Lezione 15 e 16

- Programmazione strutturata
- Progettazione del codice di un programma complesso

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 15 e 16

Programmazione strutturata

Obiettivo:

- Progettazione del codice di un programma complesso

Programmazione strutturata

- **La programmazione strutturata è la tecnica di programmazione che**
  - ha lo scopo di semplificare la scrittura dell'algoritmo
  - utilizza i tre seguenti requisiti:
    - solo sequenze, scelte e iterazioni
    - un ingresso ed un'uscita per ogni blocco
    - un blocco può essere a sua volta una struttura

3

1. Solo sequenze, scelte e iterazioni

4

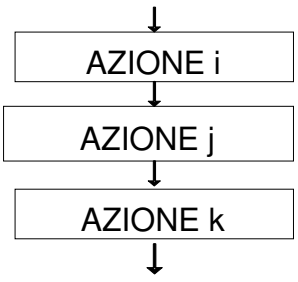
Solo sequenze, scelte e iterazioni

- **Il primo requisito impone che per la realizzazione dell'algoritmo e del programma vengano impiegate solo tre strutture di controllo fondamentali:**
  - Sequenza;
  - Scelta (chiamata anche selezione);
  - Iterazione.

5

Sequenze

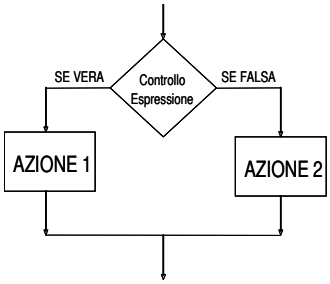
Nome costrutto	In linguaggio C	Negli schemi a blocchi

Sequenza	`azione_i;`
	`azione_j;`
	`azione_k;`
		

6

Scelta

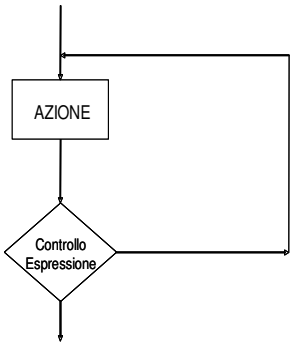
Nome costrutto	In linguaggio C	Negli schemi a blocchi

Scelta (o selezione)	`if(Espressione)`
	`{`
	`  azione_1;`
	`}`
	`else`
	`{`
	`  azione_2;`
	`}`
		

7

Iterazione (1)

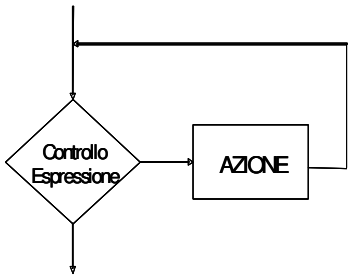
Nome costrutto	In linguaggio C	Negli schemi a blocchi

Iterazione	`do`
	`{`
	`  AZIONE;`
	`}while(Espr)`
		

8

Iterazione (2)

Nome costrutto	In linguaggio C	Negli schemi a blocchi

Iterazione	`while(Espr)`
	`{`
	`  AZIONE;`
	`}`
		

9

Completezza (1)

- **Il teorema di Jacopini-Bohm (1966) ci assicura che**
  - le tre strutture di controllo fondamentali formano un insieme di strutture completo,
  - cioè tramite le quali si possono descrivere tutti gli algoritmi.

Teorema*: Ogni algoritmo può essere espresso con le *sole* tre strutture di controllo fondamentali *sequenza*, *selezione* e *iterazione

10

Completezza (2)

- **Esprimendo gli stessi concetti usando i costrutti in C:**
  - ogni algoritmo può essere espresso mediante sequenze di istruzioni, `if` e `while` (oppure `do-while`).
- **In linea teorica (ed anche pratica) nemmeno il ciclo `for` sarebbe necessario.**
- **In generale, l'uso di salti (chiamati non condizionati) nei programmi rende la programmazione non strutturata:**
  - non vanno assolutamente impiegati

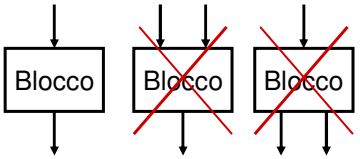
Completezza (3)

