NUMERO: 1564A - ANNO: 2015

A P P U N T I

STUDENTE: Botta

MATERIA: Informatica. Prof.Mezzalama

① Personal (client)

Workstation

Server elaboratore che fornisce servizi ad altri elaboratori ("clients") attraverso una rete ("computer network").

Mainframe (host) elaboratori di grandi prestazioni usati x rilevanti applicazioni

Server Farm: insieme di server nello stesso locale.

software.

esprimere → strumenti astrazione e metodi formali → tecniche di programmazione automatica

PROBLEMA → SOLUZIONE → SOLUZIONE FORMALE → REALIZZAZIONE

Punti critici: - Sviluppo di una soluzione "informale"

- formalizzazione di una soluzione

permette una più semplice "traduzione" delle regole di realizzazione

La soluzione passa attraverso lo sviluppo di un algoritmo

ALGORITMO: descrizione precisa (formale) di una sequenza finita di azioni che devono essere eseguite per raggiungere la soluzione.

LINGUAGGIO MACCHINA (1011 0) ↔ LINGUAGGIO ALTO LIVELLO (A = 3,0 + C/B)

↓ TRADUTTORE

STADI SVILUPPO

① Scrittura di un programma
- file "sorgente" scritto linguaggio di programmazione

② Traduzione in linguaggio macchina
- Assembler in + fasi
- Gestore automaticamente chi traduce

Soluzione informale: descrizione a parole.

formale: descrizione attraverso operazioni elementari

pseudo-codice, flowchart + intuitivo + astratto richiede la conoscenza dei blocchi

immediata descrizione poco astratta, interpretazione + complicata

ALU (Aritmetic-logic unit): svolge tutti i calcoli aritmetici e logici
Solitamente è composta da circuiti combinatori

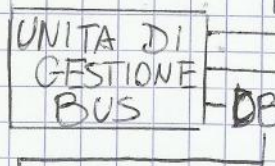
FPU (Floating Point Unit): una ALU dedicata ai numeri reali, conosciuta come coprocessore matematico.

REGISTRI: Elementi di memoria locale usati per conservare temporaneamente dei dati. Sono pochi (da 8 a 128) con dimensioni in funzione della potenza del processore (da 8 a 64 bit).
Si parla di architettura a n bit in funzione della dimensione dei registri (es. architettura a 32 bit, architettura a 64 bit).

UNITÀ DI CONTROLLO: È il "cervello" dell'elaboratore: in base al programma fornitole ed allo stato di tutte le unità decide l'operazione da eseguire ed emette gli ordini relativi

CPU (SCHEMA):

I BUS:
il "sistema circolatorio" del PC. Con BUS si intende un canale attraverso cui diversi componenti elettronici dialogano tra loro.



(data bus - bus dati) è il canale utilizzato per trasferire byte tra la CPU e le altre componenti (memoria esterna).
L'ampiezza determina quanti byte sono trasferiti contemporaneamente.
ES: 16 bit → 2 byte, 32 bit → 4 byte

ABUS (address bus - bus degli indirizzi) è utilizzato dalla CPU per specificare l'indirizzo della locazione di memoria che la CPU vuole leggere o scrivere
L'ampiezza (n° bit) determina l'ampiezza massima della memoria accessibile
es: un ABUS a 16 bit permette di indirizzare al max $2^{16} = 64$ KB locazioni di memoria.

CBUS (control bus - bus di controllo)

MIPS (Milioni di istruzioni al secondo)
- Frequenza di clock f [Hz = cicli/s]
- Periodo di clock $T = 1/f$
- Cicli macchina $C = \frac{\text{Periodo di clock}}{\text{istruzione}}$
- IPS = $f/C = 1/T \cdot C$
- MPS = IPS / 10^6

CHIP: circuito integrato, microcircuito, componente elettronico contenente moltissimi transistor (~32 nm)

x10 transistor → unità logica

x10 → circuito integrato (MSI)
x10^6 → circuito integrato (VLSI)

CLOCK: ogni elaboratore contiene un elemento di temporizzazione (il clock) che genera un riferimento temporale comune per tutti gli elementi costituenti il sistema di elaborazione

UNITÀ INPUT/OUTPUT: trasformano informazioni dal mondo umano a quello computer e viceversa
- diversi segnali fisici, analogici, asincroni
- segnali solo elettronici, digitali, sincroni.

MEMORIA: memorizzano i dati e le istruzioni necessarie all'elaboratore per operare.
Caratteristiche: indirizzamento, parallelismo, accesso (sequenziale o casuale).
INDIRIZZAMENTO: la memoria è organizzata in celle (unità minima), ad ogni cella è associato un indirizzo numerico che la identifica.
PARALLELISMO: ogni cella contiene una quantità fissa di bit, identica per tutte le celle di una stessa unità di memoria, accessibile con un unica operazione, multiplo del byte (min 1 byte).

