Programmazione 1

Un **algoritmo** è un procedimento che risolve un determinato problema attraverso un numero finito di passaggi. La programmazione è la traduzione di un algoritmo in un programma.
Le fasi del ciclo di vita di un programma sono:

- Analisi e Progettazione.
- Relizzazione.
- Verifica.

La **metodologia** adotta quattro passaggi dopo i quali è possibile scrivere l'effettivo codice, questi ultimi sono:

- Analisi **input.**
- Analisi **output.**
- Procedimento **risolutivo.**
- Realizzazione **flow chart.**

Per realizzare un buon flow chart bisogna seguire due regole:

- Deve avere un ingresso e un'uscita.
- Devono poter integrare altri diagrammi al loro interno.

Storia del C

Il computer comprende solo **0** e **1**, i linguaggi di programmazione devono adattarsi a questa esigenza.
Nel 1973 Dennis Ritchie crea il linguaggio C, C++ è la sua evoluzione con paradigmi orientati agli oggetti.

Per sviluppare nel linguaggio C, servono varie componenti, al livello più alto dobbiamo avere un editor di testo. Esistono due tipologie: **VI** o **IDE** (Integrated Development Enviroment). Al livello sottostante servono: **preprocessore** e **compilatore** ed il **linker** che serve per collegare le varie librerie. Gli ultimi due passi sono il **loader** e l'**esecuzione** dalla cpu.

Primo programma in C

```
#include <stdlib.h>
```

```
int main(int argc, char** argv) {
{
    printf("Hello World!");
    return 0;
}
// Output: Hello World!
```

Blocchi e Annidamenti

Un costrutto fondamentale nella programmazione è l'inserimenti di blocchi condizionali, in base a una condizione cambia il flusso del programma.
Esempio: Scrivere :"Esame passato!" se l'esame ha un voto
≥ 60.

```
#include <stdio.h>

int main(int argc, char **argv) {
    int voto = 60;
    if(voto >= 60) {
        printf("Esame passato!");
    }
    else {
        printf("Esame NON passato!");
    }
    return 0;
}
```

Per poter controllare il flusso di un programma dobbiamo usare la la keyword `if` seguito da una condizione. Per controllare il voto dobbiamo fare uno `spaghetti code`.

```
#include <stdio.h>

int main(int argc, char **argv) {
    int voto = 0;
    printf("Inserisci un voto: ");
    scanf("%d", &voto);
    if(voto <= 100 && voto >= 90) {
        printf("A");
    }
    /* Continua con tutti gli altri... */
```

```
    return 0;
}
```

> **Nota bene**: il prof vuole dei collegamenti a **cascata** dopo una serie di **if, non** una linea che colleghi tutto.

Inviarianti di ciclo

> Un **invariante di ciclo** per un costrutto iterativo è una proposizione riguardante le variabili modificate nel ciclo la quale:

- è vera immediatamente prima di eseguire il costrutto iterativo.
- è vera dopo ogni ripetizione del corop dell'iterazione.
- è vera all'uscita del costrutto iterativo, cioè all'uscita definitiva del ciclo.

Bisogna quindi scrivere qual'è la condizione del ciclo.

Complessità degli algoritmi

Per complessità di un algoritmo non si intende il tempo fisico impegato dall'algoritmo a livello fisico, ma a livello logico ovvero il numero di istruzioni da eseguire.

O grande. In generale in informatica non si è interessato ad uno studio preciso della complessità degli algoritmo ma piuttosto il loro comportamento.

Ecco i comportamenti della notazione:

- `O(1)` quanto vi è una costante k.
- `O(n)` quando vi è una una somma di una costante a un numero $n + k$.
- `O(n)` quando vi è una moltiplicazione di nua costante a un numero $n \times k$.
- $O(n^1)$ quando vi sono somme di valori costanti $k_1 \times n^i + k_2 \times n^{i+1}$.

Ecco un esempio in programmazione:

```
#include <stdio.h>

int main(int argc, char **argv) {
    int n;
  int i = 1;
  scanf("%d", &n);
  while(i <= 100) {
```

```
      printf("%d\n", n);
      i = i + 1;
  }
  return 0;
}
```

In questo caso l'algoritmo ha una complessità di $O(1)$ dato che abbiamo un numero costanti di operazioni.

Un altro esempio è $O(n)$ in quanto abbiamo a che fare con una moltiplicazione tra variabili

```
#include <stdio.h>

int main(int argc, char **argv) {
    int n; i = 1, fattoriale = 1;
  scanf("%d", &n);
  while(i <= n) {
        printf("%d\n", fattoriale);
        fattoriale = fattoriale * i;
    i = i + 1;
  }
  return 0;
}
```

A seconda di quanto è grande l'input ci saranno più moltiplicazioni, rispetta all'esempio prima che doveva solo stampare una costante.

