

Linguaggi, programmi, compilazione

Stefano Ferrari

Università degli Studi di Milano
stefano.ferrari@unimi.it

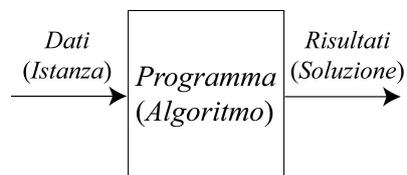
Programmazione

anno accademico 2017–2018

Programmi e problemi

Un programma trasforma **dati** in **risultati**

Un algoritmo trasforma l'**istanza** di un problema nella sua **soluzione**



Poiché un algoritmo è una macchina, occorre

- ▶ **codificare l'istanza** perché l'algoritmo possa **manipolarla**
- ▶ **codificare la soluzione** perché l'algoritmo possa **produrla**

Codifica di istanza e soluzione

Istanza e soluzione sono insiemi di oggetti matematici che modellano parti del mondo reale

Codificarle significa

1. sostituire gli oggetti matematici con **simboli** (numeri, lettere, parole chiave, glifi)
2. raccogliere i simboli in una semplice struttura monodimensionale (**stringa**)

Nel nostro caso istanza e soluzione saranno file di testo

Esempi di codifica

1. Crittografia con sistema algebrico su \mathbb{Z}_{27} (matrice 1×1 : $A = 7$)

- ▶ Istanza (testo in chiaro): **ARCHIMEDE PITAGORICO**
- ▶ Codifica dell'istanza (stringa di simboli):

1	18	3	8	9	13	5	4	5	0	16	9	20	1	7	15	18	9	3	15
---	----	---	---	---	----	---	---	---	---	----	---	----	---	---	----	----	---	---	----

- ▶ Codifica della soluzione (stringa di simboli)

7	18	21	2	9	10	8	1	8	0	4	9	5	7	22	24	18	9	21	24
---	----	----	---	---	----	---	---	---	---	---	---	---	---	----	----	----	---	----	----

- ▶ Soluzione (testo cifrato): **GRUBIJHAH DIEGVXRIUX**

Esempi di codifica

2. Correzione automatica di testi

- ▶ Istanza (testo "scorretto"):

bisogna k fate qlc x' nn si kapisce

- ▶ Codifica dell'istanza (stringa di simboli):

bisogna	k	fate	qlc	x'	nn	si	kapisce
---------	---	------	-----	----	----	----	---------

- ▶ Codifica della soluzione (stringa di simboli)

Bisogna	che	facciate	qualcosa	,	perché	non	si	capisce
---------	-----	----------	----------	---	--------	-----	----	---------

- ▶ Soluzione (testo corretto):

Bisogna che facciate qualcosa, perché non si capisce

Esempi di codifica

3. Interpretazione di immagini

- ▶ Istanza (immagine)



- ▶ Codifica dell'istanza (formato bitmap):

```
1B0: FF FE 0F FF FF FF FF FF FF C7 FC 00 00 00 F8 1F
1C0: FF F8 0F FF FF FF FF FF FF F3 FC 00 00 00 F1 87
1D0: FF E0 C7 FF FF FF FF FF FF F9 FC 00 00 00 FF C3
...
3D0: FF FC
```

- ▶ Codifica della soluzione (stringa di simboli)

A g

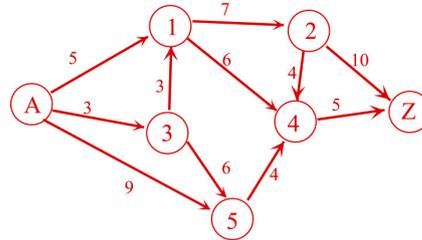
- ▶ Soluzione (elenco degli oggetti riconosciuti)

Una lettera "A" e una lettera "g"

Esempi di codifica

4. Navigazione su strada

- ▶ Istanza (rete stradale, origine, destinazione, costi)



- ▶ Codifica dell'istanza (grafo in formato lista archi)

A	Z	A	1	5	A	3	3	A	5	9	1	2	7	1	4	6	2	Z	10	2	4	4	3	1	3	3	5	6	4	Z	5
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	---	---	---	---	---	---	---	---	---	---	---	---

- ▶ Codifica della soluzione (sequenza dei nodi visitati)

A	1	4	Z
---	---	---	---

- ▶ Soluzione (percorso ottimo)