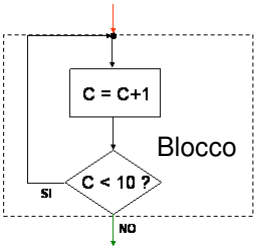
- **Nel nostro corso usiamo soltanto i costrutti**
  - **`if` e `case`**
  - **`while`, `do-while` e `for`**
  - le sequenze di istruzioni
- **In nessun caso useremo altri costrutti**

2. Un ingresso ed una uscita per ogni blocco

Un ingresso ed una uscita per ogni blocco

- **Il secondo requisito della programmazione strutturata impone che**
  - ogni struttura di controllo dell'algoritmo, immaginata mediante il formalismo degli schemi a blocchi, debba essere **un blocco con una sola freccia in entrata ed una sola in uscita.**



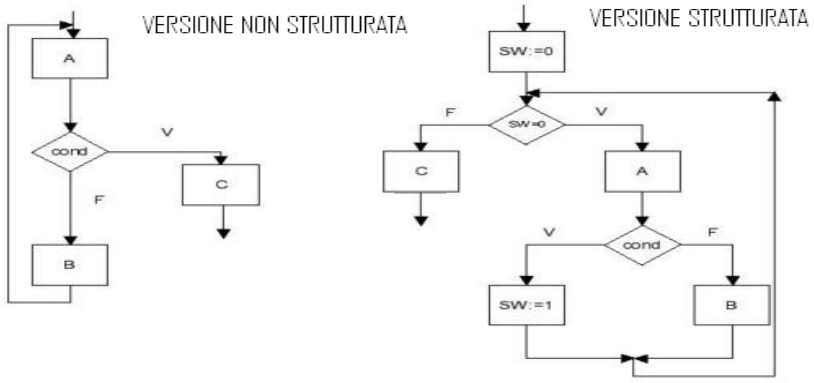


Esempio (1)

- **L'applicazione del primo e del secondo requisito, è apparentemente banale.**
- **Esempio: progettare un diagramma a blocchi che:**
  - esegue l'azione A
  - verifica la condizione cond
    - se cond è vera esegue l'azione C e termina
    - se cond è falsa esegue l'azione B e ricomincia il programma daccapo

15

Esempio (2)



- A sinistra il costrutto per realizzare l'iterazione non è di quelli specificati per la programmazione strutturata
- Lo schema a blocchi a destra soddisfa i requisiti della program strutturata, usando una variabile *SW* inizializzata a zero.

16

3. Un blocco può essere a sua volta una struttura

Un blocco può essere una struttura

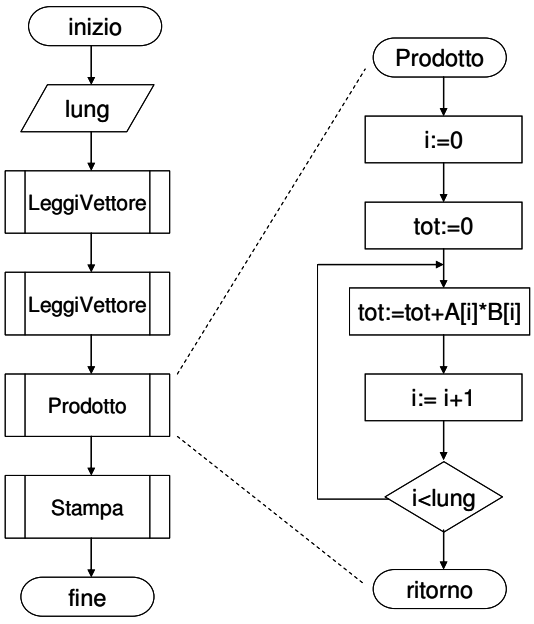
- **Ogni blocco interno ad una struttura di controllo**
  - non è detto che sia semplice (singola istruzione)
  - ma può essere a sua volta una struttura
- **Come abbiamo già visto nel corso, questo concetto è legato alle tecniche di risoluzione top-down.**

Esempio (1)

- **Esempio: definiamo uno schema a blocchi che implementa**
  - un algoritmo per il calcolo del prodotto scalare di due vettori di numeri A e B di lunghezza lung elementi.
- **La soluzione è composta da soluzioni di sottoproblemi:**
  - la lettura del vettore A di lung elementi
  - la lettura del vettore B di lung elementi
  - il calcolo del prodotto scalare degli elementi
  - la stampa del prodotto scalare calcolati