L'ultimo caso si presenta quando incapsuliamo il blocco $O(n)$ in n operazioni, generando così un blocco di complessità $O(n^2)$.

```
#include <stdio.h>

int main(int argc, char **argv) {
    int n, i = 1, j = 1, fattoriale = 1, somma_fattoriale = 0;
  scanf("%d", &n);
  while(i <= n) {
        j = 1;
        fattoriale = 1;
        while(j <= i) {
            fattoriale = fattoriale * j;
            j = j + 1;
        }
```

```
    printf("%d\n", fattoriale);
        somma_fattoriale = somma_fattoriale + fattoriale;
    i = i + 1;
  }
  return 0;
}
```

Vettori con tappo

Quando parliamo di vettore, sappiamo che è l'insieme di elementi di vario genere, per esempio il vettore `[1, 2, 3, 4, 5, 6]` è un vettore di interi mentre `['c', 'i', 'a', 'o']` è di caratteri. Se vogliamo delimitare un vettore dobbiamo usare un "tappo" ovvero un vettore che nella posizione finale ha **-1** fondamentale per la riuscita del problema.

Per esempio `[1, 2, 3, 4, -1]` è un **vettore con tappo** di interi.

10	5	23	4	-1	

Come possiamo vedere il vettore non è riempito totalmente ed è questa un'altra funzione del -1, deliminarlo anche prima della sua dimensione totale che in questo caso la massima è 6 ma per il -1 è 5.

Bisogna tenere a mente che per usarli bisogna sempre fare a capo alla condizione `while(condizione_1 && (vettore[i] != -1))` in questo modo possiamo operare senza incorrere in errori non previsti.

Vettori X-Padded e 0-Padded

Usiamo questa tipologia di vettori quando dobbiamo arrivare alla fine del più lungo.

5	3	4	2	-1

3	7	-1		

Se volessimo arrivare alla fine del più lungo dobbiamo capire come trattare gli argomenti mancanti attraverso due tecniche: **0-padded** o **X-padded**. Nel primo caso consideriamo le celle mancanti come "0" non presenti nel vettore realmente ma solamente come controllo. Mentre nel secondo caso consideriamo il valore con cui fare la somma, il confronto o altre operazioni come un una **X variabile prensa in input.**

Inviariante di ciclo

L’invariante è così chiamata perchè è una condizione che presa durante lo svolgimento del ciclo stesso è sempre vera e verificata come una formula matematica.

Per individuarla dobbiamo capire cosa vogliamo ottenere nello stato del ciclo, per esempio la somma continua a ogni ciclo o la posizione da 0 alla posizione raggiunta di un vettore.

```
int main(int argc, char *argv[])
{
        int i = 0, sum = 0;
        while(i < 10) {
            sum += 5;
        }
        printf("La somma e': %d", sum);
    return 0;
}
```

In questo caso l’invariante di ciclo è la somma degli elementi fino all’ **i-esimo-1,** dato che deve essere una condizione vera sia prima che dopo il ciclo.

> Pagina nuovo argomento: **algoritmi**.
>
> Algoritmi

Esercizio 1: Livello Semplice

Descrizione:

Crea un flowchart che calcola la somma dei numeri da 1 a 5, e poi ne trova la media.

Istruzioni:

1. Inizializza una variabile per la somma a 0.
2. Utilizza un ciclo che iteri da 1 a 5.
3. Somma il valore corrente al totale della somma.
4. Dopo il ciclo, calcola la media dividendo la somma per 5.
5. Mostra la somma e la media come output.

Esercizio 2: Livello Medio

Descrizione:

Crea un flowchart che legge 3 numeri per 4 studenti, ne calcola la somma e la media per ciascuno studente e mostra i risultati.

Istruzioni:

1. Utilizza un ciclo esterno per iterare su 4 studenti.
2. All'interno del ciclo, crea un ciclo annidato per leggere 3 numeri per ogni studente.
3. Somma i numeri inseriti per ogni studente.
4. Dopo aver raccolto tutti i numeri per uno studente, calcola la media dividendo la somma per 3.
5. Mostra la somma e la media per ciascuno studente prima di passare allo studente successivo.

Esercizio 3: Livello Difficile

Descrizione:

Crea un flowchart che calcola la somma e la media di 5 numeri per ciascuna delle 3 classi di studenti. Dopo aver completato tutti i calcoli, trova la media delle somme di tutte le classi.

Istruzioni:

1. Utilizza un ciclo esterno che itera sulle 3 classi.
2. All'interno di questo ciclo, crea un ciclo annidato per sommare i 5 numeri forniti per ciascuna classe.
3. Calcola la somma e la media per ciascuna classe.
4. Dopo il ciclo per le classi, calcola la media delle somme delle 3 classi.
5. Mostra le somme e le medie per ciascuna classe, e la media delle somme complessive.

Questi esercizi ti aiuteranno a padroneggiare i concetti di cicli annidati e operazioni di somma e media tramite flowchart.