Fisica 2

Angelo Perotti

1 Riassunto: Algebra di Boole, Introduzione al C e Codifica Algoritmi

1.1 1. Algebra di Boole

1.1.1 Concetti Fondamentali

- **Operatori logici binari**: operano su valori VERO (1) e FALSO (0)
- **Tre operazioni base**:
  - **NOT (unario)**: `¬A` o `!A` - inversione del valore
  - **AND (binario)**: `A  B` - vero solo se entrambi gli operandi sono veri
  - **OR (binario)**: `A  B` - vero se almeno uno degli operandi è vero

1.1.2 Proprietà Matematiche

- **Commutativa**: `A OR B = B OR A`, `A AND B = B AND A`
- **Distributiva**: `A AND (B OR C) = (A AND B) OR (A AND C)`
- **Precedenza operatori**: NOT → AND → OR

1.1.3 Leggi di De Morgan

1. `A AND B = NOT ((NOT A) OR (NOT B))`
2. `A OR B = NOT ((NOT A) AND (NOT B))`

1.1.4 Concetti Avanzati

- **Tautologia**: espressione sempre vera (es. `A OR NOT A`)
- **Contraddizione**: espressione sempre falsa (es. `A AND NOT A`)
- **Equivalenza**: due espressioni con identica tabella di verità

1.2 2. Linguaggio C - Fondamenti

1.2.1 Caratteristiche del C

- **Linguaggio di medio livello**: bilanciamento tra astrazione e controllo hardware
- **Case sensitive**: distinzione tra maiuscole e minuscole
- **Compilato**: tradotto in codice macchina prima dell'esecuzione
- **Portabile**: codice trasferibile tra diverse macchine

1.2.2 Struttura del Programma

[] #include <stdio.h> int main(int argc, char *argv[]) { // corpo del programma return 0; }

1.2.3 Elementi Sintattici

- **Identificatori**: nomi per variabili, funzioni (lettere, cifre, underscore)
- **Keywords**: 32 parole riservate del linguaggio
- **Commenti**: `/* multi-linea */` o `// singola linea`

1.2.4 Sistema di Tipi e Variabili

- **Dichiarazione**: `tipo identificatore [= valore];`
- **L-value**: locazione di memoria (sinistra assegnazione)
- **R-value**: valore contenuto (destra assegnazione)
- **Costanti**: `const tipo identificatore = valore;`

1.2.5 Operatori

Aritmetici

- Base: `+`, `-`, `*`, `/`, `%` (modulo)
- Assegnazione composta: `+=`, `-=`, `*=`, `/=`, `%=`
- Incremento/decremento: `++`, `--` (pre/post fisso)

Precedenza Operatori (dal più alto al più basso)

1. Parentesi `()`, array `[]`
2. Unari `!`, `++`, `--`
3. Moltiplicativi `*`, `/`, `%`
4. Additivi `+`, `-`
5. Relazionali <, <=, >, >=
6. Uguaglianza `==`, `!=`
7. AND logico `&&`
8. OR logico ||
9. Operatore ternario `?:`
10. Assegnazione `=`, `+=`, etc.

1.3 3. Strutture di Controllo

1.3.1 Istruzioni Condizionali

if-else [] if (condizione) { // istruzioni se vero } else { // istruzioni se falso }

Operatore Ternario [] risultato = condizione ? valore_se_vero : valore_se_falso;

Ambiguità if-else

- L'`else` si associa sempre all'`if` più vicino
- Uso delle parentesi graffe per disambiguare

1.3.2 Istruzioni Iterative

Ciclo while [] while (condizione) { // corpo del ciclo // aggiornamento variabili controllo }
Esecuzione:

1. Valutazione condizione
2. Se vera: esecuzione corpo + ritorno al passo 1
3. Se falsa: uscita dal ciclo

1.3.3 Istruzioni Composte

- Raggruppamento di più istruzioni con {}
- Equivalenti a singola istruzione
- Non richiedono ; dopo la parentesi chiusa

1.4 4. Input/Output

- **Input**: `scanf()` per leggere da tastiera
- **Output**: `printf()` per scrivere su schermo
- **Caratteri**: `getchar()` e `putchar()`
- **EOF**: costante per fine file

