

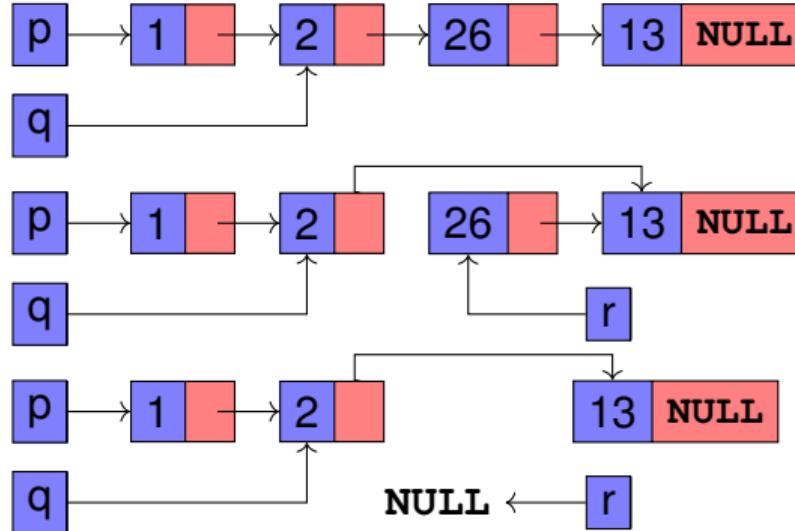


```
nodo* q=p;
```

```
while (q->next !=NULL) {
 if (q->next->dato==26) {
 node *r = q->next;
 q->next = q->next->next;
 }
 q=q->next;
}
```

# Rimozione di un elemento particolare

- Si vuole cercare un nodo che contiene nel campo `dato` un dato valore ed eliminarlo!

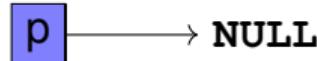


```
nodo* q=p;
```

```
while (q->next !=NULL) {
 if (q->next->dato==26) {
 node *r = q->next;
 q->next = q->next->next;
 delete r;
 }
 q=q->next;
}
```

# Rimozione di un elemento particolare

- Si vuole cercare un nodo che contiene nel campo `dato` un valore ed eliminarlo!



```
if (p != NULL) {
 nodo* q=p;

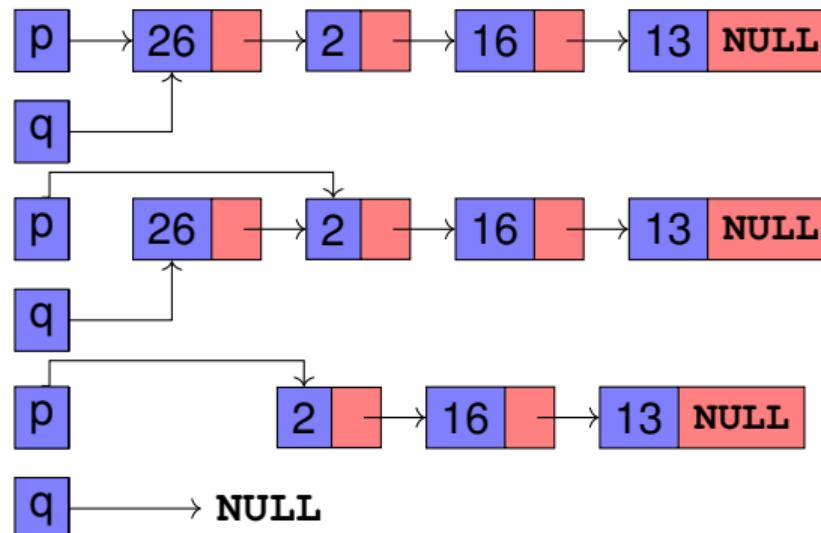
 while (q->next !=NULL) {
 if (q->next->dato==26) {
 node *r = q->next;
 q->next = q->next->next;
 delete r;
 }
 q=q->next;
 }
}
```

Nota:

Primo caso limite: Lista vuota!!!

# Rimozione di un elemento particolare

- Si vuole cercare un nodo che contiene nel campo `dato` un valore ed eliminarlo!