19

Esempio (2)



20

Esempio (3)

- **La soluzione proposta mostra la decomposizione del**
  - sottoprogramma Prodotto che si occupa di calcolare il prodotto scalare di due vettori
- **Gli altri sottoprogrammi LeggiVettore e Stampa posso essere sviluppati in modo analogo**

21

Esempio (4)

- **La soluzione presentata nella forma di uno schema a blocchi può essere presentata in modo equivalente**
  - scrivendo un programma C con un main che chiama altre funzioni
- **Consideriamo come ben scritto un codice che ha un main snello, ovvero composto solo dalla chiamata di alcune principali funzioni**

22

Esempio (5)

```
int main()
{       …
        scanf("%d", &lung);
        LeggiVettore(A, lung);
        LeggiVettore(B, lung);
        tot = Prodotto (A, B, lung);
        Stampa(tot);
        …
} // fine del main
// definizione della funzione Prodotto
float Prodotto( float A[], float B[], int lung)
{
        int i;
        float tot;
        i = 0;
        tot = 0;
        do{
                tot = tot + A[i] * B[i];
                i = i + 1;
        }while(i<lung)
        return tot;
}
```


Programmazione strutturata: vantaggi (1)

- **La programmazione strutturata non è un puro esercizio di programmazione ma porta dei vantaggi diretti:**
  - maggiore facilità di uso di tecniche top-down
  - algoritmi più leggibili
  - maggiore facilità di individuazione degli errori
- **La programmazione strutturata**
  - non è la più moderna metodologia per scrivere programmi ma è da considerarsi come una delle prime
  - è riconosciuta ancora oggi come una delle fondamentali.

Programmazione strutturata: vantaggi (2)

- **I concetti della programmazione strutturata**
  - sono stati subito impiegati nell'industria
  - sono stati recepiti ormai da tutti i linguaggi di programmazione moderni
- **I vantaggi descritti non sono ancora soddisfacenti, e quindi sono state inventate altre metodologie per la creazione di programmi con vantaggi ancora maggiori per il programmatore come**
  - la programmazione modulare
  - e successivamente la programmazione ad oggetti

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 15 e 16

Esempio di programma

Obiettivo:

- Progettazione del codice di un programma complesso

Obiettivi

- **Comprendere che la corretta progettazione dei tipi di dato e delle funzioni è il cuore delle programmazione strutturata.**
- **Approfondire la stesura in modalità top-down del programma** (dalla scrittura delle intestazioni delle funzioni alla loro definizione)**.**
- **Essere in grado di scrivere un programma che simuli l'evoluzione di una colonia di microbi in un ambiente.**

27

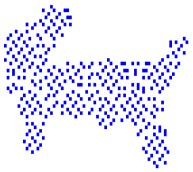
Life: le caratteristiche

- **Si sviluppa in un "universo" bidimensionale popolato da cellule che nascono, si riproducono e muoiono secondo determinate regole.**
- **In letteratura questo tipo di simulazione è noto con il nome di automi cellulari.**
- **Sono simulazioni molto conosciute: sul motore www.google.it digitare "`automi cellulari 2D`" e scorrere le pagine dei risultati.**

28

Life : l'ideatore

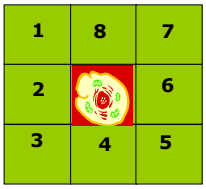
- Ideato da **John Horton Conway**, uno dei più noti matematici contemporanei.
- Conway è attivo in molteplici campi: teoria dei codici correttori d'errore, teoria dei gruppi, teoria dei giochi, ecc.



29

Life: le regole del gioco (1)

- **La vita di ogni cellula è determinata dalla presenza di altre cellule negli 8 quadretti che la circondano:**



- **Regola della nascita: se 3 cellule attive sono adiacenti ad un quadretto vuoto, in quel quadretto nascerà una nuova cellula nella successiva generazione.**

30

Life: le regole del gioco (2)