1.5 5. Esempi di Algoritmi Implementati

1.5.1 Massimo Comune Divisore

Metodo 1 - Definizione: ricerca divisori comuni fino al minimo **Metodo 2 - Euclide**: sottrazione iterativa `MCD(n,m) = MCD(n-m,m)`

1.5.2 Moltiplicazione Binaria

Algoritmo che usa solo somme e moltiplicazioni/divisioni per 2

1.5.3 Problema delle Scale

Calcolo del numero di modi per salire una scala con passi di 1, 2 o 3 gradini: `S(n) = S(n-1) + S(n-2) + S(n-3)`

1.6 6. Processo di Compilazione

1. **Edit**: scrittura codice sorgente
2. **Preprocess**: elaborazione direttive `#include`
3. **Compilation**: traduzione in codice oggetto
4. **Link**: collegamento librerie
5. **Load**: caricamento in memoria
6. **Execute**: esecuzione programma

1.7 7. Note Teoriche Importanti

1.7.1 Macchina Astratta

- **Concetto**: astrazione dell'hardware che nasconde dettagli implementativi
- **Interprete**: componente essenziale che esegue il ciclo fetch-decode-execute
- **Livelli di astrazione**: bilanciamento tra facilità programmazione ed efficienza

1.7.2 Compilazione vs Interpretazione

- **Compilazione**: traduzione completa prima dell'esecuzione (maggiore efficienza)
- **Interpretazione**: traduzione ed esecuzione simultanee (maggiore flessibilità)

1.7.3 Standard e Portabilità

- **ANSI C (1989)**: primo standard ufficiale
- **C99**: revisione del 1999
- **Portabilità**: codice eseguibile su diverse architetture hardware

2 Array in C: Teoria e Concetti Fondamentali

2.1 8. Definizione e Motivazioni

2.1.1 Variabili Strutturate

- Gli **array** sono un arricchimento della macchina astratta C dopo le istruzioni
- Rappresentano **variabili che memorizzano insiemi di elementi** definiti da una relazione
- Analoghe ai vettori e matrici in matematica
- **Esempio**: insieme dei numeri di matricola degli studenti

2.1.2 Caratteristiche Fondamentali

- **Dato strutturato** con modalità di accesso pre-definite per celle di memoria
- **Sequenza di celle consecutive e omogenee** in memoria
- Ogni variabile nell'array è accessibile tramite un **indice** (intero 0)

2.2 9. Implementazione in C

2.2.1 Struttura Base

[] int a[100]; // Dichiarazione di array di 100 interi

Caratteristiche dell'Implementazione:

- **Contenitore** di variabili dello stesso tipo
- **Nome unico** (identificatore) + indice tra parentesi quadre `[]`
- **Accesso agli elementi**: `a[i]` per l'elemento in posizione i
- **Primo elemento**: sempre a indice 0 (`a[0]`)
- **Range di indici**: da 0 a n-1 per array di n elementi

2.2.2 Operatori e Precedenza

- Le **parentesi quadre** `[]` hanno alta precedenza (come le parentesi tonde)
- Associano da sinistra a destra
- Tra parentesi quadre può esserci qualsiasi espressione che restituisce un intero

---

2.3 10. Allocazione di Memoria

2.3.1 Memoria Fisica

- Gli elementi sono memorizzati in **celle consecutive**
- Esempio di allocazione:

```
Elemento: a[0] a[1] a[2] a[3] ...Indirizzo: 1000 1001 1002 1003 ...
```

2.3.2 Accesso agli Elementi

La macchina astratta accede tramite:

1. **Calcolo del valore dell'indice**
2. **Calcolo dell'indirizzo** rispetto alla prima cella (indice 0)

2.4 11. Dichiarazione e Inizializzazione

2.4.1 Array Statici

[] int a[100]; // Spazio riservato a compile-time

- Lo spazio in memoria è **predeterminato** a tempo di compilazione
- La dimensione **non può essere variata** dopo la dichiarazione