La prima a sinistra, poi la prima a destra, poi sempre dritto

Esempi di codifica

5. Pianificazione della produzione

- ▶ Istanza (le (dis)equazioni che descrivono l'impianto e la funzione che lega produzione e profitto)

$$\max z = 250x_1 + 350x_2$$

$$2x_1 + 3x_2 \leq 1200$$

$$x_1 \leq 300$$

$$x_2 \leq 400$$

$$x_1, x_2 \geq 0$$

0	-250	-350	0	0	0
1200	2	3	1	0	0
300	1	0	0	1	0
400	0	1	0	0	1

- ▶ Codifica dell'istanza

3	5	0	0	-250	-350	0	0	0	1200	300	400	...
---	---	---	---	------	------	---	---	---	------	-----	-----	-----

Esempi di codifica

5. Pianificazione della produzione

- ▶ Codifica della soluzione

3	5	0	-145 000	0	0	350/3	50/3	0	300	1	...
---	---	---	----------	---	---	-------	------	---	-----	---	-----

- ▶ Soluzione: livelli di produzione ottimali

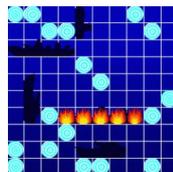
-145 000	0	0	350/3	50/3	0
300	1	0	0	1	0
200	0	0	-1/3	2/3	1
200	0	1	1/3	-2/3	0

$x_1^* = 300$ e $x_2^* = 200$ forniscono $z^* = 145\,000$

Esempi di codifica

6. Battaglia navale

- ▶ Istanza: griglia, posizioni colpite, navi affondate, colpite e mancanti



- ▶ Codifica dell'istanza

0	0	x	x	...	x	x	0	A	A	A	A	A	0	x	...	x	x	0	x
---	---	---	---	-----	---	---	---	---	---	---	---	---	---	---	-----	---	---	---	---

- ▶ Codifica della soluzione

5	2
---	---

- ▶ Soluzione: prossimo bersaglio

$(E, 2)$

Codifica monodimensionale

Perché **una codifica monodimensionale**?

- ▶ L'uomo di solito usa codifiche bidimensionali (grafici)
- ▶ Una macchina potrebbe usare altre codifiche
- ▶ Ma **per una macchina è la codifica più semplice**
(In pratica: legge i dati da file e scrive i risultati su file...)

Codifica binaria, cioè **stringhe composte di 0 e 1**

- ▶ ogni simbolo dell'istanza va tradotto in una stringa binaria
- ▶ ogni sottostringa binaria della soluzione va tradotta in un simbolo

Linguaggio macchina

Ma come si realizza un programma?

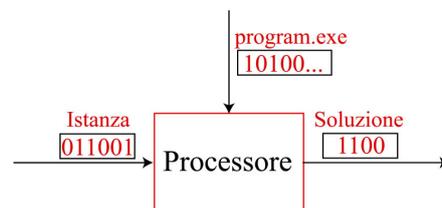
Come si costruisce una macchina che trasforma simboli?

I **processori** (vedi il corso di "Architettura degli elaboratori")
trasformano stringhe binarie in stringhe binarie,
eseguendo istruzioni espresse come stringhe binarie

Linguaggio macchina è la **codifica binaria delle istruzioni**

- ▶ il processore usa direttamente il linguaggio macchina
- ▶ ogni processore ha il suo linguaggio specifico
- ▶ il linguaggio macchina è ostico per un essere umano

Processore



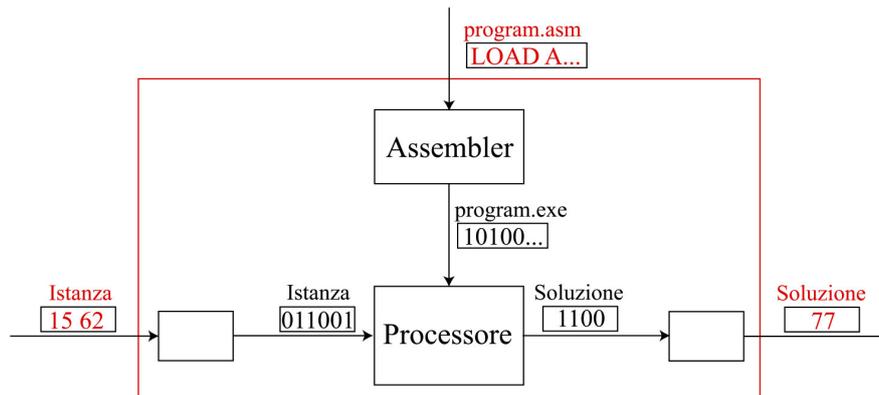
Linguaggio assembly

Linguaggio assembly: per ogni istruzione macchina elementare definisce un'istruzione simbolica corrispondente

