

## Programmazione Procedurale

---

---

---

---



INFORMATICA : discipline che studia il trattamento automatico delle informazioni

- ASPECTI TECHNOLOGICI (opinione pubblica) (ICT)
- ASPECTI METODOLOGICI (architettura e ingegneria del software)
- ASPECTI SCIENTIFICI (computer scienze)
  - ↓
  - TEORIA DELLA CALCOLABILITÀ  
(se un problema è risolvibile **INDECIDIBILE**)
  - TEORIA DELLA COMPLESSITÀ  
(se un problema è risolvibile **DECIDIBILE**)

## • AUTOMI E LINGUAGGI

COMPUTER o elaboratore elettronico :insieme di dispositivi elettromeccanici programmabili per l'elaborazione e l'emissione di informazioni sotto forma di numeri, testi, immagini, suoni e filmati

- HARDWARE
- SOFTWARE

PENSIERO COMPUTAZIONALE : contributo culturale dell'informatica nello sociale moderno (def. corretta utidi dispense)

## CENNI storici INFORMATICA

1.2

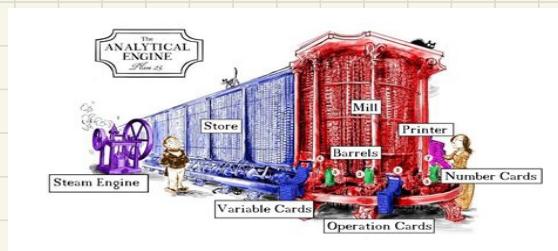
1642 Pascal inventa la macchina meccanica per fare addizioni e sottrazioni



1672 Leibniz "più quelle moltiplicazione e divisione"

1822 Babbage "Differente Engine"

—  
1834 Babbage "Analytical Engine"



programmabilità, architettura unica di soluzioni

1843 Ada Byron Lovelace Traduzione del Principio @ inglese  
Delle Analytical engine  
La prima programmatrice delle storia

1936 Turing Formalizzazione algoritmo

1938 Claude Shannon Teorie dell' informazione  
Elettronica elettronica

1946 Eniac Elettronica General purpose Colossus

1952 Von Neumann IAS computer a programma monoruttino

1955 - 1965 Tecnologie a transistor ( CEP in Italia )

1965 - 1980 Circuiti integrati Elettronica elettronica  
P101 personal computer

1969 Nasce internet

1971 Microprocessore

1980 da poi VLSI

Primi due linguaggi  
Programmazione relazionale  
app. scientifiche app. commerciale  
( FORTRAN ) ( COBOL )  
( ALGOL ) ( C ) ( PASCAL ) -----

PROGRAMMAZIONE A OGGETTI ( SIMULA ) C++, JAVA, C#

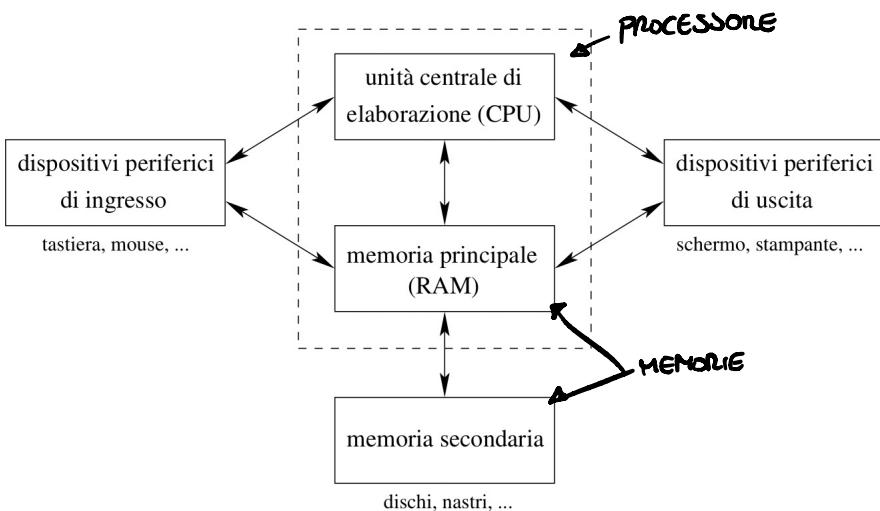
PROGRAMMAZIONE DICHIARATIVA ( LISP ) ( HASKELL )

# ELEMENTI DI ARCHITETTURA DEI ELABORATORI

1.3

(Babbage, Turing e Von Neumann)

## Architettura



Computer e programma memorizzato  $\rightarrow$  - PROGRAMMI (SEQUENZE DI CIFRE BINARIE)  
- DATI

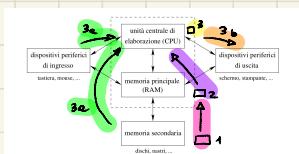
Se si cambia il programma si cambiano le operazioni da fare

$\downarrow$   
CARICATI IN MEMORIA

$\downarrow$   
ELABORAZIONE

PASSI ESECUZIONE PROGRAMMA :

- ① PROGRAMMA RISIEDE NELLA MEMORIA SECONDARIA (NON VOLATILE)
- ② IL PROGRAMMA VIENE TRASFERITO DALLA MEMORIA SECONDARIA A PRIMARIA PER ESSERE ESEGUITO
- ③ IL PROGRAMMA VIENE ESEGUITO DALLA CPU
  - 3a acquisisce i dati di ingresso
  - 3b produce dati di uscita



- PUNTO DI VISTA DEL PROGRAMMA : - RISORSA CHE LO ESEGUE CPU  
 - RISORSA CHE LO CONTIENE MEMORIA  
 - RISORSE CHE ACQUISISCONO E CONDIVIDONO I DATI DISPOST. ING.  
DISPOST. USCIT.

CPU → COORDINA AZIONI DEL COMPUTER (LIV. HARDW.)  
 ESEGUE OP. ARIT. LOGICHE SUI DATI

COMPOSTA DA : - UNITÀ DI CONTROLLO (CU), DETERMINA QUALI OPERAZIONI BISOGNA ESEGUIRE IN ORDINE  
 - UNITÀ ARITMETICO-LOGICA (ALU), ESEGUE LE OPERAZ.  
 ARIT.-LOGICHE SEGNALATE DALLA CU SUI DATI STABILITI IN QUEL COMANDO  
 - INSERIMENTI DI REGISTRI VOLTI A IMMAGAZZINARE  
 ISTRUZIONI, DATI CORRENTI, ISTRUZ. CORR. ECC; SONO PICcole MEMORIE  
 DIRETTAMENTE COLLEGATE ALLA CPU E HANNO UNA VELOCITÀ DI IMMAGAZZ.  
 DATI SIMILI A QUELLA DEI PROCESSORI (CPU)