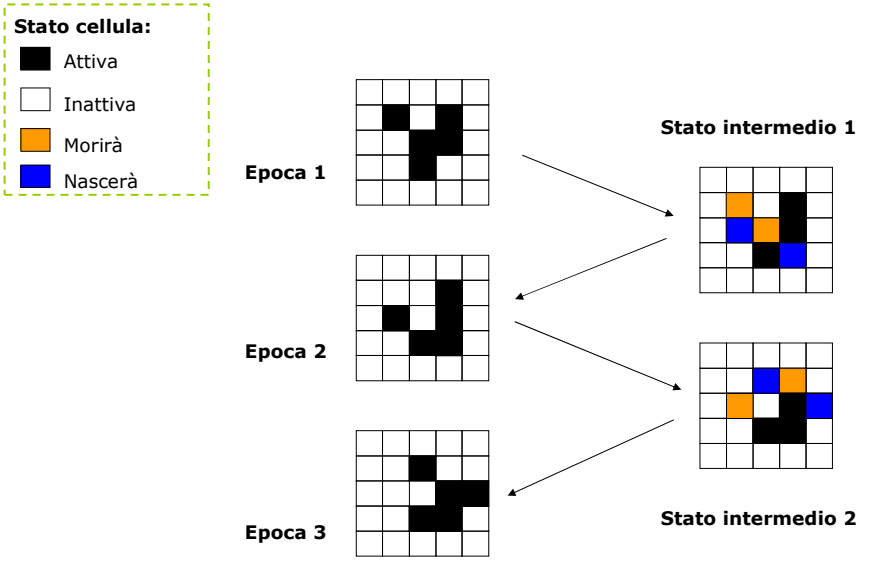
- **Regola di sopravvivenza: una cellula viva continua a vivere anche nella successiva generazione se intorno ad essa ci sono 2 o 3 cellule vive.**

- **Regola d'inedia (o sovrappopolazione): se una cellula viva ha meno di 2 cellule adiacenti vive o più di 3 vive, essa morirà per inedia o sovrappopolazione.**

una grandissima varietà di situazioni e di comportamenti è racchiusa in queste (semplici) regole.

31

Life: un esempio



32

Primo passo: progettazione del codice

Chiedersi chi sono gli attori e le azioni.

- Attori/oggetti/variabili:
  - non singole cellule o batteri ma una matrice che descrive il vetrino;
  - un contatore delle epoche.
- Azioni/metodi/funzioni:
  - funzione per inizializzare la matrice;
  - funzione per modificarla;
  - funzioni per stamparla.

Utilizzare una libreria `strangeLife.h` contenente tutte le funzioni ed i prototipi.

33

Stesura main

```
//life.c
#include <stdio.h>
#include "strangeLife.h"    // usare "  "  per il vostro file

#define MAX_EPOCHE 400

int main()
{

    int epoche;            // le epoche segnano il tempo che passa
    int a[N_MAX][N_MAX];   // ecco la matrice dei microbi

    riempiMatrice(a);      // la PROCEDURA riempie la matrice
    stampa_Matrice(a);     // la PROCEDURA stampa la matrice


    for (epoche=1; epoche < MAX_EPOCHE; epoche++)
    {
         aggiorna_Matrice(a);  // la PROCEDURA modifica la matrice
         stampa_Matrice(a);    // la PROCEDURA stampa la matrice

         getchar();
    }
    exit(0);
}
```

File **life.c**

34

Include e definizioni utili

```
// strangeLife.h

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define N_MAX 30          // lato della matrice
#define ELEM_RANGE2 2     // concentrazione in matrice di cellule
```

File **strangeLife.h**

Prototipi

I prototipi delle funzioni che servono nel main :

```
// prototipi

void riempi_Matrice(int a[][N_MAX]);

void stampa_Matrice(int a[][N_MAX]);

void aggiorna_Matrice(int a[][N_MAX]);
```

File **strangeLife.h**

RiempiMatrice

```
void riempiMatrice(int a[][N_MAX])
{
    int i, j, num_random;
    float temp;
    int n = N_MAX; // per comodita' chiamo n N_MAX

    // Genero una sequenza casuale per riempire la matrice
    srand(time(NULL));
    for(i=0;i < n;i=i+1)
    {
         for (j=0;j < n;j=j+1)
         {
                  // qui genero un numero casuale
                  num_random = rand();
                  // ora scalo il numero casuale fra 0 e 1
                  temp =  (num_random * ELEM_RANGE2) / RAND_MAX  ;
                  // non invertire l'ordine dei fattori!!!!

                  a[i][j]= (int)floor( temp );
                  // il float ritorna ad essere un intero
                  if ( a[i][j] ) a[i][j]=1;
                  }
    }
}
```