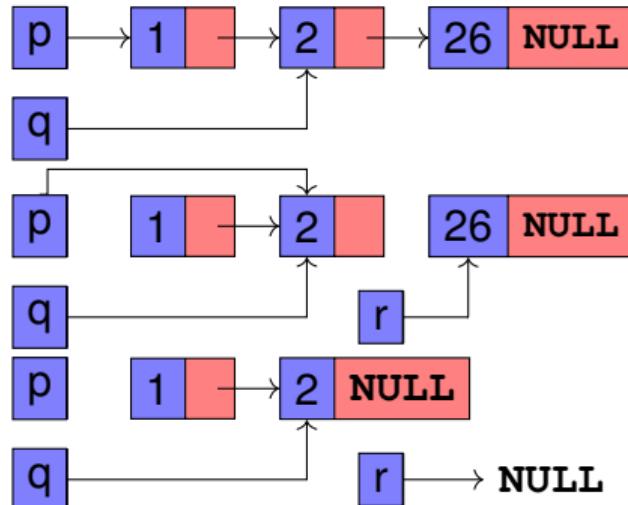
```
if (p != NULL) {
 nodo* q=p;
 if (p->dato == 26) {
 p = p->next; delete q;
 }
 while (q->next !=NULL) {
 if (q->next->dato==26) {
 node *r = q->next;
 q->next = q->next->next;
 delete r;
 }
 q=q->next;
 }
}
```

Nota:

Secondo caso limite: primo nodo da levare!!!

# Rimozione di un elemento particolare

- Si vuole cercare un nodo che contiene nel campo `dato` un valore ed eliminarlo!



```
if (p != NULL) {
 nodo* q=p;
 if (p->dato == 26) {
 p = p->next; delete q;
 }
 while (q->next !=NULL) {
 if (q->next->dato==26) {
 node *r = q->next;
 q->next = q->next->next;
 delete r;
 return;
 }
 q=q->next;
 }
}
```

Nota:

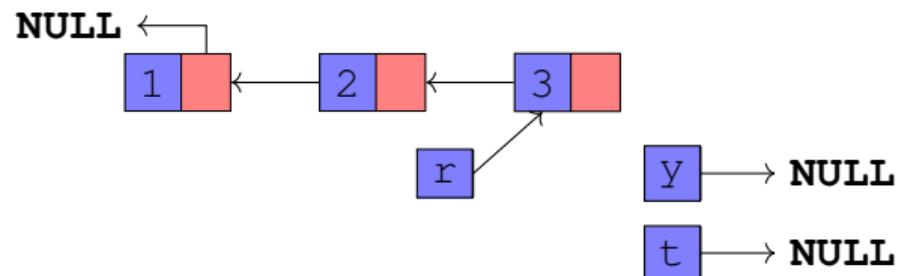
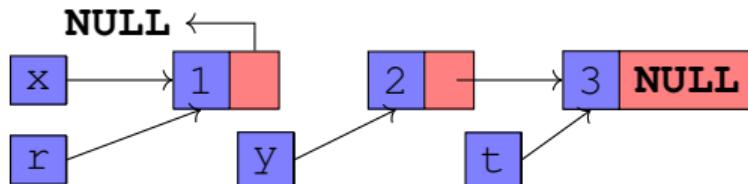
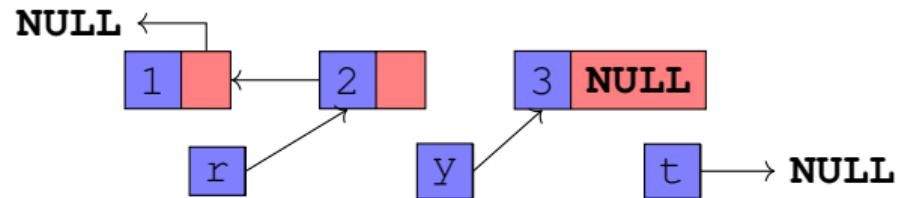
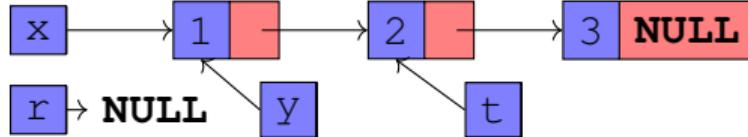
Terzo caso limite: ultimo nodo da levare!!!