1. si scrive un testo che descrive il programma (codice o listato del programma)
2. un programma (**assembler**) traduce il codice in linguaggio macchina

LOAD A	1010...00101
LOAD B	1010...01101
ADD	0011...00101
STORE	1001...10101

Assembler



Linguaggio assembly

Scrivendo in linguaggio assembly

- ▶ si indicano direttamente le istruzioni elementari
- ▶ ma non si lavora con stringhe binarie

Vantaggi:

- ▶ il linguaggio assembly è **più leggibile** per un essere umano
- ▶ si può usare **un solo linguaggio assembly per diverse macchine purché le macchine abbiano le stesse istruzioni elementari** (basta cambiare traduttore)

Linguaggio assembly

Svantaggi

- ▶ anche operazioni banali richiedono più istruzioni

Somma a e b	LOAD A LOAD B ADD STORE
-----------------	----------------------------------

- ▶ il programma è molto lungo
- ▶ il significato del programma non è evidente

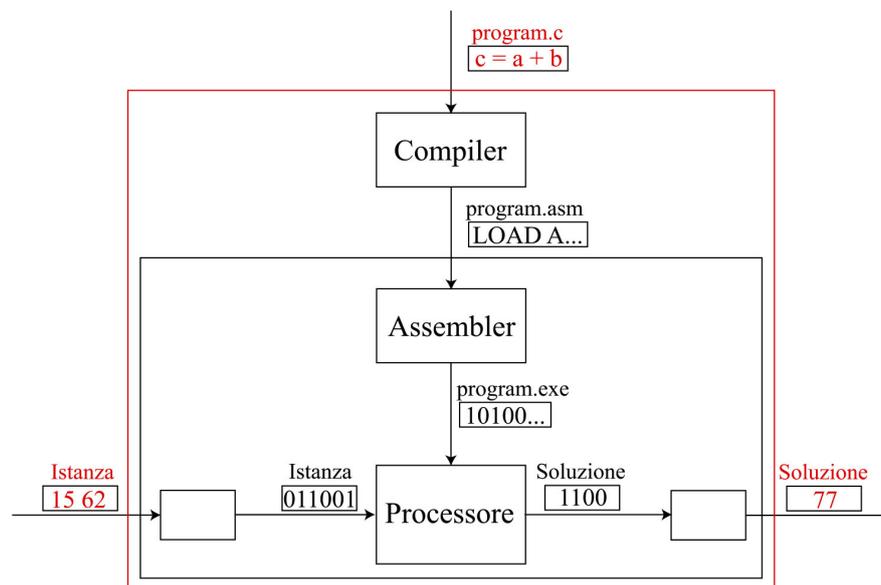
Linguaggio ad alto livello

Linguaggio ad alto livello: definisce una sola istruzione simbolica per ogni sequenza di istruzioni assembly con uno scopo comune

- ▶ si scrive un testo che descrive il programma (codice o listato del programma)
- ▶ un programma (compiler) traduce il codice in linguaggio macchina (di solito passando per l'assembly)

$c = a + b$	LOAD A LOAD B ADD STORE
-------------	----------------------------------

Compilatore



Linguaggi ad alto livello

Ci sono migliaia di linguaggi ad alto livello

Tutti i linguaggi sono equivalenti: consentono di fare le stesse cose

Ogni linguaggio è più adatto per alcuni scopi e meno per altri, cioè consente di fare alcune cose più facilmente e altre meno

Imparare a programmare significa imparare a usare **TUTTI** i linguaggi ad alto livello

Cominceremo con il C...

Il linguaggio C

- ▶ Fu inventato nel 1972 da Dennis Ritchie presso i laboratori della AT&T Bell
- ▶ Serviva a scrivere il codice del sistema operativo UNIX per un processore DEC PDP-11
- ▶ Presto si fecero traduttori da C a linguaggio macchina per poter usare il C su qualsiasi processore (portabilità)
- ▶ Nel 1989 l'American National Standards Institute (ANSI) ha approvato lo standard C89, noto anche come ANSI C
- ▶ L'anno seguente ISO ha adottato lo standard ANSI C
- ▶ Revisioni in 1999 (C99) e 2011 (C11)