2.4.2 Inizializzazione

[] int a[5] = {5, 2, -5, 10, 234}; // Inizializzazione completa int b[4] = {5, 2, -5}; // Quarto elemento = 0 int b[4] = {0}; // Tutti inizializzati a zero

2.4.3 Inizializzazione Programmatica

[] int voti[5]; int i = 0; while (i < 5) { scanf("%d", &voti[i]); i++; }

2.5 12. Array Multidimensionali

2.5.1 Dichiarazione

[] int a[10][5]; // Matrice 10 righe × 5 colonne int b[10][5][20]; // Array tridimensionale

2.5.2 Caratteristiche

- **Accesso**: `a[i][j]` per elemento in riga i, colonna j
- **Memoria**: memorizzazione **lineare** (una riga dopo l'altra)
- **Numero totale di elementi**: N×M per array bidimensionale

2.5.3 Inizializzazione Multidimensionale

[] int a[4][5] = { {2, 5, -8, 7, 6}, {3, 10, 7, 6, 1}, {-1, 8, -8, 5, 3}, {2, 5, 8, 4, 2} };

Regole di Inizializzazione

- `int D[][]={1,2,3,4};` // ERRORE
- `int E[2][]={1,2,3,4};` // ERRORE (manca numero colonne)
- `int F[][2]={1,2,3,4};` // OK (righe calcolate automaticamente)

2.6 13. Array Dinamici

2.6.1 Variable Length Arrays (C99)

[] int n; scanf("%d", &n); int array[n]; // Dimensione calcolata a runtime

2.6.2 Caratteristiche

- **Flessibilità**: dimensione determinata durante l'esecuzione
- **Memoria**: spazio allocato dinamicamente
- **Limitazione**: n deve essere inizializzato **prima** della dichiarazione

---

2.7 14. Limitazioni e Considerazioni

2.7.1 Operazioni Non Permesse

[] array = 5; // Non si può assegnare all'intero array array1 = array2; // Non si può copiare un array in un altro

2.7.2 Controllo degli Indici

- **Responsabilità del programmatore**: C non controlla automaticamente i limiti
- **Accesso fuori range**: `a[100]` per array `a[100]` causa comportamento indefinito

2.7.3 Visualizzazione

[] // SBAGLIATO printf("%d", voti);
// CORRETTO for(int i = 0; i < 5; i++) { printf("%d ", voti[i]); }

2.8 15. Terminologia Tecnica

2.8.1 Classificazione

- **Array in C**: non è un tipo ma un **costruttore di tipo**
- **Forma corretta**: "variabile di tipo array di int" (non "variabile di tipo array")

2.8.2 Tipi di Allocazione

- **Array Statici**: dimensione nota a compile-time
- **Array Dinamici**: dimensione calcolata a runtime (C99)
- **Allocazione automatica**: inizializzazione all'esecuzione del blocco
- **Allocazione statica**: inizializzazione prima dell'avvio del programma

---

2.9 16. Algoritmi Comuni con Array

2.9.1 Ricerca e Confronto

- **Ricerca lineare** negli elementi
- **Confronto di array** elemento per elemento
- **Verifica proprietà** (es. matrice simmetrica: `a[i][j] == a[j][i]`)

2.9.2 Ottimizzazioni

- **Early termination**: interrompere cicli quando condizione è soddisfatta
- **Evitare confronti ridondanti**: nelle matrici simmetriche, controllare solo triangolo superiore

2.9.3 Matrici Speciali

- **Matrici Magiche**: somma costante per righe, colonne e diagonali
- **Formula somma magica**: `n * (n² + 1) / 2`

3 Riassunto: Stringhe in C e Rappresentazione delle Informazioni

3.1 17. Stringhe in C

3.1.1 Concetti fondamentali

- **Definizione**: Una sequenza di caratteri trattati come un oggetto singolo
- **Rappresentazione**: Il C usa array di caratteri (`char`) per rappresentare stringhe
- **Carattere terminatore**: Ogni stringa deve terminare con il carattere nullo `'\0'`

3.1.2 Proprietà delle stringhe in C

- Per memorizzare una stringa di n caratteri servono n+1 posizioni (per il carattere `'\0'`)
- Il carattere terminatore serve alle funzioni di manipolazione per identificare la fine della stringa