## Rimozione di un elemento particolare (II)

```
void search_remove(nodo* &p, int val) {
 if (p != NULL) {
 nodo* q = p;
 if (q->dato == val) {
 p = p->next;
 delete q;
 }
 else {
 while (q->next != NULL) {
 if (q->next->dato == val) {
 nodo* r = q->next;
 q->next = q->next->next;
 delete r;
 return;
 }
 if (q->next != NULL) {
 q=q->next;
 }
 }
 }
 }
}
```

# Rovesciamento di una lista concatenata

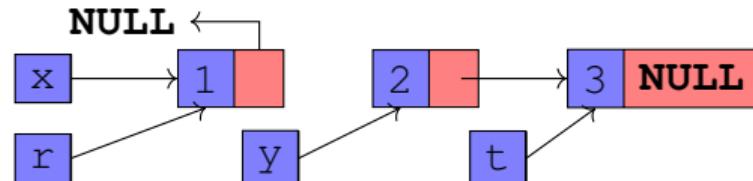
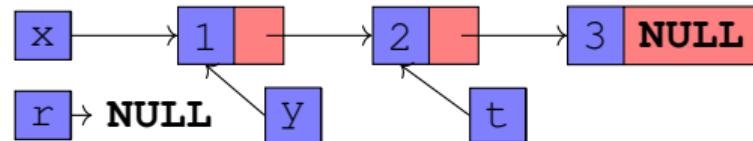
- La funzione di rovesciamento inverte i link di una lista concatenata:
  - restituisce un puntatore al nodo finale che a sua volta punta al penultimo e così via.
  - Il link del primo elemento della lista è posto a **NULL**.



# Rovesciamento di una lista concatenata (II)

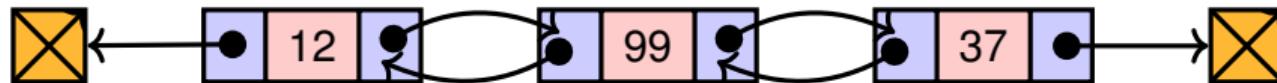
- La funzione di rovesciamento inverte i link di una lista concatenata:
  - restituisce un puntatore al nodo finale che a sua volta punta al penultimo e così via.
  - Il link del primo elemento della lista è posto a **NULL**.

```
node * reverse(node * x) {
 node * t;
 node * y = x;
 node * r = NULL;
 while (y != NULL) {
 t = y->next;
 y->next = r;
 r = y;
 y = t;
 }
 return r;
}
```



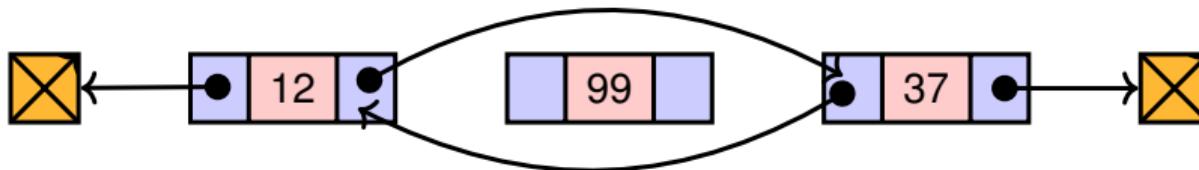
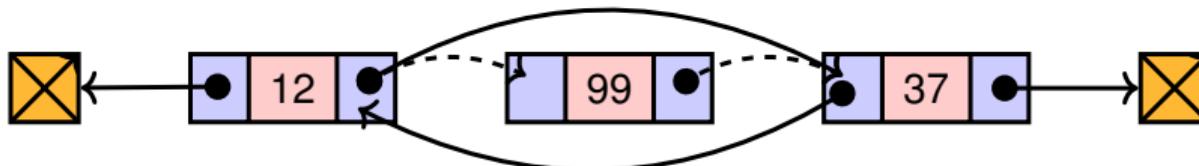
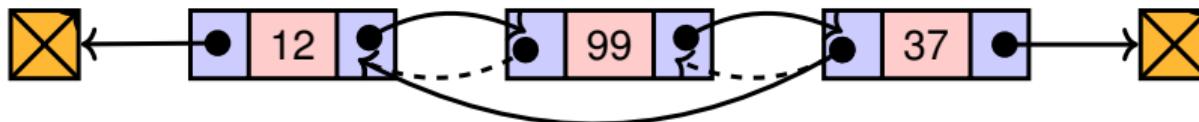
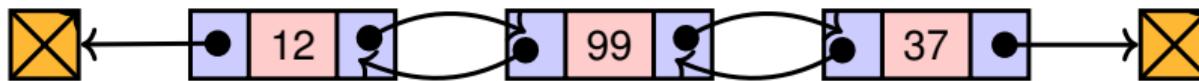
# Liste doppiamente concatenate