Nel seguito useremo sempre lo standard C89

Standard

Nonostante la definizione dello standard C89

- ▶ alcune caratteristiche sono ancora machine-dependent (rischio per la portabilità)
- ▶ ogni compilatore aggiunge estensioni del linguaggio che facilitano la stesura del codice, ma lo rendono non portabile
- ▶ standard successivi includono estensioni ulteriori, talvolta discutibili (per esempio, il C99)

Eviteremo accuratamente quasi tutte le estensioni

Vantaggi del C

- ▶ È **molto espressivo**
 - ▶ offre **diverse strutture di controllo**
 - ▶ l'utente può **definire nuovi tipi di dato**
 - ▶ l'utente può **definire funzioni e procedure**
- ▶ Offre un **accesso diretto al basso livello**: rappresentazione dei dati, operazioni sui bit, gestione della memoria
- ▶ Produce **codice efficiente, compatto e rapido da compilare**
- ▶ Ha **poche parole chiave** e una **sintassi definita formalmente**
- ▶ Favorisce lo **sviluppo top-down** (**raffinamenti successivi**)
- ▶ Ha **molte librerie standard aggiuntive**

Svantaggi del C

- ▶ Ha una **sintassi non chiarissima**, perché è **sintetica** e **consente combinazioni complesse** degli elementi di base
- ▶ Consente di scrivere **codice criptico** ("obfuscated") o con un **significato ambiguo o diverso da quello apparente**
The International Obfuscated C Code Contest, <http://www.ioccc.org/>
- ▶ Alcuni operatori hanno **regole di precedenza controintuitive**

Lo usiamo perché è un **ottimo compromesso** fra

- ▶ la possibilità di **definire istruzioni ad alto livello**
- ▶ la possibilità di **controllare meccanismi a basso livello**

Per garantire chiarezza, portabilità e manutenibilità, eviteremo accuratamente molte cose consentite

hello_world.c

```
main(1
,a,n,d)char**a;{
for(d=atoi(a[1])/10*80-
atoi(a[2])/5-596;n="@NKA\
CLCCGZAAQBEEADAFaISADJABBA^\  
SNLGAQABDAXIMBAACTBATAHDBAN\  
ZcEMMCCCCAAhEIJFAEAAABAfHJE\  
TBdFLDAANEfDNBPHdBcBBBEA_AL\  
H_E_L_L_O,WWO_R_L_D!"
[l++-3];)for(;n-->64;)
putchar(!d+++33^
l&l);}

```

<http://www.ioccc.org/1992/westley.c>

Struttura modulare

Il codice di un programma complesso viene distribuito in più file perché sarebbe troppo lungo, anche se scritto in linguaggio ad alto livello

- ▶ il singolo file è **gestibile** da un essere umano
- ▶ si possono assegnare brani di codice a **gruppi indipendenti**
- ▶ si possono **riusare brani di codice** in altri programmi senza copiarli (**librerie**)
- ▶ si possono tradurre librerie in linguaggio macchina
 - ▶ per **evitare di tradurle ogni volta** col resto del codice
 - ▶ per **poterle vendere** indipendentemente

Struttura modulare

I singoli brani

- ▶ non corrispondono a programmi funzionanti
- ▶ vanno ricollegati in un codice unico prima di eseguirli

La **traduzione** da linguaggio ad alto livello a linguaggio macchina **richiede anche una fase di collegamento** dei vari brani di codice

Occorre **aggiungere ai file le indicazioni su come ricollegarli**

Compilazione

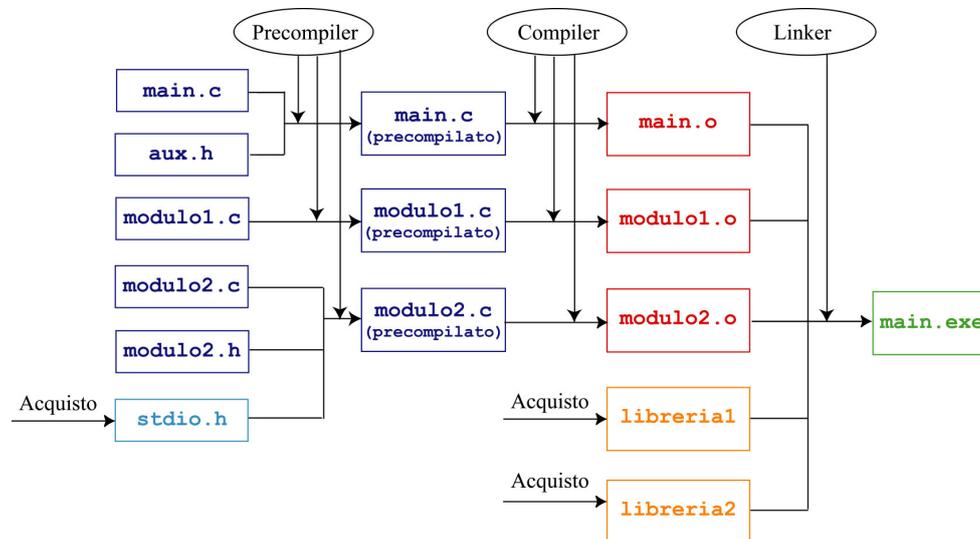
Compilazione è il **processo di traduzione**