PROGRAMMA CARICATO IN MEMORIA PRINCIPALE / CICLO FETCH-DECODE-EXECUTE

- ① MEMORIA PRINCIP → (IR) INSTRUCT REGISTER PROSSIMA AZIONE DA ESEGUIRE, INDICATA NEL REGISTRO (PC)
- ② AGGIORNARE IL PC → ISTRUZIONE SUCCESSIVA A QUELLA CARICATA NEL IR
- ③ INTERPRETARE ISTRUZIONE NELL' IR → OP. DA COMP.
- ④ MEMORIA PRINC → (IR) DATI DI CUI HANNO BISOGNO NELL' ISTRUZIONE
- ⑤ ESEGUIRE OPERAZ.
- ⑥ RITORNARE ALL' INIZIO DEL CICLO

SISTEMA IN BASE 2 ADEGUATO PER LA RAPP. DI INFORMAZIONI NELLA MEMORIA DI UN COMPUTER. 0,1 ASSOCIAVOLI A DIVERSE GAMME DI VALORI DI TENSIONE +SV, -SV; DUE POLARIZZAZIONI OPPoste (AMPO MAGNETICO)

- MEMORIA : - SEQUENZE DI ALLOCAZIONI (CELLE), OGUNA HA IL PROPRIO INDIRIZZO
- NOMEGLIO A CELLE  $2^m$
- INDIRIZZI 0 e  $2^m - 1$  ponono un'area nopp. con m cifre bin.
- CONTENUTO delle celle  $\rightarrow$  sequenze di cifre binarie / bit
- numero di bit  $\rightarrow$  dimensione delle celle
- cella di dimensione d può contenere  $2^d$  valori diversi  
comprendenti 0 e  $2^d - 1$

unità di misura byte    1 BYTE = 8 bit

MEMORIA PRINCIPALE : - ACCESSO CASUALE (RAM)

- VOLATILE

MEMORIA SECONDARIA : - ACCESSO SEQUENZIALE

- PERMANENTE

ES: USB, HARD DISK, MASTRI MAGNETICI

GERARCHIE MEMORIA      REGISTRI - MEMORIA PRINC - MEMOR. SEC :

- VELOCITÀ DECRES.
- CAPACITÀ CRESC.
- COSTI DECRES.

DAL PUNTO DI VISTA DEI PROGRAMMI : - MEMORIA SEC. LI CONTIENE TUTTI I PROGRAMMI DA ESEGUIRE ORA

- MEMORIA PRINC. PROGRAMMA DA ESEGUIRE ORA
- REGISTRO ISTRUZIONE DEL PROGRAMMA DA ESEGUIRE ORA

## ELEMENTI SISTEMA OPERATIVO

1° SOFTWARE INSTALLATO È IL SISTEMA OPERATIVO (non sempre)

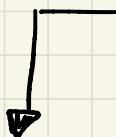
Lo gestisce l'interazione utente - hardware (ESSENZIALE PER COMPRENDERE IL FUNZIONAMENTO DI UN PROGRAMMA)

NEGLI ANNI 90 → ESISTEVA UN UNICO OPERATORE E UN UNICO PROGRAMMA ALLA VOLTA IN ESECUZIONE (MODALITÀ BATCH)

Avvento dei sistemi di elaborazione → MULTIPROGRAMMING, ovvero molti più

programmi possono essere eseguiti e tenuti nello stesso momento

TIME SHARING, in cui più utenti condividono la risorsa hardware così da stabilire un dialogo più diretto tra macchina e uomo mediante la turnazione delle risorse (modalità interattiva)



NASCONO I PRIMI SISTEMI OPERATIVI volti a coordinare le risorse in ambiente multiprogramma - time sharing

OBIETTIVI DEL SISTEMA OPERATIVO : - PRESTAZIONALE deve garantire tempi di esecuzione accettabili: niente sprechi e altro, donde la grande percentuale di hardware ai programmi

- USABILITÀ rendere il sistema operativo un ambiente amichevole, favor incrementare la produttività c'è il gioco di ruoli.

## COMPITI SPECIFICI DI UN SISTEMA OPERATIVO:

- SCHEDULARE I PROGRAMMI Quale viene prima e dopo
- GESTIRE LA MEMORIA PRINCIPALE Quale programma caricare e quale no
- GESTIRE LA MEMORIA SECONDARIA Quale programma è dato prendere dalla memoria
- GESTIONE DISPOSITIVI PERIFERICI In che ordine evadere le risposte di periferici
- TRATTARE I FILE In che ordine e modo conservarli nella memoria secondaria
- PRESERVARE L'INTEGRITÀ Da errori interni o attacchi esterni
- ACQUISIRE COMANDI DAGLI UTENTI

## ELEMENTI DI LINGUAGGI DI PROGRAMMAZIONE E CALCOLATORI

1.5

LA PROGRAMMAZIONE di un computer richiede un linguaggio in cui esprimere i programmi

IL PROCESSORE HA 2 LINGUAGGI : - LINGUAGGIO MACCHINA, Ogni istruzione del sistema oper. è in formato binario (ADDZ, MOVT, DIV, SOTT., TRANSF.DATI, SALTO ISTRUZIONI) È direttamente interpretabile dal processore

Questi due linguaggi non sono a basso livello di astrazione  
e ci manca di avere la stessa architettura ma si possono eseguire sui computer diversi



- LINGUAGGIO ASSEMBLATIVO i valori e indirizzi e gli operatori sono in formato immemoria, omologabili da noi esseri umani, in cambio però questo linguaggio non è comprensibile dal processore per tanto avremmo bisogno di un traduttore detto "assemblatore" per farlo capire al processore

LINGUAGGI DI PROGRAMMAZIONE AD ALTO LIVELLO (Sviluppati dal 1955)

COBOL → GESTIONE  
FORTRAN → SCIENTIFICHE )

- ALTO LIVELLO DI ASTRAZIONE, facilita nella scrittura dei programmi
- INDEPENDENZA DALL'ARCHITETTURA, possono essere eseguiti su tutti i computer
- NECESSITÀ DI TRADURRE IL COMANDO Un comando dato ad alto livello non è un comando a basso livello

IL TRADUTTORE :

- INTERPRETE, traduzione contestuale dell'intuizione (linguaggi dichiarativi, più intuitivi)
- COMPILATORI, traduzione viene effettuata separatamente dall'esecuzione del programma (linguaggi imperativi, più efficienti)

IL COMPILATORE o INTERPRETE produce il codice alla macchina quello che noi vogliamo che lei fa, ma lo scrivere è meno intuitivo dopo il traduttore lo trovi in linguaggio macchina

COMPONENTI DI UN COMPILATORE E RELATIVE FASI DI ANALISI E SINTESI DEL CODICE :

- Analizzatore formale, controllo che i simboli, numeri, identificati sono utilizzate nello congruenti con l'uso del linguaggio
- Analizzatore sintattico, (parser) organizza i termini in una struttura accostandoli con le regole grammaticali del linguaggio controllando se ci sono errori
- Analizzatore semantico, correttezza del significato compiuta
- Generatore di codice, produce il programma in linguaggio assemblativo
- ottimizzazione, ottimizza il codice togliendo parti ripetitive ma senza intaccare la semantica

# METODOLOGIA DI SVILUPPO SOFTWARE (DI BUONA QUALITÀ)

## 1. Specifiche del problema (TESTO)

### 2. Analisi del problema

- dati di ingresso del problema (DATI INGRESSO) RESI
- dati di uscita del problema (DATI USCITA) RISOLUZIONE
- relativi concetti