- Sono una estensione della definizione delle liste concatenate
  - Differiscono per la presenza di un ulteriore puntatore al nodo che lo precede



```
struct node {
 int n;
 node * prev;
 node * next;
};
```

# Rimozione di un nodo in una lista doppiamente concatenata



# Inserimento di un nodo in una lista doppiamente concatenata

- Soluzione 1:

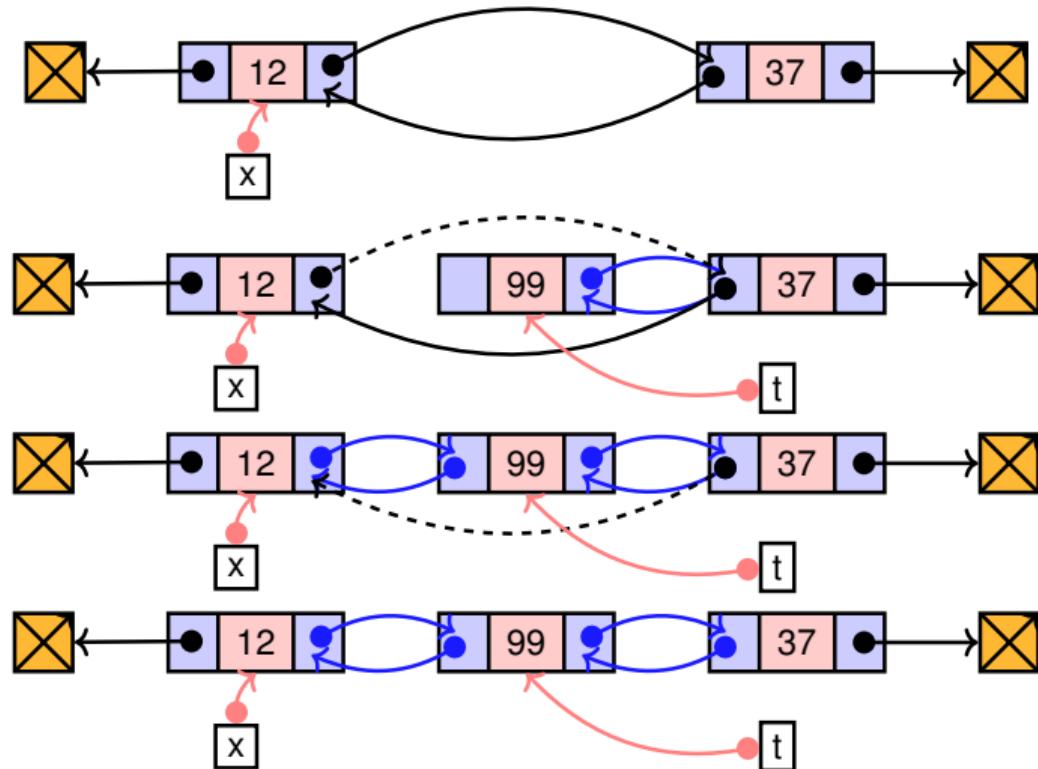
```
node * remove(node * t) {
 t->next->prev = t->prev;
 t->prev->next = t->next;
 t->next = t->prev = NULL;
 return t;
}
```

- Soluzione 2:

```
void remove(node * t) {
 t->next->prev = t->prev;
 t->prev->next = t->next;

 delete t;
}
```

# Inserimento di un nodo in una lista doppiamente concatenata



## Inserimento di un nodo in una lista doppiamente concatenata

```
void insert_node(node * x, node * t) {
 t->next = x->next;
 t->next->prev = t;
 t->prev = x;
 x->next = t;
}
```

# Esercizi proposti

**Vedere file `ESERCIZI_PROPOSTI.txt`**

# Corso “Programmazione 1”

## Capitolo 12: Strutture Dati Astratte

Docente: **Marco Roveri** - marco.roveri@unitn.it

Esercitatori: **Giovanni De Toni** - giovanni.detoni@unitn.it  
**Stefano Berlato** - stefano.berlato-1@unitn.it