MEMORIA INTERNA: all'interno dell'elaboratore, allo stato solido (chip), solitamente volatile, veloce, quantità limitata (qualche GB), non rimovibile.
MEMORIA ESTERNA: all'esterno, talvolta rimovibile, non elettronica (magnetica o ottica...), permanente, lenta, grande quantità (qualche TB)

Introduzione alla Programmazione

Dalla soluzione al programma: la scrittura del programma è immediata a partire dalla soluzione formale.

I linguaggi di programmazione forniscono costrutti di diversa complessità a seconda del tipo di linguaggio.

Diversi livelli di astrazione:

- Linguaggi alto livello: elementi del linguaggio hanno complessità equivalente ai blocchi dei diagrammi di flusso strutturati (condizionali, cicli...)
  - Es: C, C++, Basic, Java... → `if(x>3) then x=x+1 ...`
  - Indipendenti dall'hardware.
- Linguaggi "Assembler" o "macchina" o "binario": elementi del linguaggio sono istruzioni dipendenti dall'hardware. Specifici per una specifica Architettura.
  - ES: Assembler del microprocessore Intel Pentium. `LOAD REG1, Mem[1000] ...`

Elementi di un linguaggio:

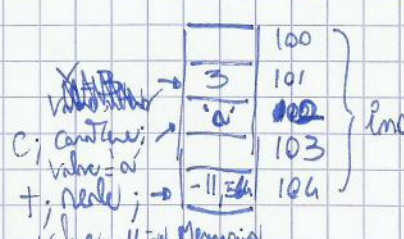
Lessico (insieme delle parole), Sintassi (la struttura della frase), semantica (significato).

Esistono elementi lessicali e sintattici essenziali x l'uso del linguaggio stesso: Parole chiave (Keyword), Identificatori, Istruzioni.

Gli elementi sintattici definiscono la struttura formale di tutti i linguaggi.

KEYWORD: Vocaboli riservati al traduttore per riconoscere altri elementi del linguaggio, non possono essere usate per altri scopi. Costituiscono i "mattoni" della sintassi

Dati: sono individuati da un nome (identificatore), un'interpretazione (tipo), una modalità d'accesso (costante o variabile)



IDENTIFICATORE: indica il nome di un dato o di altre entità.

TIPO: indica l'interpretazione dei dati di memoria. Permette di definire tipi primitivi (numeri, simboli) indipendentemente dal tipo di memorizzazione del sist.

Il compilatore C traduce i programmi sorgenti scritti in C in programmi eseguibili. È a sua volta un programma eseguibile. Controlla l'assenza di errori di lessico e sintassi. Non serve all'utente.

Un programma formalmente corretto non è detto che faccia l'operazione desiderata.

SCRITTURA DEL PROGRAMMA: un sorgente è un file di testo, si utilizza editor di testi (Blocco note, editor specializzati)

Indentazione automatica, colorazione della sintassi, compilazione automatica...

Ambienti integrati: applicazioni software integrate (IDE) che contengono al loro interno un editor di testi, un compilatore, un debugger (ambiente di verifica).

IDENTIFICATORI: si riferiscono a: costanti, variabili, sottoprogrammi.

regole iniziano con carattere alfanumerico o "_"; contengono caratteri alfanumerici o "_"

Interni: le entità del programma, case sensitive, significativi i primi 31 caratteri

Esterni: gli oggetti del sistema, case insensitive, significativi i primi 6 caratteri.

Riservati: parole chiave di C, elementi della libreria C standard.

COMMENTI: testo libero all'interno del programma non considerato dal compilatore:

/* commento */

STRUTTURA PROGRAMMA C:

Parte dichiarativa globale → elenco degli oggetti che compongono il programma e specifica le loro caratteristiche. Gli oggetti sono tipicamente dati di tipo num. e non num.

main() → parola chiave per iniziare

{

Parte dichiarativa locale → elenco degli oggetti che compongono il main e specifica le loro caratteristiche

Parte esecutiva → Sequenza di istruzioni

}

PRE-PROCESSORE C: la compilazione passa per un passaggio preliminare che precede la traduzione in linguaggio macchina, realizzato dal pre-processore. Funzione principale: espansione delle direttive che iniziano con '#': #include, #define

- #include <file>

↓

viene espanso ed incluso nel programma sorgente

Il sist. decimale è posizionale $253 = 2\times100 + 5\times10 + 3\times1$

Sist di numerazione posizionale → bisogna definire la base B da cui discendono varie caratteristiche: - cifre = $\{0, \dots, B-1\}$