- ▶ **dal codice** in linguaggio ad alto livello (uno o più file di testo)
- ▶ **al programma** in linguaggio macchina (un file eseguibile)

Attraversa tre fasi

1. **precompilazione** o **preprocessing** (**da codici a codice**): fonde codici, modifica o cancella brani di codice
2. **compilazione** (**da codice a oggetto**): traduce un file codice in un file oggetto, di solito passando per il codice assembly
3. **collegamento** o **linking** (**da oggetti a eseguibile**): lega file oggetto ed eventuali librerie esterne in un programma

Compilazione



Struttura dei listati C

Un listato C ha una struttura regolare

1. **Direttive**
2. **Prototipi delle procedure secondarie**
3. **Funzione `main`**, o programma principale
 - 3.1 parte **dichiarativa**
 - 3.2 parte **esecutiva**
4. **Definizione delle procedure secondarie**
 - 4.1 parte **dichiarativa**
 - 4.2 parte **esecutiva**

Tale struttura è strettamente legata al processo di compilazione

Commenti

I **commenti** sono **spiegazioni del codice ad uso degli utenti** racchiuse fra **/* e */** (non si possono annidare!)

- ▶ sono **eliminati dal precompilatore** (la macchina non li usa)
- ▶ sono **essenziali per lavorare in gruppo** o **a distanza di tempo**
- ▶ esistono strumenti software che **creano automaticamente il manuale** del programma partendo dai commenti

Commento è anche **il testo compreso fra // e la fine della riga** secondo lo standard C99 (non C89!)

Struttura: direttive

```
/* hello1.c */

#include <stdio.h>

#define BORDO    '*'
#define SPAZIO   ' '
#define LARGHEZZA 17
#define SALUTO   "Hello, world!"

void StampaStringa (char *s);
void StampaCarattere (char c, int num);
void ACapo ();

int main (int argc, char *argv[])
{
    char b;
    int l;
```

Direttive

Cominciano con il carattere speciale **#**

Danno istruzioni al precompilatore

- ▶ **inclusione di altri file** (**#include nomefile**):
si sostituisce la direttiva con l'intero contenuto del file (ricorsivamente)
- ▶ **espansione di macro** (**#define macro valore**):
si sostituisce ogni occorrenza della macro col valore (ricorsivamente)
- ▶ **compilazione condizionale** (**#ifdef macro codice #endif**):
se la macro è definita, il codice permane;
altrimenti, il codice si cancella

Struttura: prototipi di funzione

```
/* hello1.c */

#include <stdio.h>

#define BORDO    '*'
#define SPAZIO  ' '
#define LARGHEZZA 17
#define SALUTO   "Hello, world!"

void StampaStringa (char *s);
void StampaCarattere (char c, int num);
void ACapo ();

int main (int argc, char *argv[])
{
    char b;
    int l;
```

Struttura: prototipi di funzione

Presentano le **funzioni** o **procedure**: brani di codice abbastanza importanti da avere un nome, dei dati e dei risultati, come i programmi

Il prototipo (**dichiarazione**) di una funzione ne **specifica**

1. il **tipo del risultato** (void se non c'è risultato)
2. il **nome** simbolico usato per chiamarla
3. il **tipo e il nome dei dati** (**parametri di ingresso**)

La **definizione** della funzione (il **brano di codice**) è in un'altra sezione

I file *header* inclusi dalla direttiva `#include` spesso contengono prototipi di funzioni definite in librerie esterne