C.D.L.: **Informatica (INF)**

A.A.: **2020-2021**

Luogo: **DISI, Università di Trento**

URL: <https://sites.google.com/view/marco-roveri/teaching/pgm1-2021>

Ultimo aggiornamento: 26 novembre 2020

# Terms of Use and Copyright

## USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2020-2021.

## SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

## COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

# Outline

- 1 Tipo di Dato Astratto
- 2 Strutture Dati Astratte Importanti
  - Le Pile (Realizzate Tramite Array)
  - Le Code (Realizzate Tramite Array)
  - Le Pile (Realizzate Tramite Liste Concatenate)
  - Le Code (Realizzate Tramite Liste Concatenate)
  - Gli alberi Binari (Realizzati Tramite Grafi)
  - Gli alberi Binari (Realizzati Tramite Array)
- 3 Esempi
  - Calcolatrice RPN
  - Coda a Priorità
  - Rubrica
  - Rubrica Doppia
  - Calcolatrice Standard

# Tipo di Dato Astratto/Abstract Data Type

Un tipo di dato astratto (TDA)/abstract data type (ADT) è un insieme di valori e di operazioni definite su di essi in modo indipendente dalla loro implementazione

- Per definire un tipo di dato astratto occorre specificare:
  - i dati immagazzinati
  - le operazioni supportate
  - le eventuali condizioni di errore associate alle operazioni
- Per lo stesso TDA si possono avere più implementazioni
  - diversa implementazione, diverse caratteristiche computazionali (efficienza, uso di memoria, ecc.)
  - stessa interfaccia (stessi header di funzioni, riportati in un file .h)  
    ⇒ implementazioni interscambiabili in un programma
- È spesso desiderabile nascondere l'implementazione di un TDA (information hiding): solo i file .h e .o disponibili

N.B.: La nozione di TDA è la base della programmazione ad oggetti.

## Esempio di Tipo di Dato Astratto

- Consideriamo la definizione di un tipo di dato astratto che rappresenta un punto nello spazio cartesiano  $X \times Y$ .
- Le operazioni che vogliamo effettuare su un punto (indipendentemente da come viene implementato) sono:
  - Crea un nuovo punto.
  - Ritorna la coordinata x (y) rispettivamente come double.
  - Assegna la coordinata x (y) rispettivamente.
  - Confronta due punti per vedere se sono uguali o diversi.
  - Stampa le coordinate di un punto.
  - Calcola la distanza tra due punti.
  - Somma due punti.
  - Verifica se tre punti stanno su una retta.

## Esempio di Tipo di Dato Astratto - II

punto.h

```
// versione 1 // // versione 2
struct Point { // struct Point {
 double x; // double coord[2];
 double y; // };
};

// Definizione dei metodi dell'ADT Point
Point PointInit(void);
Point PointInit(const double x, const double y);
double Point_GetX(const Point & p);
double Point_GetY(const Point & p);
void Point_SetX(Point & p, double x);
void Point_SetY(Point & p, double y);
bool Point_Equal(const Point & P1, const Point & P2);
void Point_Print(const Point & P, const char * n);
double Point_GetDistance(const Point & P1, const Point & P2);
Point Point_Sum(const Point & P1, const Point & P2);
bool Point_Aligned(const Point & P1, const Point & P2, const Point & P3);
```

## Esempio di Tipo di Dato Astratto - III

main.cc

```
#include <iostream>
using namespace std;
#include "point.h"

int main() {
 double t;
 Point P2, P1, P3;
 P1 = PointInit(5.0, 5.0);
 Point_Print(P1, "Coordinate del Punto P1");
 cout << "Inserire coordinate di un Punto P2" << endl << "X = ";
 cin >> t;
 Point_SetX(P2, t);
 cout << "Y = "; cin >> t;
 Point_SetY(P2, t);
 cout << "La distanza tra P1 e P2 e' : "
 << Point_GetDistance(P1, P2) << endl;
```

## Esempio di Tipo di Dato Astratto - III

main.cc (cont)

```
if (Point_Equal(P1, P2)) {
 P3 = Point_Sum(P1, P2);
}
else {
 P3 = PointInit(1.0, 1.0);
}
Point_Print(P3, "Coordinate_del_Punto_P3");
if (Point_Aligned(P1, P2, P3)) {
 cout << "I_tre_punti_risiedono_su_una_rettta" << endl;
}
else {
 cout << "I_tre_punti_non_risiedono_su_una_rettta" << endl;
}
```