- peso della i-esima cifra = B^i
- rappresentazione numeri naturali su N cifre $A = \sum_{i=0}^{N-1} a_i B^i$

Il sistema binario

Base = 2 → cifre = $\{0,1\}$ $\quad 101_2 = 1\times2^2 + 0\times2^1 + 1\times2^0 = 4+0+1 = 5_{10}$

0_2	0_{10}	1000_2	8_{10}	$2^0 = 1$	$2^4 = 16$
1_2	1_{10}	1001	9		
10_2	2_{10}	1010	10		
11	3	1011	11	$2^1 = 2$	$2^5 = 32$
100	4	1100	12		
101	5	1101	13	$2^2 = 4$	$2^6 = 64$
110	6	1110	14		
111	7	1111	15	$2^3 = 8$	$2^7 = 256$
	...			...	

Conversione binario → decimale: somma pesata delle cifre.

$1101_2 = 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 = 8+4+0+1 = 13_{10}$

(3 2 1 0)

1010010110

↑ MSB most significant bit

↑ LSB less significant bit

Con N bit si possono rappresentare $2^N - 1$ numeri es:

se $N = 3$ $\quad 2^N - 1 = 8-1 = 7$ infatti $7_{10} = 111_2$, $8_{10} = 1000_2$, $3_{10} = 11_2$, $4_{10} = 100_2$

$N = 2$ $\quad 2^2 - 1 = 4-1 = 3$

Conversione decimale → binario: divisioni successive

$25_{10} = ?$

25 : 2

resto 1 — 12 : 2

resto 0 — 6 : 2

R_3 0 — 3 : 2

$R_4 = 1$ — 1 : 2

$R_5 = 1$

modo di restituire il contenuto $R_5\ R_4\ R_3\ R_2\ R_1$

$1\ 1\ 0\ 0\ 1_{(2)} = 25_{(10)}$

6_{10} : 2

$R = 0$ — 3 : 2 $\quad 6_{10} = 110_{(2)}$

$R = 1$ — 1 : 2

$R = 1$

Con N bit si possono rappresentare i numeri: $0 \le x \le 2^N - 1$

I numeri con segno:

Esistono vari modi per rappresentare il segno:

- Modulo e segno
- Complemento ad 1
- Complemento a 2
- Eccesso X

Codifica modulo e segno:

1 bit	N-1 bit
segno	modulo

un bit per il segno (solitamente il MSB): 0 = "+" ; 1 = "-"

N-1 bit per il modulo

ES con N=4 bit

$+3_{(10)} \rightarrow 0011_{(MeS)}$ $\quad 0000_{(MeS)} \rightarrow +0_{(10)}$

$-3_{(10)} \rightarrow 1011_{(MeS)}$ $\quad 1000_{(MeS)} \rightarrow -0_{(10)}$

Svantaggi:

- doppio zero (+0 e -0)
- operazioni complesse

ES: somma A+B

	A>0	A<0
B>0	A+B	B-|A|
B<0	A-|B|	-(|A|+|B|)

- In una rappresentazione su N bit si possono rappresentare i numeri tra: $-(2^{N-1}-1) \leq X \leq (2^{N-1}-1)$

Complemento a 2:

numero 0 1 1 0 1 0 $1_{(CA2)}$ $\quad$ 1 0 1 1 0 1

peso cifra -2^6 2^5 2^4 2^3 2^2 2^1 2^0 $\quad$ -2^5 2^4 2^3 2^2 2^1 2^0

$0+32+16+0+4+2+1=21$ $\quad -32+8+4+1=-19$

$0001_{(CA2)} \rightarrow +1_{(10)}$ Il Msb ha peso negativo, le altre cifre positivo.

$1111_{(CA2)} \rightarrow -1_{(10)}$ L'Msb indica sempre il segno: 0="+", 1="-"

$1000_{(CA2)} \Rightarrow -8_{(10)}$ La codifica CA2 è la più utilizzata perché semplifica le operazioni aritmetiche.

La somma e la differenza si eseguono direttamente senza badare ai segni

Es overflow sottrazione CA2

$3-(-6) = 3+6$

$0011 +$

$0110 =$

1001 (-7!)