### 3. Progettazione dell'algoritmo

- scelte di progetto
- piani dell'algoritmo

### 4. Implementazione dell'algoritmo

### DOCUMENTAZIONE

### 5. Testing programma

- Interne
- Esterne (MANIFESTAZIONI)

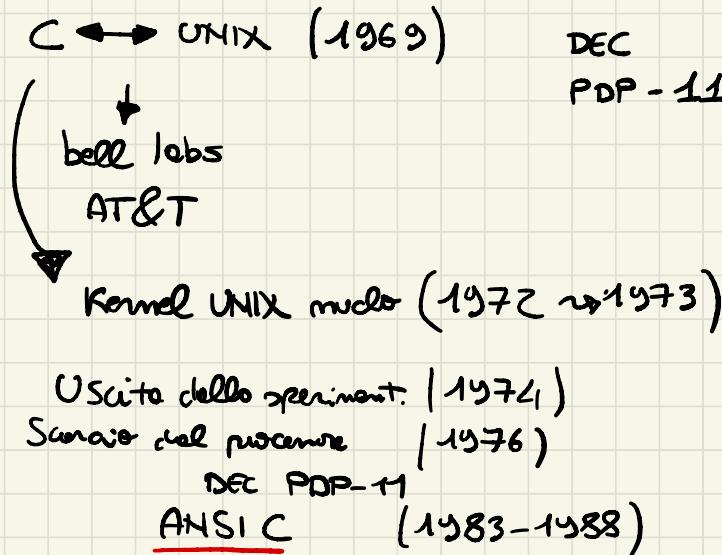
### 6. Monitorizzazione del programma

- Corrective
- Evolutive (AGGIUNTA FUNZIONALITÀ)

- a scrittura programma / file sorgente editor di testo ] (per il caso) gvim
- b compilazione " / " (file oggetto) ] (gcc)
- c collegamento del file sorgente con altri (file eseguibile) ]
- d caricamento del file eseguibile in memoria principale ] licenza lms creazione (debugger) (gdb)
- e esecuzione del file eseguibile

## IL LINGUAGGIO C

2.1



LINGUAGGIO C È UN LINGUAGGIO IMPENATIVO DI NATURA PROCEDURALE

↓  
SERVIZI DI COMANDI

↓  
SOPRIVISO IN SOTTOPROGRAMMI

▷ direttive al pre processore ▷

▷ Intestazione funzione main ▷

{

▷ Variabili funzione main ▷

▷ Istruzioni funzione main ▷

}

CORPO DELLA FUNZIONE

BODY

HEADER

① CONVENTIRE UNA DISTANZA ESPRESSA IN MIGLIA IN UNA DISTANZA  
ESPRESSA IN KM

② ANALISI DEL PROBLEMA :

- dati di input → DISTANZA IN MIGLIA
- dati di uscita → DISTANZA EQUIVALENTE IN KM
- relazioni intercorrenti → RATT. CONVERGENZA DA MIG A KM  
 $1 \text{ miglia} = 1,609 \text{ km}$

③ Progettazione algoritmo

- ACQUISIRE LA DISTANZA IN MIG
- CALCOLARE LA DISTANZA EQUIV. IN KM
- COMMUNICARE LA DISTANZA EQUIV. IN KM

④ Implementazione algoritmo

#include <stdio.h>

#define KM\_PER\_Mi 1.609

int main (void)

{  
    double miglia, kilometer;

(convenzione programatori C)

LETTERE MAIUSC  
LETTERE MINUSC

1.609

/\* input distanza in miglia \*/  
/\* output distanza equivalente in km \*/

indentazione

```
printf ("Digite la distanza in miglia: ");  
scanf ("%lf",  
&miglia);
```

$$\text{Kilometri} = \text{miglia} * \text{KM\_PER\_MI};$$

```
printf ("La distanza equivalente in Km: %f\m",  
Kilometri);
```

```
Return (0);
```

}

identificatori: seg, lettere, cifre

che chiudono con una lettera

- mazovia
- Standard
- programmatore

## INCLUSIONE DI LIBRERIA

2.3

Sintomi:

#include < a file di intestazione di libreria standard >

oppure

#include "a file di intestazione di libreria del programma"

One l'importante al pre-processore C la sostituzione del testo con le direttive, stava, il contenuto nel file di intestazione, così da utilizzare tutto quello dichiarato nelle librerie stesse

## FUNZIONE MAIN

2.4

- OGNI programma C deve avere una main, dove le istruzioni, all'interno di essa, manipolano variabili comprese tra parentesi graffe
- Le prime istruzioni eseguite in un programma C, c'è la prima all'interno delle main, le altre istruzioni poi vengono poi eseguite nell'ordine prestabilito

## IDENTIFICATORI

2.5

Gli identificatori nel linguaggio C sono:

- sequenze di lettere
  - sequenze di cifre decimali
  - sottosostituzioni
- (NON INIZIANO CON UNA CIFRA DECIMALE)

Sono case sensitive → lettere minuscole ≠ da lettere maiuscole

Denotano nomi: - costanti simboliche (KM\_PER\_Mi)

- tipi (int, void, double)
- variabili (miglia, kilometri)
- funzioni (main, scanf, printf)
- istruzioni

Si dividono in

- RISERVATI (int, void, double, return)
- STANDARD (sinf, printf)
- introdotti dal programmatore (KM-PER-MI, miglia, chil)

Tab identificatori riservati, non modificabili del programmatore

(nel nome di utilizzo, per scopi diversi da quelli stabiliti dal linguaggio)

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Gli identificatori standard sono definiti all'interno delle librerie standard del C, ponendo come, modificabili ma è sconsigliato

Tutti gli altri nomi identificatori introdotti dal programmatore

Regole per una buona scelta degli identificatori:

- Diveni dei RISERVATI
- Diveni degli STANDARD
- Non: che quegli identificatori rappresentino
- Usare sottosti con identificatori di più parole (KM-PER-MI)
- Evitare identificatori molto lunghi o minimi
- Una convenzione c'è usare lettere tutte maiuscole per le costanti simboliche e lettere minuscole per tutti gli altri identificatori

Q

tutte maiuscole per le costanti simboliche e lettere minuscole per tutti gli altri identificatori

TIPI DI DATO  $\leftrightarrow$  STRUTTURA ALGEBRICA

- int  $\mathbb{Z}$  -13 → VIRGOA FISSA
- double 1.1E-13 → VIRGOA MOBILE
- char (TUTTO dentro 'a') → CHE SI TROVA NELLA TASTIERA)
- stringa "ciao"
- void TIPO VUOTO SI USA IN 2 SITUAZIONI
  - ASSEGNAZIONE DI PARAMETRI FORMALI
  - ASSUNTO DI VALORI

`#include <stdio.h>` → librerie standard input e output

- `scanf()`
- `printf()`

`scanf(<stringa formata>, <segnale di chiusura di variabili>);`

- %d → INTERO
- %lf → DOUBLE VIRGOA FISSA
- %le → DOUBLE VIRGOA MOBILE
- %lg → DOUBLE v-f o V.m → "%e"
- %c → CHAR → getch() → "%c"
- %s → STRINGA → "%s"