Struttura: main

```
void ACapo ();

int main (int argc, char *argv[])
{
    char b;
    int l;

    /* Stampa il bordo superiore */
    StampaCarattere(BORDO,LARGHEZZA);
    ACapo();

    /* Stampa il bordo laterale sinistro */
    StampaCarattere(BORDO,1);
    StampaCarattere(SPAZIO,1);

    /* Stampa il saluto */
    StampaStringa(SALUTO);

    /* Stampa il bordo laterale destro */
    StampaCarattere(SPAZIO,1);
    StampaCarattere(BORDO,1);
    ACapo();

    /* Stampa il bordo inferiore */
    b = BORDO;
    l = LARGHEZZA;
    StampaCarattere(b,l);

    return 0;
}

/* Stampa la stringa di caratteri "s" */
void StampaStringa (char *s)
```

Struttura: main

L'intestazione presenta anche qui

1. tipo del risultato (sempre int)
2. nome del programma principale (sempre main, per distinguerlo dalle altre funzioni)
3. tipi e nomi dei dati (sempre argc e argv)

La definizione è racchiusa fra parentesi graffe ({})

1. **parte dichiarativa**: introduce le **variabili**, cioè gli **oggetti manipolati dal programma**
2. **parte esecutiva**: introduce le **istruzioni**, cioè le **operazioni compiute dal programma**

Struttura: parte dichiarativa

```
void ACapo ();

int main (int argc, char *argv[])
{
    char b;
    int l;

    /* Stampa il bordo superiore */
    StampaCarattere(BORDO,LARGHEZZA);
    ACapo();

    /* Stampa il bordo laterale sinistro */
    StampaCarattere(BORDO,1);
    StampaCarattere(SPAZIO,1);

    /* Stampa il saluto */
    StampaStringa(SALUTO);

    /* Stampa il bordo laterale destro */
    StampaCarattere(SPAZIO,1);
    StampaCarattere(BORDO,1);
    ACapo();

    /* Stampa il bordo inferiore */
    b = BORDO;
    l = LARGHEZZA;
    StampaCarattere(b,l);

    return 0;

    /* Stampa la stringa di caratteri "s" */
    void StampaStringa (char *s)
}
```

Struttura: parte dichiarativa

Le **variabili** sono **risultati parziali**; di ognuna si specifica

- ▶ il **tipo**, cioè l'**insieme dei valori che può assumere** e le **operazioni che si possono compiere con essa**
- ▶ il **nome**, cioè un **identificatore simbolico usato per manipolarla**

Ogni dichiarazione di variabile termina con **;**

Più variabili di tipo identico possono essere dichiarate insieme

```
int l1, l2;           equivale a  int l1;
                        int l2;
```

Struttura: parte esecutiva

```
void ACapo ();

int main (int argc, char *argv[])
{
    char b;
    int l;

    /* Stampa il bordo superiore */
    StampaCarattere(BORDO,LARGHEZZA);
    ACapo();

    /* Stampa il bordo laterale sinistro */
    StampaCarattere(BORDO,1);
    StampaCarattere(SPAZIO,1);

    /* Stampa il saluto */
    StampaStringa(SALUTO);

    /* Stampa il bordo laterale destro */
    StampaCarattere(SPAZIO,1);
    StampaCarattere(BORDO,1);
    ACapo();

    /* Stampa il bordo inferiore */
    b = BORDO;
    l = LARGHEZZA;
    StampaCarattere(b,l);

    return 0;

    /* Stampa la stringa di caratteri "s" */
    void StampaStringa (char *s)
```

Struttura: parte esecutiva

Le **istruzioni** possono essere

- ▶ operazioni definite nel linguaggio (per es., =)
- ▶ funzioni tratte da librerie standard (per es., `printf`)
- ▶ funzioni definite dall'utente (per es., `StampaStringa`)

Ogni istruzione termina con un punto e virgola (;)

Struttura: procedure

```
/* Stampa il bordo inferiore */
StampaCarattere(BORDO,LARGHEZZA);

return 0;
}

/* Stampa la stringa di caratteri "s" */
void StampaStringa (char *s)
{
    printf("%s",s);
}

/* Stampa il carattere "c"
ripetuto "num" volte */
void StampaCarattere (char c, int num)
{
    int i;

    for (i = 1; i <= num; i++)
        printf("%c",c);
}

/* Va a capo */
void ACapo ()
{
    printf("\n");
}
```

Struttura: procedure

La **definizione** è il **brano di codice preannunciato dal prototipo**

Ha la stessa struttura del main

- ▶ intestazione: copia del prototipo (senza ;)
- ▶ parte dichiarativa: tipo e nome delle variabili
- ▶ parte esecutiva: istruzioni