$-3-(-6) = -3+6$

$1101 +$

$0110 =$

~~1~~0011 (3)

+9 non rappresentabile su 4 bit

Rappresentazione dei numeri reali

Occorre indicare la posizione della virgola: si usa la notazione a virgola mobile (floating point)

notaz. scientifica

$3,14 \rightarrow 0,314 \cdot 10^1$ $\quad 137 \rightarrow 0,137 \cdot 10^3$ $\quad N = \pm \text{Mantissa} \cdot \text{Base}^{\text{Esponente}}$

$0,0001 \rightarrow 0,1 \cdot 10^{-3}$

Nella memoria un numero è così memorizzato: [±] [E] [M] (la base è 2)

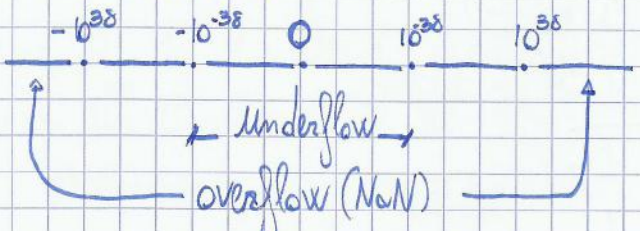
Un numero in tale notazione utilizza di norma 32 bit, fornendo un intervallo $-10^{38} \div 10^{38}$

Formato IEEE-754

- Mantissa nella forma 1,... (valore max < 2)
- Base = 2

	1 bit	8 bit	23 bit
Precisione singola	±	Esponente	Mantissa
Doppia precisione	1	11	52 bit

intervallo valori



Sistemi posizionali

Problemi: numero fisso del numero di bit

i numeri sono rappresentati da sequenze di cifre

→ Intervallo di rappresentazione

Overflow

Precisione

Rappresentazione di un testo in ASCII
- Caratteri in codice ASCII
  ogni riga terminata da CR+LF (Win) o LF (UNIX) o CR (Mac)
  pagina talvolta separata da FF

Un testo può essere memorizzato in 2 modi:
- formattato: sono memorizzate sequenze di byte che definiscono la formattazione.
- non formattato: memorizzati solo i caratteri.

Linguaggio di descrizione di pagina: formato di file x descrivere pagine di documenti indipendentemente dal dispositivo di stampa.
↓
PDF

Istruzioni per I/O
output: printf() input: scanf()
queste funz. richiedono la libreria studio.h
sintassi printf(<formato>, <arg1> ... <argn>);
↓ sequenza di caratteri che determina il formato di stampa degli argomenti
può contenere caratteri (stampati così come sono) direttive di formato es %d intero
%f float
%c carattere
%lf double

sintassi scanf(<formato>, <arg> ...);
↓ puntatore di &

Operatori relazionali
a == b → forniscono risultato: valore risultato = 0 FALSO
a != b risultato ≠ 0 VERO
a > b
a >= b
⋮
⋮

sintassi if (<condizione>) Se è vera condiz. esegui: Blocco 1
<Blocco 1> Altrimenti: Blocco 2
else
<Blocco 2>

Espressioni booleane: combinazione di variabili ed operatori

Funzioni booleane: applicazione "molti a uno" $f: B^N \to B$ es $f(A,B) = A \text{ e } (\text{non } B)$

Proprietà

- $A \times B = B \times A$
- $A \times B \times C = (A \times B) \times C = A \times (B \times C) = (A \times C) \times B$
- $A + B = A + B$
- $A + B + C = (A+B)+C = A+(B+C) = (A+C)+B$
- $A \times (B+C) = A \times B + A \times C$
- $A + (B \times C) = (A+B) \times (A+C)$

Regole precedenza:

- NOT > AND > OR

Teoremi base

- $A \times \bar{A} = \text{falso} = 0$
- $A + \bar{A} = \text{vero} = 1$
- $A + 1 = 1$
- $A + 0 = A$
- $A + A \times B = A$
- $A \times (A+B) = A$
- $A \times 1 = A$
- $A + \bar{A} \times B = A + B$
- $A \times (\bar{A} + B) = A \times B$
- $A \times 0 = 0$

Teoremi di de Morgan:

$f(a,b,\dots,z;+,\times) = f'(a',b' \dots z';\times,+)$

ovvero negando entrambi i membri

$f'(a,b,\dots z;+,\times) = f(a',b' \dots z';\times,+)$

es: $\cdot A+B = \overline{(\bar{A} \times \bar{B})}$

$\cdot \overline{(A+B)} = \bar{A} \times \bar{B}$

Dimostrazione algebra di Boole → confronto tabelle di verità

Dal problema al circuito:

Dato un problema per ottenere il circuito corrispondente si applicano i seguenti passi:

1- individuare le variabili booleane

2- Creare tabella della verità

3- Ottenere la funzione dalla tabella

4- Progettare il circuito usando le porte logiche corrispondenti con F

Dalla tabella alla funzione:

a	b	c	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$F = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + \bar{a}\bar{b}c$

si prendono le variabili che danno 1, si negano le variabili = 0, si moltiplicano tra loro, si sommano con le altre

Si dividono in 2 grandi categorie:

- Senza perdita (Loseless): l'operazione di decompressione permette di recuperare tutti gli elementi iniziali (es. zip)
- Con perdita (Lossy): i dati compressi non contengono tutta l'informazione iniziale. La decompressione non permette quindi di recuperare tutti gli elementi iniziali (MP3)

Per ridurre le quantità di dati da memorizzare/trasmettere si può cercare di eliminare le ripetizioni.