RESTITUI SCE UN RISULTATO  
n° DI VALORI ACQUISITI  
CONSIDERAMENTE RISPETTO  
ALLA STRINGA FORMATA

printf (4 stringe formate p, ,  
4 sequenze di espressioni);

→ FILE DATI  
new line

%d → INTERO

\n → A CARO

%f → DOUBLE VIRGOLA PISSA

\t → TABULAZIONE

%e → DOUBLE VIRGOLA MOBILE

\vTAB

%g → DOUBLE v-f o V.m

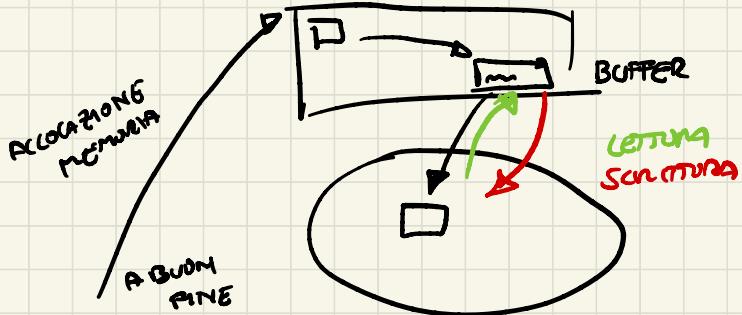
= "%c"

%c → CHAR → getch or → = "%c"

%s → STRINGA → %s man DS

ACQUISIZIONE BASTIT

FILE \* fid vor file D;



fid vor file D = fopen (" nome file"  
"r");

"r" READ

"w" SCRITTURA

→ RESTITUZIONE EOF

fscanf (fid vor file D,  
4 stringe formate p,  
4 seq imd vor D)

fprintf (fid vor file D,  
4 stringe formate p,  
4 seq esprive D)

fclose (4 id vor file D)

FOPEN . MAX numero di file aperti contemporaneamente

fEOF (4 id vor file D)

## ESEMPIO PROGRAMMA CONVERSIONE MIGLIA IN KM:

```
#include <stdio.h>

/***** definizione delle costanti simboliche ****/
/* definizione delle costanti simboliche */
/***** definizione della funzione main ****/
/* definizione della funzione main */
/***** dichiarazione delle variabili locali alla funzione */
FILE *file_miglia; /* input: file contenente la distanza in miglia */
double miglia; /* input: distanza in miglia */
FILE *file_chilometri; /* output: file contenente la distanza in chilometri */
double chilometri; /* output: distanza in chilometri */

int main(void)
{
    /* aprire i file */
    file_miglia = fopen("miglia.txt",
                         "r");
    file_chilometri = fopen("chilometri.txt",
                           "w");

    /* acquisire la distanza in miglia */
    fscanf(file_miglia,
           "%lf",
           &miglia);

    /* convertire la distanza in chilometri */
    chilometri = miglia * KM_PER_MI;

    /* comunicare la distanza in chilometri */
    fprintf(file_chilometri,
            "La stessa distanza in chilometri e': %f\n",
            chilometri);

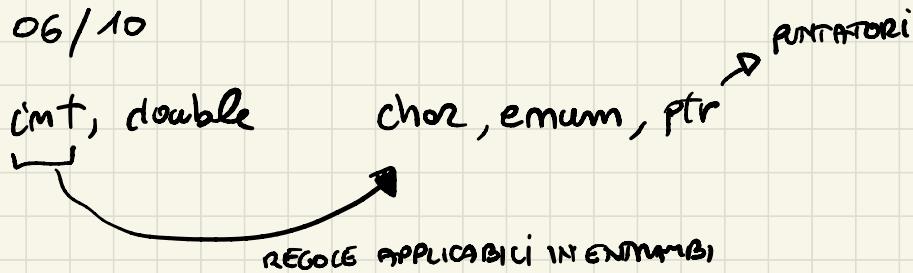
    /* chiudere i file */
    fclose(file_miglia);
    fclose(file_chilometri);
    return(0);
}
```

- Esempio di lancio in esecuzione del programma di Sez. 2.2 con ridirezione dei file standard per `scanf` (da `stdin` a `miglia.txt`) e `printf` (da `stdout` a `chilometri.txt`):

`conversione_mi_km < miglia.txt > chilometri.txt`

■fltp\_4

06/10

 $x, y, z$  ESPRESSIONE

$$\begin{array}{lll} x > z \quad \& \& \quad y > z \\ x == 1 \quad || \quad x == 3 & & x = 1 \quad o \quad x = 3 \end{array}$$

$x \leq y$

$x \leq z \quad \& \& \quad y \leq z \quad \cancel{x \leq z \leq y}$

$!(x \leq z \quad \& \& \quad y \leq z) \quad == \quad z < x \quad || \quad z > y \quad .. \quad x \quad y \dots$

$m \% 2 == 0 \quad m \% 2 == 1 \quad \text{int } n$

 $\downarrow$   
pari $\downarrow$   
dispari

$(\text{anno} \% 4 == 0 \quad \& \& \quad \text{anno} \% 100 != 0) \quad || \quad (\text{anno} \% 400 == 0)$

 $\downarrow$   
ANNO BISESTILE

int anno

se tutti gli operandi sono di tipo int allora l'operando è di tipo int altrimenti c'è di tipo double

REGOLA

UNICA ECCEZIONE  $\% \rightarrow \text{INT}$

$\Delta \text{VAR D} = \Delta \text{OPD}$

double      int  
5.0      0  
              s

ESEMPIO

int si trasforma in double

$\Delta \text{VAR} = \Delta \text{OPD}$

int      double  
g      9.99

ESEMPIO

int rimane int

si perdono tutte le informazioni

ESEMPIO  $5/2 \rightarrow 2 = 5/2 \rightarrow 2.5 = 5/2$

int int

int

int int

double

.no double

double

double int

### REGOLE DI PRECEDENZA TRA OP. DIVISI

- OPERATORI UNARI : + uno, - uno; !, ++, --
- OPERATORI ARITMETICI MOLTIPLICATIVI : \*, /, %
- OPERATORI ARITMETICI PROPRIETIVI : +, -
- OPERATORI RELAZIONALI D'ORDINE : <, >, <=, >=
- OPERATORI RELAZ. D'EGUAG : ==, !=
- OPERATORI LOGICI MOLTIPLICATIVI : &&
- OPERATORI LOGICI ADDITIVI : ||
- OPERATORE CONDIZIONALE : ?
- OPERATORE ASSEGNAZIONE : = ; +=, -=, \*=, /=, %=
- OPERATORE VERSOCA : ,