# Esempio di Tipo di Dato Astratto - III

point.cc

```
Point PointInit() {
 Point r = {0.0, 0.0};
 return r;
}

Point PointInit(const double x,
 const double y) {
 Point r = {x, y};
 return r;
}
```

versione 1

point.cc

```
Point PointInit() {
 Point r;
 r.coord[0] = 0.0;
 r.coord[1] = 0.0;
 return r;
}
```

versione 2

```
Point PointInit(const double x,
 const double y) {
 Point r;
 r.coord[0] = x;
 r.coord[1] = y;
 return r;
}
```

# Esempio di Tipo di Dato Astratto - IV

point.cc (cont)

```
// Ritorna la coordinate X e Y di P
double Point_GetX(const Point & p) {
 return p.x;
}
double Point_GetY(const Point & p) {
 return p.y;
}

// Assegna le coordinate X e Y di P
void Point_SetX(Point & p,
 const double x) {
 p.x = x;
}
void Point_SetY(Point & p,
 const double y) {
 p.y = y;
}
```

versione 1

point.cc (cont)

```
// Ritorna la coordinate X e Y di P
double Point_GetX(const Point & p) {
 return p.coord[0];
}
double Point_GetY(const Point & p) {
 return p.coord[1];
}

// Assegna le coordinate X e Y di P
void Point_SetX(Point & p,
 const double x) {
 p.coord[0] = x;
}
void Point_SetY(Point & p,
 const double y) {
 p.coord[1] = y;
}
```

versione 2

# Esempio di Tipo di Dato Astratto - V

## point.cc (cont)

indipendente dalla versione

```
// Predicato per controllare se due Punti sono uguali
bool Point_Equal(const Point & P1, const Point & P2) {
 return ((Point_GetX(P1) == Point_GetX(P2)) &&
 (Point_GetY(P1) == Point_GetY(P2)));
}

// Stampa coordinate di un punto P inserendo
// la stringa n prima della stampa delle coordinate
void Point_Print(const Point & P, const char * n) {
 cout << n << endl;
 cout << ".X=" << Point_GetX(P) << endl;
 cout << ".Y=" << Point_GetY(P) << endl;
}

// calcola la distanza tra due punti
double Point_GetDistance(const Point & P1, const Point & P2) {
 double dx = (Point_GetX(P1) - Point_GetX(P2));
 double dy = (Point_GetY(P1) - Point_GetY(P2));
 return sqrt(dx * dx + dy * dy);
}
```

# Esempio di Tipo di Dato Astratto - V

point.cc (cont)

indipendente dalla versione

```
// Costruisci il punto risultante dalla somma delle
// rispettive coordinate di due punti P1 e P2
Point Point_Sum(const Point & P1, const Point & P2) {
 return PointInit(Point_GetX(P1) + Point_GetX(P2),
 Point_GetY(P1) + Point_GetY(P2));
}

bool Point_Aligned(const Point & P1, const Point & P2,
 const Point & P3) {
 return ((Point_GetY(P1) - Point_GetY(P2)) *
 (Point_GetX(P1) - Point_GetX(P3))) ==
 ((Point_GetY(P1) - Point_GetY(P3)) *
 (Point_GetX(P1) - Point_GetX(P2)));
}
```

- **TDA Point:**

$$\left. \begin{array}{l} \text{TDA/point.h} \\ \text{TDA/point.cc} \\ \text{TDA/point_main.cc} \end{array} \right\}$$

# Esempi Molto Importanti di Tipi di Dato Astratto

- Le [Pile \(Stack\)](#)
- Le [Code \(Queue\)](#)
- Gli [Alberi \(Tree\)](#)