ES. codifica Run-length AAAAAABAAAAAAA → A7B1A7 (da 15 byte a 6 byte)

ES. Codifica dizionario "I Torinesi abitano a Torino" → "I ïesi abïano a ïo"
con l'informazione aggiuntiva (=dizionario) ï=Torin

Misurazione della compressione:

Rapporto o fattore di compressione: C = (dim dati)/(dim. dati compressi)
Solitamente espresso come N:1 o Nx

Risparmio spazio: $S = 1 - \frac{\text{dim. dati compressi}}{\text{dim. dati}}$
Solitamente espresso come %

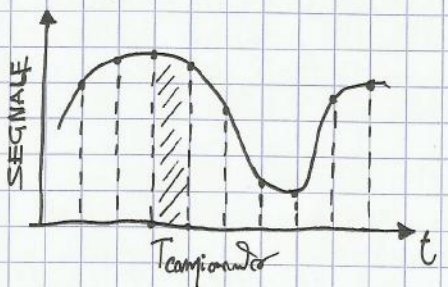
ES 10 MB compressi a 2 MB $C = \frac{10}{2} = 5 \rightarrow 5:1$ (o 5x)

$S = 1 - \frac{2}{10} = 1 - \frac{1}{5} = \frac{4}{5} \rightarrow 0{,}8 = 80\%$

Formati raster: BMP (fixmy senza compressione)
GIF (8BPP → 256 colori → immagini semplici; buona compressione Loseless)
PNG (24BPP → 16M colori; ottima compressione Loseless; ottimizzato x grafica e foto)
JPEG (24 BPP; compressione variabile, massima compressione è lossy; x foto)

Modifica dimensioni di un'immagine: Si parla di "scaling" (riduzione o ingrandimento). In genere si scalano meglio le immagini vettoriali (ossia = istruzioni per disegnarle) che quelle raster (= riduzione o ingrandimento dei pixels) e la differenza si nota maggiormente negli ingrandimenti.

Campionamento



Frequenza di campionamento: $f_c = \frac{1}{T_c}$

- Campionamento ad intervalli regolari
- La fedeltà è data da:
  - frequenza di campionamento (campioni/s)
  - numero di livelli distinti (ossia bit × campione)
- La frequenza di campionamento deve essere almeno il doppio della massima frequenza contenuta nel suono.

CD audio: la musica ha frequenze percepite tra 0 e 20 000 Hz.
44100 campioni al secondo (f_c) e 16 bit per campione.
Stereo (2 canali)
Owero 176 KBps (≈ 1,5 Mbps)
Formato CD audio: brani digitali su CD codificati con 16 bit/campione, 2 canali (stereo), campionamento a 44,1 kHz
Velocità di lettura (bitrate): dividendo × 8

$44100 \frac{\text{camp}}{s} \cdot 16 \frac{\text{bit}}{\text{camp}} \cdot 2 \text{ (canali)} = 1\,400\,000 \text{ bit per sec} \overset{:8}{=} 175000 \text{ bytes} \times \text{sec}$

10 MB ≈ 1' di audio su CD
Capacità di un CD:
- Capacità nominale è per registrare dati, 650 o 700 MB (≈ 74' e 80')
- Settori fisici da 2352 byte per:
  - Settori logici dati (es MP3) = 2048 byte
  - Settori logici audio = 2352 byte
- Capacità reale è maggiore:

700 MB / 2 kB = 358 400 settori fisici
$358400 \text{ settori} \times 2352 \frac{B}{\text{settori}} = 842\,956\,800 \text{ B}$
842 956 800 B / 175000 B/sec = 4816'' = 80'

ESERCIZIO "QUADRATO": costruire un quadrato di n asterischi x lato, con n dato da tastiera.

```
int main () {
int i, j, n;
printf ("inserisci n: ");
scanf ("%d", &n);
i = 0
while (i<n)
{ j=0
  while (j<n)
    {printf ("*");
     j++;
    }
  i++; printf ("\n")
}
```

ripete n volte: stampa una riga di asterischi
↓
stampa n asterischi e vai a capo

Istruzione for

sintassi
```
for (<inizializzazioni>; <condizione>; <incremento/decremento>) {
    blocco
}
```

equivalente a:
```
<inizializzazioni>
while (<condizione>) {
  blocco
  <incremento>
}
```