$x+y*3$

$x-\overbrace{y-z}^{\neq}$

## REGOLA DI ASSOCIAZIONE

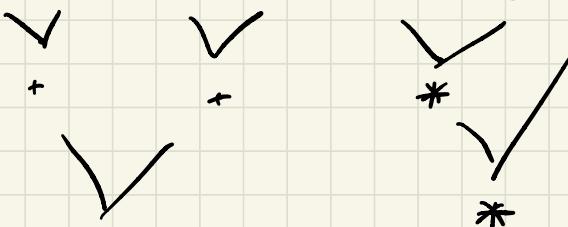
- TUTTI GLI OPERATORI SONO ASSOCIAZIVI DA SINISTRA TANNE :
- GLI OPERATORI DI INCREMENTO E DECREMENTO NON SONO ASSOCIAZIVI
- UNARI E ASSOCIAZIVI SONO ASSOCIAZIVI DA DX



N.B. PARENTESI  
PER DIVIDERE  
REGOLE ASSOCIAZIVE  
E PRECEDENZA

## ALBERO DI UCATAZIONE

$$(a+b) / (c+d) - w * w * z$$



RISULT

# ESEMPIO PROGRAMMA

## ① Specifica problema

Calcolare il valore complessivo di un insieme di monete depositate, esposto come numero di euro ed eventuale parte residua di euro

## ② Analisi problema

INGRESSO : numero di monete depositate di ogni tipo  
—> uscita : valore totale espresso un numero di euro  
relazioni: interconnessi  
 $1\text{¢} = 0,01\text{€}$   
e numero di contenimi residuali

## ③ Progettazione algoritmo

scelto di progetto calcolare in prime  
bottute in contenimi

### — poni algoritmo

- occupare numero monete depositate di ogni tipo
- calcolare valore tot in ¢
- calc valore in ¢ in €
- canonizzare val € e ¢

## ④

#include < stdio.h >

#define VAL-CENT-MONETE-0MC 1  
#define VAC-CENT-MONETE-02E 200

int main (void)

{

int num-monete-01c; /\* VARIAZIONE INPUT \*/  
int num-monete-02E; /\* VARIAZIONE INPUT \*/  
int valore-EURO; /\* VARIAZIONE OUTPUT \*/  
int FRAZIONE-EURO;

int value\_cent; /\* VARIABLE VARO \*/

## ISTRUZIONI LINGUAGGIO C

Istruzioni base (semplici)

$\langle \text{var} \rangle = \langle \text{espressione} \rangle ;$  Istruzione d'assegnamento



Valutazione e' inserimento in quella variabile

Istruzione controllo di flusso

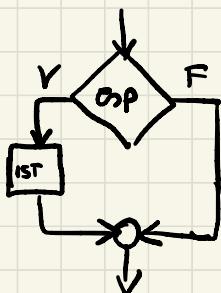
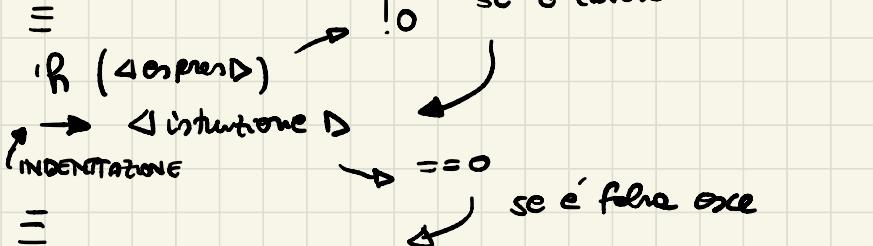
- Istruzioni di selezione (*if, switch*)

$\begin{cases} \langle \text{intD}_1, \dots, \text{intD}_m \rangle, \text{ISTRUZIONE} \\ \text{COMPOSTE} \end{cases}$

- Istruzioni di ripetizione (*while, for, do while*) (*repeat, until*)

### IF / SWITCH

IF SINGOLA



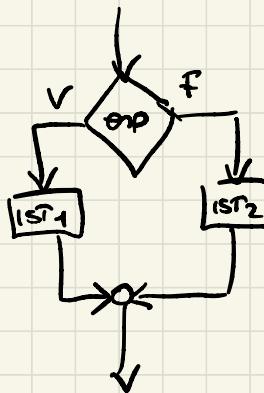
**IF CON PIÙ ALTERNATIVE**

if ( $\triangleleft$  expenditure  $\triangleright$ )

## ↳ Constraints ↳

*else*

4 street 2 ▷



ESP     $\text{tmp} = x;$  ;    }     $x = y;$  ;    }     $y = \text{tmp};$  ;    }    scontate valori

if ( $x > y$ )

S

$\text{tmp} = x;$   
 $x = y;$   
 $y = \text{tmp};$

۳

and if  $x > y$

## GSP 2 VI E

if ( $y \neq 0$ )

$$\text{ris} = x/y;$$

else

print F ("Divisione imponibile");

## ALTERNATIVE IMMEDIATE, CORRUOTE

`if ( $\downarrow \text{exp}_1 \downarrow 0$ )`

`1 int1 ;`

`else if ( $\downarrow \text{exp}_2 \downarrow 0$ )`

`1 int2 ;`

`else if ( $\downarrow \text{exp}_{m-1} \downarrow 0$ )`

`1 intm-1 ;`

`else`

`1 intm ;`

→ DI SOLITO CONTROLLA  
LE STEESSE VARIABILI

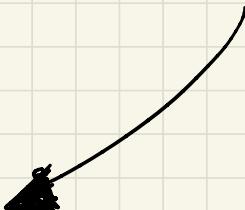
numerose chi che  $\leq$  o i numeri chi ci sono  
ogni chi viene associato all'if  
più vicino tra quelli liberi  
che lo precedono

`if ( $x == 0$ )`

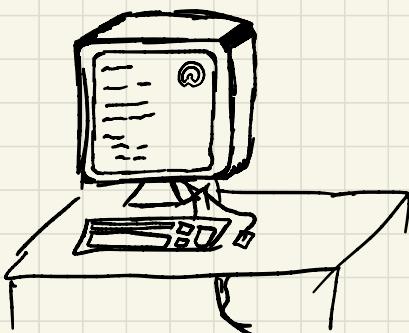
`if ( $y >= 0$ )`

`x + = y ;`  
    `else`

`x - = y ;`



{ }



DEVIANO  
DA REGOLE DI ASSOCIAZIONE

# SWITCH

## switch (<espressione>)

{

case &lt;val1&gt; :

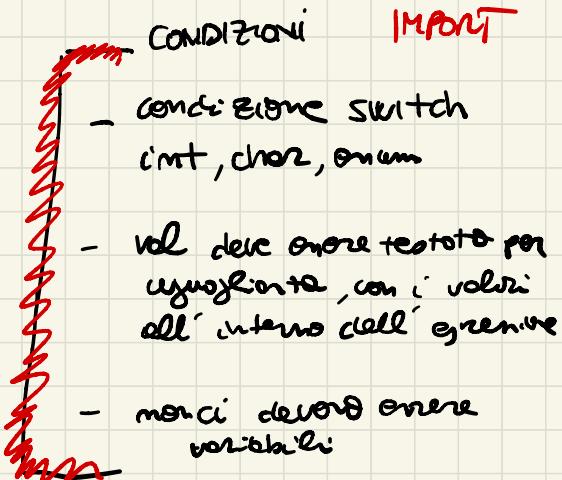
case &lt;val1, m1&gt; :

:     <int1>;  
      break;

case &lt;valm, 1&gt;

    <intm>;  
    break;case <val1, m<sub>m</sub>> ::     <intm>;  
      break;

case &lt;valm, &gt;

    <intm>;  
    break;



13/10

## SPECIFICA PROBLEMA

①

In un laboratorio può verificarsi una perdita di un materiale pericoloso che produce un certo livello iniziale delle radiazioni e poi si dimezza ogni 3 giorni.  
Calcolare il livello delle radiazioni ogni 3 giorni fino a raggiungere il giorno in cui il livello delle radiazioni scende al di sotto di quello di riferimento  $0.466 \text{ mrem}$ .