# Le Pile (Stack)

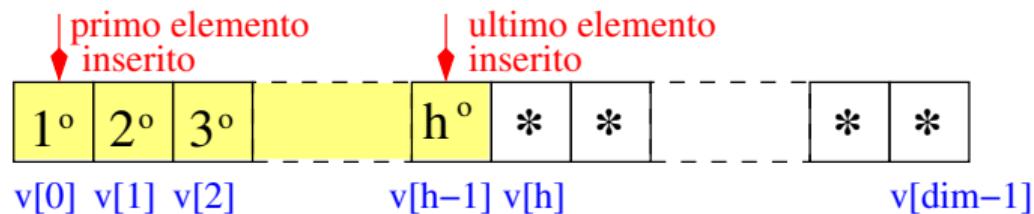
- Una **pila** è una collezione di dati omogenei (e.g., puntatori a struct) in cui gli elementi sono gestiti in modo **LIFO (Last In First Out)**
  - Viene visualizzato/estratto l'elemento inserito più recentemente
  - Es: una scatola alta e stretta contenente documenti
- Operazioni tipiche definite su una pila di oggetti di tipo  $T$ :
  - `init()`/`deinit()`: inizializza/deinizializza la pila
  - `push(T)`: inserisce elemento sulla pila; fallisce se piena
  - `pop()`: estraе l'ultimo elemento inserito (senza visualizzarlo); fallisce se vuota
  - `top(T &)`: ritorna l'ultimo elemento inserito (senza estrarrelo); fallisce se vuota
- Varianti:
  - `pop()` e `top(T &)` fuse in un'unica operazione `pop(T &)`
  - talvolta disponibili anche `print()`
  - [ `deinit()` non sempre presente]

# Le Pile (Stack) II

## Nota importante

In tutte le possibili implementazioni di una pila, le operazioni `push(T)`, `pop()`, `top(T &)` devono richiedere un numero costante di passi computazionali, indipendente dal numero di elementi contenuti nella pila!

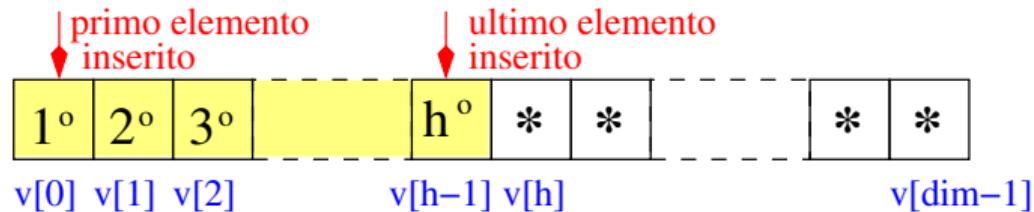
# Implementazione di una pila mediante array



- **Dati:** un intero  $h$  e un array  $v$  di  $\dim$  elementi di tipo  $T$ 
  - $v$  allocato staticamente o dinamicamente
  - $h$  indice del prossimo elemento da inserire (inizialmente 0)  
⇒ numero di elementi contenuti nella pila:  $h$
  - pila vuota:  $h==0$
  - pila piena:  $h==\dim$   
⇒ massimo numero di elementi contenuti nella pila:  $\dim$

N.B.:  $\dim$  elementi sempre allocati.

# Implementazione di una pila mediante array II



- **Funzionalità:**

- `init()`: pone  $h=0$  (alloca  $v$  se allocazione dinamica)
- `push(T)`: inserisce l'elemento in  $v[h]$ , incrementa  $h$
- `pop()`: decrementa  $h$
- `top(T &)`: restituisce  $v[h-1]$
- `deinit()`: dealloca  $v$  se allocazione dinamica

# Esempi su pile di interi

- semplice stack di interi come struct:

{ STACK\_QUEUE\_ARRAY/struct\_stack.h  
STACK\_QUEUE\_ARRAY/struct\_stack.cc  
STACK\_QUEUE\_ARRAY/struct\_stack\_main.cc }

- uso di stack per invertire l'ordine:

{ STACK\_QUEUE\_ARRAY/struct\_stack.h  
STACK\_QUEUE\_ARRAY/struct\_stack.cc  
STACK\_QUEUE\_ARRAY/struct\_reverse\_main.cc }

(struct\_stack.h | .cc stessi del caso precedente)