3.1.3 Dichiarazione e inizializzazione

Forma semplificata [] char mia_stringa[] = "Ciao a tutti!";

- Il compilatore calcola automaticamente la dimensione (14 caratteri = 13 + 1)
- Aggiunge automaticamente il carattere `'\0'`

Forma esplicita [] char mia_stringa[] = {'C','i','a','o',' ','a',' ','t','u','t','t','i','!','\0'};

- Il programmatore deve inserire manualmente `'\0'`

Con dimensione predefinita [] char frase[20] = "Ciao a tutti!";

- Uno dei 20 elementi viene riservato automaticamente per `'\0'`

3.1.4 Visualizzazione

- Utilizzo di `printf("%s", nome_stringa)` per stampare l'intera stringa
- `%s` è lo specificatore di formato per le stringhe
- La stampa continua fino al carattere `'\0'`

3.1.5 Lettura

- `scanf("%s", nome_stringa)` - **NON** serve l'operatore `&`
- Per singoli caratteri: `printf("%c", stringa[indice])`

3.1.6 Limitazioni

- Non è possibile assegnare direttamente: `stringa = "nuovo valore";`
- Gestione dinamica complessa (richiede allocazione manuale)

3.2 18. Rappresentazione delle Informazioni

3.2.1 Processo di codifica/decodifica

- **Informazioni → Codifica → Dati → Decodifica → Informazioni**
- I calcolatori memorizzano e manipolano informazioni sotto forma di dati organizzati secondo rappresentazioni specifiche

3.2.2 Tipi di dati

- **Tipi semplici**: interi, caratteri, numeri frazionari
- **Tipi complessi**: costruiti a partire da quelli semplici

3.2.3 Rappresentazione dei numeri interi

Sistemi numerici posizionali

- **Sistema decimale** (base 10)
- **Sistema binario** (base 2): fondamentale per i calcolatori

Interi relativi - Complemento a 2 (CA2)

- **Vantaggi**: rappresentazione uniforme di numeri positivi e negativi
- **Processo di inversione**:
  1. Inversione di tutti i bit
  2. Aggiunta di 1 al risultato
- **Esempio**: +6 = 0110 → inversione: 1001 → +1: 1010 = -6

3.2.4 Rappresentazione dei numeri frazionari

Virgola fissa

- Posizione decimale fissata in anticipo

Virgola mobile (Standard IEEE-754)

- **Componenti**: segno, esponente, mantissa
- **Precisione singola** (32 bit): 1 bit segno + 8 bit esponente + 23 bit mantissa
- **Mantissa normalizzata**: compresa tra [1,2)
- **Bias dell'esponente**: 2^(N-1) per N bit

3.2.5 Rappresentazione dei caratteri

Standard ASCII

- **American Standard Code for Information Interchange**
- Definito dal 1963
- **Esempi**:
  - 'A' = 65
  - 'a' = 97
  - Differenza tra maiuscole e minuscole: 'a' - 'A' = 32

Manipolazione caratteri

- Conversione maiuscolo/minuscolo basata sulla differenza ASCII
- Confronti diretti possibili: `if(C >= 'a' && C <= 'z')`

3.3 19. Gestione e manipolazione stringhe

3.3.1 Funzioni di libreria (cenni)

- Calcolo lunghezza
- Concatenazione
- Ricerca di caratteri o sottostringhe
- Conversioni (es. stringa → float)

3.3.2 Limitazioni del C

- Gestione dinamica complessa
- Necessità di calcolare e allocare manualmente la memoria
- **Soluzione C++**: classe String per gestione ad alto livello

3.4 Concetti chiave da ricordare

1. Le stringhe in C sono array di char terminati da `'\0'`
2. La rappresentazione binaria è fondamentale nei calcolatori
3. Il complemento a 2 è lo standard per i numeri relativi
4. IEEE-754 standardizza la rappresentazione floating point
5. ASCII fornisce la codifica standard per i caratteri
6. La codifica/decodifica è essenziale per l'interpretazione dei dati