② Analisi del problema

- dati d'ingresso : LIV. INIZ. delle radiazioni
- dati d'uscita : GIORNO DI RIENTRO IN SICUREZZA  
LIV. RAD. OGNI 3 GIORNI

- relazioni/interventi : dimezzamento liv rad ogni 3 giorni

③ Progettare algoritmo

- acquisire il livello iniz. radiazioni
- calcolare e comunicare il liv. rad ogni 3 giorni finché il livello delle radiazioni non scende sotto il valore riferito
- comunicare il giorno del rientro in sicurezza

④ Implementazione algoritmo :

```
#include < stdio.h >
#define SOGLIA_SICUREZZA 0.0466
#define FATT_RIDUZIONE 2
#define N°_GIORNI 3
```

```
int main ( void )
```

```
{
```

```
DOUBLE LIV_INIZIALE ;
INT GIORNO_SICUREZZA;
DOUBLE LIV_CORRENTE ;
```

do

{

```
printf ("Dig. liv. init. radice (>0) : ");
scanf ("%lf",
&liv_init);
```

}

```
while (livello_init <= 0);
```

```
for (liv_corrente = liv_init,
```

```
giorno_sic = 0;
```

```
(livello_corrente >= SOGLIA_SICUREZZA);
```

```
livello_corrente /= FATTORE_RIDUZIONE;
```

```
giorno_nicht += numeri_giorni)
```

```
printf ("Il livello radice al giorno %d è %g f mrem.\n",
```

```
giorno_sicurezza,
livello_corrente);
```



3 spazi

3 spazi

+

WWWW,WWWW

```
printf ("Il giorno in cui mi posso lavorare in sicurezza %d.\n"
giorno_sicurezza);
```

```
return (0);
```

}

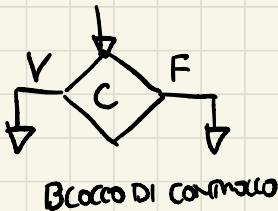
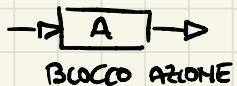
go to <etichetta>; (deviazione del flusso sequenziale)

funzione di salti

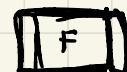
etichette -



## SCHEMI DI FLUSSO

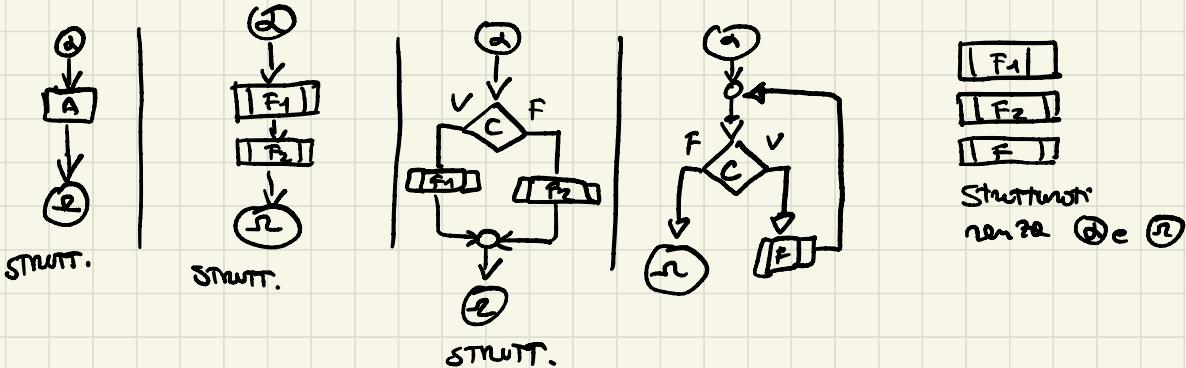


@ ① °



SCHM. DI  
FLUSSO

L'insieme degli schemi di flusso strutturati è definito per l'induzione sullo struttura grafica degli schemi di flusso come il più piccolo insieme di schemi di flusso tale che:

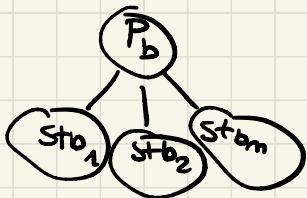
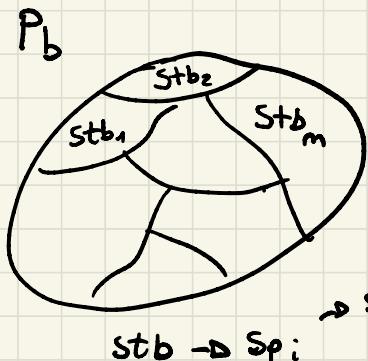


CONDIZIONE NECESSARIA AFFINCHÉ UNO SCHEMA DI FLUSSO TRUTTURATO E CHE ABbia UN SOLO DI INIZIO (1) E UN SOLO NODO DI FINE (2)

Teorema fondamentale --- strutturale (di Böhm - Jacopini)

Dato un programma  $P$  e uno schema di flusso  $F$  che lo descrive  
è sempre possibile determinare un programma  $P'$  equivalente a  $P$   
che è descritto come schema di flusso  $F'$  strutturato

## FUNZIONI

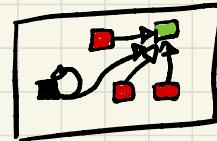


## VANTAGGI

- Maggiore leggibilità del codice
- Soddisfazione del lavoro
- Riuso del software

(architettura del software)

Favorizzare bene il codice



## Funzione C:

Sequenze di istruzioni logicamente correlate che lavorano su un insieme di dati parametrizzati

$f : A \rightarrow B$   
id funz      dominio      codominio

$B$     $f(A)$   
↓              parametri  
Tipo di risultato      funzione

SIMILITUDINE CON LA

$$f(a) = 2 + a \cdot 3$$

MATEMATICA

{ dati var loc  $\Rightarrow$   
4 const  $\Rightarrow$

4 direttive al pre processore  
4 definizione di tipi  
~~4 dichiarazione variabile globale~~

4 dichiarazione delle funzioni  
(mo main)

4 definizione delle funzioni  
(do rncm)

```
1 #include <stdio.h>
2 int somma(int a, int b) {
3     int c;
4     c=a+b;
5     return c;
6 }
7 int main() {
8     int n1=3;
9     int n2=2;
10    printf("%d ", somma(n1, n2));
11    return 0;
12 }
```

WWW.OKPEDIA.IT

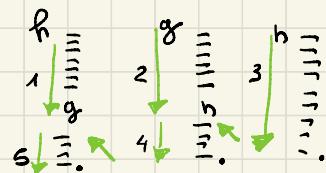
dichibrot. funt.