# Le Code (Queue)

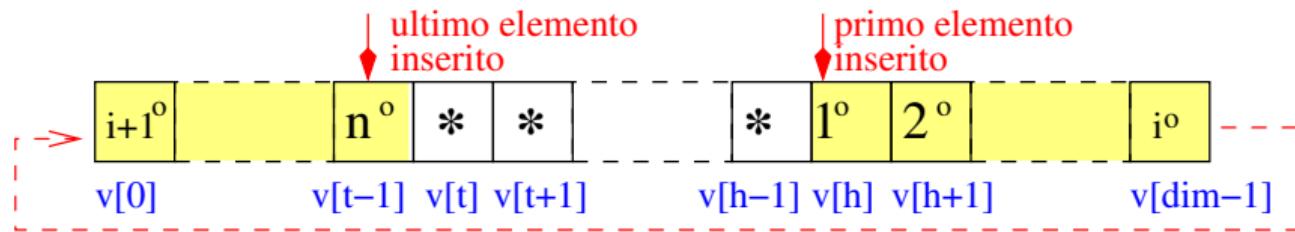
- Una **coda** è una collezione di dati omogenei in cui gli elementi sono gestiti in modo **FIFO (First In First Out)**
  - Viene visualizzato/estratto l'elemento inserito meno recentemente
  - Es: una coda ad uno sportello
- Operazioni tipiche definite su una coda di oggetti di tipo `T`:
  - `init()`/`deinit()`: inizializza/deinizializza la coda
  - `enqueue(T)`: inserisce elemento sulla coda; fallisce se piena
  - `dequeue()`: estraie il primo elemento inserito (senza visualizzarlo); fallisce se vuota
  - `first(T &)`: ritorna il primo elemento inserito (senza estrarrelo); fallisce se vuota
- Varianti:
  - `dequeue()` e `first(T &)` fuse in un'unica operazione `dequeue(T &)`
  - talvolta disponibili anche `print()`
  - [ `deinit()` non sempre presente]

# Le Code (Queue) II

## Nota importante

In tutte le possibili implementazioni di una coda, le operazioni `enqueue (T)` , `dequeue ()` , `first (T &)` **devono richiedere un numero costante di passi computazionali**, indipendente dal numero di elementi contenuti nella coda!

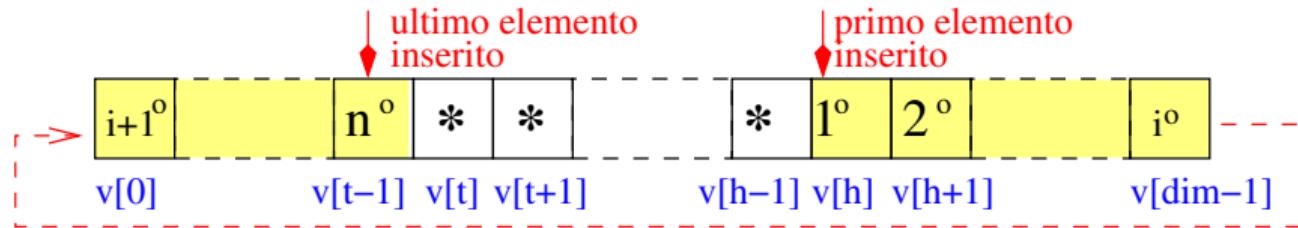
# Implementazione di una coda mediante array



- **Idea:** buffer circolare:  $\text{succ}(i) == (i+1) \% \text{dim}$
  - **Dati:** due interi  $h, t$  e un array  $v$  di  $\text{dim}$  elementi di tipo  $T$ 
    - $v$  allocato staticamente o dinamicamente
    - $h$  indice del più vecchio elemento inserito (inizialmente 0)
    - $t$  indice del prossimo elemento da inserire (inizialmente 0)
- ⇒ num. di elementi contenuti nella coda:  $n = (t >= h ? t-h : t-h+\text{dim})$
- coda vuota:  $t == h$
  - coda piena:  $\text{succ}(t) == h$
- ⇒ massimo numero di elementi contenuti nella coda:  $\text{dim}-1$

N.B.:  $\text{dim}$  elementi sempre allocati.

# Implementazione di una coda mediante array II



## • Funzionalità:

- `init()`: pone  $h=t=0$  (alloca  $v$  se allocazione dinamica)
- `enqueue(T)`: inserisce l'elemento in  $v[t]$ , “incrementa”  $t$  ( $t=\text{succ}(t)$ )
- `dequeue()`: “incrementa”  $h$
- `first(T &)`: restituisce  $v[h]$
- `deinit()`: dealloca  $v$  se allocazione dinamica

# Esempi su code di interi

- semplice coda di interi come struct:

{ STACK\_QUEUE\_ARRAY/struct\_queue.h  
  STACK\_QUEUE\_ARRAY/struct\_queue.cc  
  STACK\_QUEUE\_ARRAY/struct\_queue\_main.cc }

# Esercizi proposti

**Vedere file `ESERCIZI_PROPOSTI.txt`**