In generale si usa for per cicli di conteggio (numero di iterazioni noto a priori) e while per cicli generici (numero di iterazioni non noto a priori)

Interruzione dei cicli:

- break
  - termina il ciclo
  - esegue la prossima istruzione dopo la fine del ciclo
- continue
  - termina l'iterazione corrente
  - esegue l'iterazione seguente

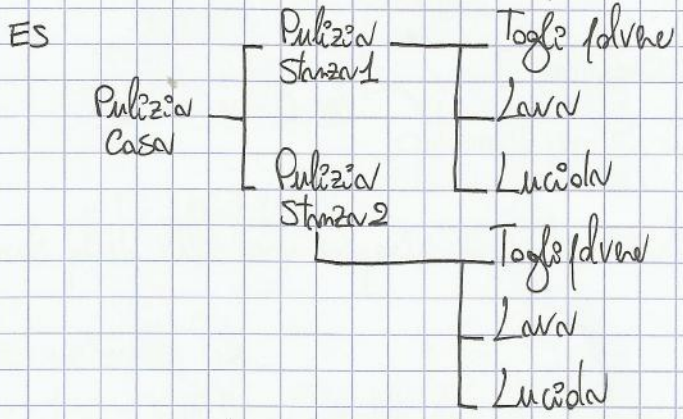
Matrici bidimensionali

sintassi <tipo> <nome>[<n° righe>][<n° colonne>]

Sottoprogrammi

Le soluzioni "monolitiche" non permettono il riuso, la comprensione può risultare difficile.

Approccio top-down: decomposizione del problema in sottoproblemi più semplici. Ogni sottoproblema può essere decomposto fino a sottoproblemi "terminali", cioè risolvibili in modo semplice.



I linguaggi di programmazione permettono di suddividere le istruzioni in blocchi detti sottoprogrammi, moduli...

- Procedure: sottoprogrammi che non ritornano un risultato
- Funzioni: sottoprogrammi che ritornano un risultato

Procedure e funzioni funzionano attraverso parametri o argomenti

Parametri ▶ funzione ▶ risultato

funzioni void ↔ procedure

```
int cubo (int a);   dichiarazione o prototipo
                    ↑ parametri
int main(){... w = cubo(x)... }   chiamata
                    ↓ parametro assegnato → diventa "a" della funzione
int cubo (int a){   definizione o corpo
  int c
  c = a*a*a
  return c; → risultato
```

È possibile modificare lo schema di passaggio per valore in modo che i parametri attuali vengano modificati dalla funzione:

PASSAGGIO "BY REFERENCE" (PER INDIRIZZO)

- Parametri attuali = indirizzi di variabili
- Parametri formali = puntatori al tipo corrispondente dei parametri attuali

Passando gli indirizzi dei parametri formali posso modificarne il valore.

- & <variabile> fornisce l'indirizzo di memoria di <variabile>
- * <puntatore> fornisce il dato contenuto nella variabile puntatore.

Le funzioni possono avere come parametri dei vettori:

- parametri formali: si indica il nome del vettore con [] senza dim.
- parametri attuali: il nome del vettore senza [].

Il nome del vettore indica l'indirizzo del primo elemento, quindi il vettore è passato per indirizzo. Gli elementi del vettore possono quindi essere modificati.

Data che il vettore è passato per indirizzo la funzione non riceve la dimensione del vettore. Occorre passare alla funzione anche la dimensione.

int funzvett (float a[], int dim)

Es. funzione che conta gli elementi non nulli di un vettore:

```
int nonnull(int v[], int dim)
{
  int i, n=0;
  for (i=0; i<dim; i++){
    if (v[i] != 0)
      n++
  }
}
return n; }
```

Stringhe

Le informazioni di tipo stringa vengono memorizzate come vettori di caratteri, terminati da un carattere aggiuntivo '\0' (NULL)

la stringa vuota contiene solo '\0' = ""

'a' il carattere a [a]

"a" la stringa a [a|\0]

x convertire char in int: `int i = '3' - '0';`

`sscanf(s, "%d-%f/%s", a, b, stringa);`

Trasf. stringa in num ← legge da s ed assegna agli argomenti

Operazioni I/O

`int sscanf(char* <stringa>, char* <formato>, <espressioni>);`
- restituisce EOF in caso di errore altrimenti il numero di campi letti.

`int sprintf(char* <stringa>, char* <formato>, <variabili>);`
- restituisce il numero di caratteri scritti.