ORDINE SEQUENZIALE

4 tipo ritorno  
4 id funz (4 tipi var funzioni);  
void  
void

≡ ISTRUZIONI

→ FLUSSO DEL PROGRAMMA



4 tipo ritorno  
4 id funz (4 dichiarazione parametri formali)  
] intestazione  
funzione  
HEADER

corpo della funzione  
BODY

4 dichiarazione variabili locali funzione  
4 istruzioni funzione

}

DIC. FUNZ. CHE RICHIAMA FUNZ. \*

4 id funz (4 parametri eff.)  
actual parameters

## ISTRUZIONE RITORNO DA FUNZIONE (RETURN)

SINTASSI

RETURN ( $\alpha$  espressione  $\beta$ );

$\rightarrow$  REST IL VAL CHIESTO

(TERMINARE LA FUNZIONE IN CUI È ESEGUITA LA RETURN)

(LE ISTRUZIONI DOPO LA RETURN SONO COMPLETAMENTE NEGLIATI PERCHÉ NON VENGONO MAI ESEGUITI)

INT MAIN (void)  $\rightarrow$  RETURN (0);

INT MAIN (int argc,  
char \* argv[])  $\rightarrow$  numero di stringhe linea di comando  
 $\rightarrow$  stringhe in riga di comando

Parametro effettivo ponere per:

VALORE (CALL BY VALUE)

- Copia del valore

INDIRETTO (CALL BY REFERENCE)

- indirizzo (riconoscibili)

stato  $\alpha$  \*  $\beta$   
 $\beta$       \*  $\beta$   
indirizzo    valore  
              indirizzo

$\&v$   
 $\downarrow$  indirizzo di  
 $*p$   
 $\downarrow$  valore di

Input per valore

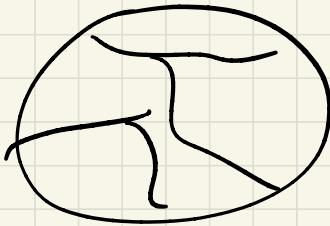
Output per indirizzo `scanf();`

Input/Output per indirizzo



20/10

## PROGRAMMAZIONE RICORSIVA



sotto pb siano dello stesso motivo del pb dato

Ricorrenza :

- con base
- con generali

## PRINCIPIO DI INDUZIONE

$\mathbb{N}$  è il più piccolo insieme che contiene 0 ed è chiuso rispetto all'operazione unaria di succendere

- ① Esiste  $0 \in \mathbb{N}$
  - ② Esiste succ :  $\mathbb{N} \rightarrow \mathbb{N}$
  - ③ Per ogni  $m \in \mathbb{N}$ ,  $\text{succ}(m) \neq 0$
  - ④ Per ogni  $m, m' \in \mathbb{N}$ , se  $m \neq m'$  allora  $\text{succ}(m) \neq \text{succ}(m')$
  - ⑤ Se  $M$  è un sott. di  $\mathbb{N}$  tale che
    - a)  $0 \in M$
    - b) per ogni  $m \in \mathbb{N}$ ,  $m \in M$  implica  $\text{succ}(m) \in M$
- principio  
di induzione

$0, \underbrace{\text{succ}(0)}, \underbrace{\text{succ}(\text{succ}(0))}_2, \dots$

precedens:  $N \not\models 0 \rightarrow \mathbb{N}$        $\text{pred}(\text{succ}(m)) = m$        $\text{succ}(\text{pred}(m)) = m$ ,  $m \neq 0$

Determine norme  $m, m'$

$$m \oplus m = \begin{cases} m & \text{se } m=0 \\ \underbrace{\text{succ}(m) \oplus \text{pred}(m)}_1 & \text{se } m \neq 0 \end{cases}$$

$$5 \oplus 2 = \text{succ}(5) \oplus \text{pred}(2) \Rightarrow 6 \oplus 1 \Rightarrow 7 \oplus 0$$

$\text{succ}(6) \oplus \text{pred}(1)$

$$m \leq m' \text{ se e solo se } \exists m' \in \mathbb{N} \text{ t.c. } m \oplus m' = m' \quad \frac{\Downarrow}{\not\exists}$$

SOMMAZIONE  $m \oplus m'$

$$m \oplus m' = \begin{cases} m & \text{se } m=0 \\ \text{pred}(m') \oplus \text{pred}(m) & \text{se } m>0 \end{cases}$$

MOLTIPLICAZIONE  $m \otimes m'$

$$m \otimes m' = \begin{cases} 0 & \text{se } m=0 \\ m \oplus (m \otimes \text{pred}(m')) & \text{se } m>0 \end{cases}$$

$$\begin{matrix} m \otimes m \\ m \neq 0 \end{matrix} = \begin{cases} 0 & \text{se } m > m' \\ \text{succ}((m \oplus m) \otimes m') & \text{se } m \leq m' \end{cases}$$

$$5 \otimes 2 \rightarrow \text{succ}((5-2):2) = \text{succ}(3:2) \rightarrow \text{succ}(\text{succ}(3-2):2)$$

$$\rightarrow \text{succ}(\text{succ}(1:2)) \rightarrow \text{succ}(\text{succ}(0)) = 2$$

INT ADDITiONE (int m, /\* $m \geq 0$ \*/,  
int n) /\* $n > 0$ \*/

```
{  
    int somma;  
    if (m == 0)  
        somma = m;  
    else  
        somma = ADDITiONE (m+1, n-1);  
    RETURN (somma);  
}
```

$$m! = \prod_{i=1}^m i = 1 \cdot 2 \cdot 3 \cdots \cdot m \quad 0! = 1$$

$$m! = \begin{cases} 1 & m = 0 \\ m \times (m-1)! & m > 0 \end{cases}$$

# 27/10 MODELLO DI ESECUZIONE SEQUENZIALE BASATO SU PILA

(STACK)

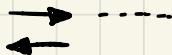
LISTA



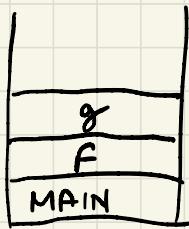
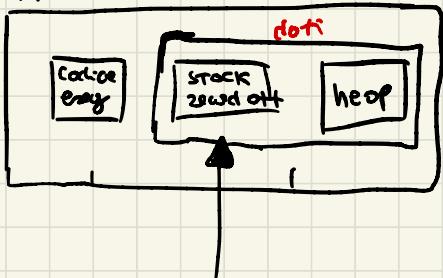
code / queue / fifo



pila / stack / lifo



RAM



- invocazione di funzione
- viene allocato record (in cima alla stack)
- per param., var, loc, info controllo (P.C.)  
(iniz, param, eff)
- reimpostare il PC : prima instruz funz. invocate
- terminazione dell'eseguibile della funz. prec. allocata  
- cancell. record di ottien.
- eventuale risultato non nel record della funzione chiamante
- P.C. viene risoluto con il valore risoluto in precedenza

## LIBRERIE DEFINITE DAL PROGRAMMATORI

COSA | File .h

interpretazione

↳ dich costant. simboliche da esport

↳ dich tipi da esport

↳ definiz. variabili globali da esport

↳ definiz. funz. da esport

} extern

come | File .

implementazione

↳ direttive al pre-procavore

(costant. simboliche da esp/import)

↳ definiz. tipi

(tipi da esport/import)

↳ definiz. variabili

(da esport/import)

↳ definiz. funz.