File **strangeLife.h**

37

StampaMatrice (1)

```
void stampa_Matrice(int a[][N_MAX]) // notare che qui non si usa il ;
{
    int i;
    int j;
    int n = N_MAX; // per comodita' chiamo n N_MAX

    //stampo una matrice facendo anche i "bordini"
    // --------   Quindi una riga  (1)
    // |      |   Un carattere "|" all'inizio ed un "|" alla fine (2)e(3)
    // |      |   idem
    // --------   Una riga per chiudere  (4)

    for(i=0;i < n+2;i=i+1) printf("_");

                                  // (1) una riga lunga quanto una matrice
```

File **strangeLife.h**

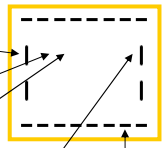
38

StampaMatrice (2)

```
    for(i=0;i < n;i=i+1)
    {
         printf("\n");  // andiamo a capo
         printf("|");   // "|" all'inizio e (2)

         for (j=0;j  < n;j=j+1)
         {
                  if (a[i][j]==1)
                  {
                           printf("@");
                  }
                  else
                  {
                           printf(" ");
                  }
         }
         printf("|");              // "|" alla fine e (3)
    }
    printf("\n");                  // andiamo a capo
    for(i=0;i < n+2;i=i+1) printf("_");
                                   // (4) una riga lunga quanto una matrice
    printf("\n\n");

}
```

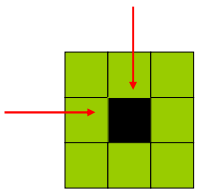


File **strangeLife.h**

AggiornaMatrice (1)

```
void aggiorna_Matrice (int a[][N_MAX])
{
    int i, j, ki, kj ,tot;
    int n = N_MAX; // per comodita' chiamo n N_MAX
    int stato[N_MAX][N_MAX];

    for(i=1;i < n-1;i=i+1)
    {
         for (j=1;j < n-1;j=j+1)
         {
             // ispeziono le 9 celle della matrice 3x3
             // e sottraggo la cella centrale
             tot = 0;
              for (ki=i-1; ki < =i+1;ki=ki+1)
             {
                  for (kj=j-1; kj < =j+1;kj=kj+1)
                  {
                                   tot = tot + a[ki][kj];
                  }
              }
             tot = tot - a[i][j] ;  // guardo solo l'intorno
```



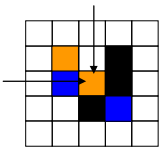
File **strangeLife.h**

AggiornaMatrice (2)

```
            // applico le regole di Conway, (ma sono giuste?)
            if ((a[i][j] == 0) && (tot == 3))
                    {
                        stato[i][j] =1;
                    }else if((a[i][j] == 1) && ((tot == 3) || (tot == 2)))
                    {
                        stato[i][j] =1;
                    }else{
                       stato[i][j] =0;
                    }
        }
    }

    // copio lo stato sulla matrice a
    for(i=1;i < n-1;i=i+1)
    {
         for (j=1;j < n-1;j=j+1)
         {
         a[i][j] = stato[i][j];
        }
    }

}
```



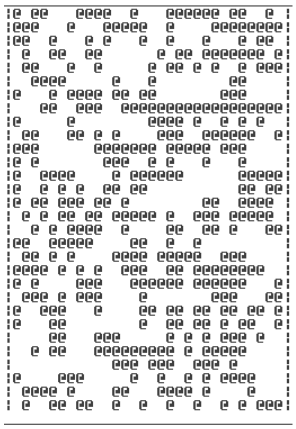
File **strangeLife.h**

Nota sull'uso dello stato

- **E' necessario notare che `a` viene solo letta mentre `stato` viene solo scritta.**
- **Solo alla fine la matrice `stato` viene INTERAMENTE copiata in `a` allo scopo di modificare la situazione delle cellule in parallelo.**
- **Non si vuole infatti leggere la situazione di una cellula e cambiare il suo stato prima che anche tutte le altre cellule siano state a loro volta esaminate.**

Piccolo errore

Solo dalla simulazione è possibile notare che vi è un errore nella gestione dei bordi della matrice (errata applicazione delle regole per effetto dei bordi):



Anche dopo moltissime epoche