↓ mette "cose" in stringa

```
int a = 45;
char s[10];
sprintf(s, "%d", i);
```

Oppure gets e puts, o scanf e printf

ES.

```
cost int MAX=20
char nome[MAX+1]
printf("nome?");
scanf("%s", nome);        | printf("nome?");
                          | gets(nome);
```

↑ senza &

```
printf("Ciao, %s!\n", nome);   | printf("Ciao, ")
                               | puts(nome);
```

Funzioni x stringhe (<string.h>)

`char* strcat(char* s1, char* s2);` concatenazione s1+s2

`char* strchr(char* s, int c)` cerca c in s

`strstr(char* s1, char s2)` cerca s2 in s1

`int strcmp(char* s1, char* s2)` =0 s1=s2 ; <0 s1<s2 ; >0 s1>s2

`int strlen(char* s)` lunghezza di s

`char* strcpy(char* s1, char* s2)` copia s2 in s1

`char* strncat(char* s1, char* s2, int n)` concatena a s1 n caratteri di s2

strncpy

strncmp

Acquisizione/stampa di una riga x volta

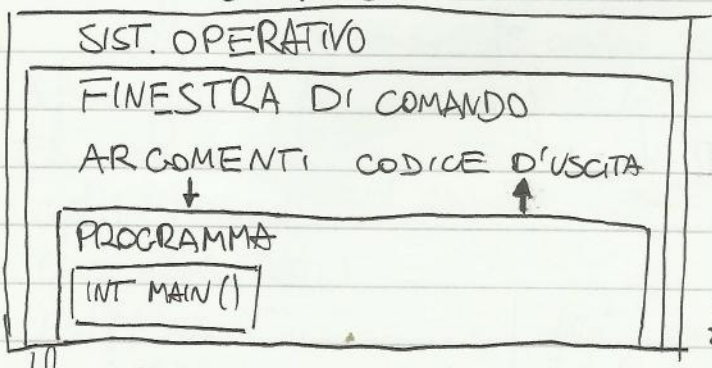
char* gets(<stringa>) • legge una riga dalla tast. fino a \n
• in caso di errore il risultato è NULL

int puts(<stringa>) • aggiunge \n alla stringa stampata

char*: puntatore → contiene l'indirizzo di memoria in cui il primo carattere della stringa è memorizzato

puts(s) è identico a printf("%s\n", s)

LINEA DI COMANDO



In C è possibile passare info dall'esterno al programma specificando degli argomenti sulla linea di comando

ES C:\> prog <arg1> <arg2> ...

Automaticamente memorizzate negli argomenti del main

main(int argc, char* argv[])

• argc: è il numero di argomenti inseriti, il primo è sempre il nome del programma

• argv: vettore di stringhe, le stringhe sono gli argomenti inseriti
argv[0] = 1° elemento = nome programma
argv[argc-1] = ultimo elemento

ES. C:\> prog.exe 3 file.dat 3.2

argc = 4 argv[0] = "prog.exe" argv[1] = "3" argv[2] = "file.dat"
argv[3] = "3.2"

NOTA:
Qualunque sia la natura dell'argomento è sempre una stringa, è necessario quindi convertirlo il dato.

accesso ai campi <variabile>.<campo>

ES
```
struct nComplesso {
    double reale;
    double immag;
};
...
struct nComplesso num1, num2;
/* variabili di tipo nº complesso dichiarate */
...
num1.reale = 2; num1.immag = 42;
...
```

```
struct stud {
  char cognome[MAX], nome[MAX];
  int matricola;
  float media;
};
struct stud s,t;
```

È possibile definire un nuovo tipo a partire da una struct:
typedef <tipo> <nome nuovo tipo>

ES
```
Typedef struct ncomplesso {
    double re;
    double im;
} compl;
...
compl num1, num2;
...
```

Non è possibile confrontare due variabili dello stesso tipo di struct → si confrontano i campi.

I campi mancanti sono inizializzati a 0 x valori numerici, NULL x puntatori.

ES. Definire struct x studente, invocare funz che riceve un vettore e ritorna insufficienti

```
#include <stdio.h>
#define NSTUD 10
typedef struct stud {
    char nome[40];
    unsigned int matricola;
    unsigned int voto;
} studente;
int contaInsuff (studente vett [], int dim);
main () {
 int i, numIns;
 studente Lista[NSTUD];
 /* supposto già letto */
 numIns = contaInsuff(Lista, NSTUD);
 printf ("il nº è %d "; numIns);
 return 0;
}
int contaInsuff (studente vett[], int dim) {
  int i, n = 0;
   for (i=0, i<dim ; i++) {
     if (vett[i].voto <18) n++;
   return n;
}
```

Il C vede i file come flusso (stream) sequenziale di byte.
- Nessuna struttura particolare: la strutturazione del contenuto è a carico del programmatore.
- Carattere terminatore alla fine del file: EOF

L'accesso sequenziale implica l'impossibilità di leggere all'indietro e di saltare ad uno specifico punto del file.
In un programma C, esiste un tipo di dato specifico per rappresentare le informazioni relative ad un file aperto:
- denominato file stream
- tipo di dato FILE * (<stdio.h>)

Aprire un file significa quindi creare un nuovo stream ed associarlo ad uno specifico file sul disco.
Una volta che il file è aperto, il suo stream rappresenta:
- un collegamento mediante il quale poter compiere delle operazioni sul contenuto del file.
- le modalità di accesso scelte (testo/binario, lettura/scrittura).
- La posizione attuale a cui si è arrivati nella scrivere o nel leggere il file.