(da esport/import)

↳ definiz. funz. (no main)

(da esport/import)

Programma .c      libreria .c

↳ libreria .h

COMPILATORE PROGRAMMA .c 1

LIBRERIA .c 2

ASSOCIAZIONE A LIBRERIA .H 3

ESEMPIO

Frazioni.c  
(mcr mcm)

Frazioni.h

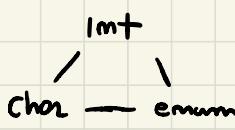


10 / 11

TIPO ENUMERATO

Onum { 4 seq id val > }

0, 1 .... m-1



puntatore (c'è niente indirizzo di memoria)

typedef enum { false,  
true } boolean\_t;

0 1

typedef enum { lunedì,  
martedì,  
mercoledì,  
...  
domenica } giorno\_t;

6

giorno\_t oggi,  
domani;

Domani = (oggi + 1) % 7;

int  
char  
Onum  
puntatore

double

double  $\xrightarrow{\text{opp}} \underbrace{\text{int}}_{\text{double}}$   $\xrightarrow{\text{opp}} \text{double} .0$   
double  

---

opp assegnamento  
double = int  
.0  
int = double  
trascrizionato

↓

CONVERSIONI AUTOMATICHE  
DI TIPO

## OPERATORI DI CAST (CONVERSIONI ESTPLICATE)

(a tipo A) o espressione A

monica, prefissi stesse per ++, --, +-; !

double x

int y

int z

x = y / z;

INT

3
2

1.0

3.0

x = (double) y / z ;

DOUBLE

1.5

TIPI DI DATI STRUTTURATI OMogenei

Value array è un aggregato di valori dello stesso tipo che sono memorizzati in celle consecutive di memoria

a tipo elementi o identificatore variabile array [a espressione di dichiara]

costant. simbolica

nopp. inserito in memoria  
del 1° elemento

omogeneo

int a [10]; e

7	33	5	82	-1	-12	3	24	32	57
0	1	2	3	4	5	6	7	8	9

[ ] = operatore di  
individuazione  
operatore monico  
post fixo  
precedendo memoria

a identf. varib. array > [espressione indice >]  
int, si vor, 0 .... m-1

a identf. variab. array > + a espressione indice >  
e

$$e[4] = -1 ; \quad e[4] = 0 ; \quad x = e[4] - 3$$

ESEMPIO

int e [NUM\_ELEM];

for (i = trovato = countare = 0 ;  
(i < NUM\_ELEM & & !trovato);  
i++)

trovato = e[i] == valore;  
countare++;

parametri formali di tipo array

cont 1 tipo elem > 1 id per array > []

↳ con questo dicitura il pongno non viene modificata  
l'array

array multidimensionale

int tabella [2][3]; = { { 7, 9, 4 }, {  
tabella [i][j] { 1, 3, 4 } } }; j

17 / 11

ESEMPIO PROGRAMMA ARRAT



15/12

## VERIFICA DI CORRETTEZZA

$$\{Q\} \; S \; \{R\}$$

invariante di ciclo  
tempo residuo

teorema dell'invariante di ciclo

dato un'intuizione di ripetizione "while ( $\beta$ ) S",  
se esiste un predicato  $P$ , e una funzione  $\underline{\text{tempo residuo}}$   
tali che:

PROPRIETÀ

$$\{P \wedge \beta\} \; S \; \{P\}$$

INVARIANTE

$$\{P \wedge \beta \wedge \text{tr}(i) = t\} \; S \; \{\text{tr}(i+1) < t\}$$
 prendendo

$$\{P \wedge \text{tr}(i) \leq 0\} \models \neg \beta$$
 limitatezza

allora

$$\{P\} \; \text{while } (\beta) \; S \; \{P \wedge \neg \beta\}$$

TERMINA      VERO

se  $(P \wedge \neg \beta) \models R$ , allora  $\{P\} \; \text{while } (\beta) \; S \; \{R\}$

## Esempio

somme = 0;

$\{P\}$        $i = 0; \quad \beta$   
 while ( $i \leq g$ )  
 {  
 $P''$   
 S     $[$  somme = somme +  $a[i]; \quad P'$   
 $i = i + 1; \quad P$   
 }  
 $\{P \wedge \neg \beta\}$

$$R = \left( \text{somme} = \sum_{j=0}^g a[j] \right)$$

$$P = (0 \leq i \leq 10 \wedge \text{somme} = \sum_{j=0}^{i-1} a[j])$$

$$\text{tr}(i) = 10 - i$$

## INVARIANZA

$$\{P \wedge i \leq g\} \leq \{P\}$$

$$P_{i+1} = (0 \leq i+1 \leq 10 \wedge \text{somme} \sum_{j=0}^{i-1} a[j]) = P$$

$$P'_{\text{somme}, \text{somme} + a[i]} = (0 \leq i+1 \leq 10 \wedge \text{somme} + a[i] = \sum_{j=0}^{i-1} a[j])$$

$$P''$$

$$\begin{aligned}
 (\rho \wedge i \leq 9) &\equiv \left( 0 \leq i \leq 10 \wedge \text{some} = \sum_{j=0}^{i-1} a[j] \wedge i \leq 9 \right) \\
 &\equiv \left( 0 \leq i \leq 9 \wedge \text{some} = \sum_{j=0}^{i-1} a[j] \right) \\
 \models \left( 0 \leq i \leq 10 \wedge \text{some} = \sum_{j=0}^{i-1} a[j] \right) &= \rho"
 \end{aligned}$$

LIMITE ET ZÉRO

$$\begin{aligned}
 (\rho \wedge r(i) = 0) &\equiv \left( 0 \leq i \leq 10 \wedge \text{some} = \sum_{j=0}^{i-1} a[j] \wedge 10-i \leq 0 \right) \\
 &\equiv \left( 0 \leq i \leq 10 \wedge \text{some} = \sum_{j=0}^{i-1} a[j] \wedge 10 \leq i \right) \\
 &\equiv \left( i = 10 \wedge \text{some} = \sum_{j=0}^9 a[j] \right) \\
 \models (i \neq 9) &
 \end{aligned}$$

$$\begin{aligned}
 (\rho \wedge \gamma \beta) &\equiv \left( 0 \leq i \leq 10 \wedge \text{some} = \sum_{j=0}^{i-1} a[j] \wedge i > 9 \right) \\
 &\equiv \left( i = 10 \wedge \text{some} = \sum_{j=0}^9 a[j] \right) \models R
 \end{aligned}$$

↓

$\{R\}$