Ogni operazione sul file avviene chiamando una funzione che riceve lo stream come parametro.
Dichiarazione: FILE * <file> FILE * contiene un insieme di variabili che gestiscono l'accesso per "tipi".
Al momento dell'attivazione di un programma vengono automaticamente attivati tre file: stdin, stout, stderr.
stdin è automaticamente associato allo standard input che è la tastiera.
stout e stderr sono automaticamente associati allo standard output (video).
Stdin, stout e stderr sono direttamente utilizzabili nelle istruzioni per l'accesso a file, in altre parole sono delle variabili predefinite di tipo file*.

L'uso di un file passa attraverso 3 fasi fondamentali:
- Apertura
- Accesso
- Chiusura.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 39 di 46

CONTROLLO DELL'ERRORE

```
int ris;
...
int ris = fclose(f);
if(ris != 0) {
  printf("errore");
  return 1;
}
```

LETTURA DEL FILE

apertura → File aperto in lettura
Posizione iniziale 1° carattere

leggi riga/carattere → File aperto in lettura
Posizione intermedia n-esimo carattere

chiusura ← File aperto in lettura
Posizione finale ultimo carattere

apertura → File aperto in scrittura
Posizione 1° carattere

scrivi riga/carattere → File aperto in scrittura
Posizione intermedia → la posizione intermedia diventa finale

chiusura

Lettura e scrittura su file
lettura → legge prossimo elemento fino alla fine
scrittura → scrive o aggiunge

	lettura	scrittura
carattere singolo	fgetc	fputc
Riga intera	fgets	fputs

```
int ch;
ch = fgetc(f); /* ch prossimo carattere del file, EOF se è finito il file */
```

```
int ch;
fputc(ch, f); /* ch carattere da aggiungere */
```

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 41 di 46

Soluzione "x righe":

```
const int LUN=200;
char riga[LUN+1];
...
while (fgets(riga, LUN, f) != NULL) {
    for(i=0; riga[i]!=0; i++) {
        if(isalpha(riga[i])!=0) {
            freq[tolower(riga[i])-'A']++;
        }
    }
}
```

Qualora sia necessario creare file con più campi nella stessa riga è scomodo ricorrere alle funzioni fputc/fputs. restituisce n° byte scritti o EOF
E' possibile utilizzare una variante di fprintf: fprintf(f, "formato", ...
Nel caso della lettura di più campi è possibile usare fscanf(f, "formato", &x...

Schema generale lettura da file
- leggi un dato dal file;
- finchè non è finito il file {
    elabora dato;
    leggi un dato dal file;
  }

restituisce il numero di campi convertiti o EOF se c'è errore

ES
```
res = fscanf(fp, "%d", &val);
while (res != EOF) {
    ...
    res = fscanf(fp, "%d", &val);
}
```
<=>
Versione "compatta"
```
while (fscanf(fp, "%d", &val) != EOF
    ...
}
```

ES Leggere "estremi.dat" contenente coppie di valori (x,y) x riga, scrivere file "d... con la differenza x-y x ogni riga.

```
/*1 acquisizione coordinate bersagli */
f = fopen(FILEB, "r");
Nb = 0
while (fgets(riga, MAX, f) != NULL) {
    r = sscanf(riga "%d %d", &Bx[Nb], &By[Nb]);
    if (r != 2) printf("errore");
    Nb++;
}
fclose(FILE f);
/* 2 coordinate colpi */
if (argc != 2) printf("errore");
f = fopen(argv[1], "r");
Nc = 0;
Ncc = 0;
while (fgets(riga, MAX, f) != NULL) {
    r = sscanf(riga, "%d %d", &Cx, &Cy);
    if (r != 2) printf("errore");
    Nc++;
    /* ricerca bersaglio */
    trovato = 0
    for (i = 0; i < Nb && trovato == 0; i++) {
        if (Cx == Bx[i] && Cy == By[i])
            trovato = 1;
    }
    if (trovato == 1)
    Ncc++;
}
fclose(f);
/* 3 Stampa */
printf("colpi sparati: %d \n", Nc);
printf("colpi a segno: %d", Ncc);
return 0;
}
```

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 45 di 46