Contents

Caratteristiche

- Procedurale
- Imperativo
- Strutturato
- Supporto alla ricorsione
- Tipicizzato
- Vicino all'hardware
- Compilato

Area di memoria C

Il modello di memoria in C è lo stack. composto da parole da 32 bit.

C'è una parte per il programma e una per i dati, quando lo stack finisce abbiamo uno stack overflow

Frame pointer(FP): Contiene l'indirizzo base dove parte il programma(funzione main()) **Stack pointer(SP)**: Contiene l'indirizzo alla prossima variabile

All'inizio FP e SP coincidono

La memoria in C viene allocata a partire dagli indirizzi alti.

Le istruzioni nell'area codice vengono mandate al registro IR della CPU che poi eseguirà l'istruzione.

Quando viene allocata una variabile lo stack si riduce per far spazio alla variabile(si può rischiare che questo stack finisca e succeda uno stack overflow)

lvalue = rvalue

Funzioni math.h

- pow(x,y) potenza
- sqrt(x) radice quadrata
- ceil(x): Restituisce il più piccolo intero maggiore o uguale a x.
- floor(x): Restituisce il più grande intero minore o uguale a x.
- round(x): Restituisce il valore di x arrotondato al numero intero più vicino.

Booleani

I booleani sono contenuti in tipo int e vengono "sprecati" 31 bit per mantenerà l'allinamento delle parole a 32 bit, in quanto basterebbe 1 bit per rappresentare un bool.

Il libro usa stdbool.h per usare il tipo bool in C

Variabili strettamente booleane

Variabili che possono essere solo booleane per natura

Espressioni logiche

- && AND
- || OR
- ^ XOR
- ! NOT

Il compilatore valuta le espressioni in maniera sequenziale

bloccante && non-bloccante

è meglio prima mettere la non-bloccante perchè nel caso la non-bloccante sia falsa, nel caso di &&, l'espressione è sicuramente falsa e si risparmia tempo.

Il C non ha un operatore XOR e va quindi scritto come:

A && !B || !A && A

Priorità degli operatori

L'AND ha priorità sull'OR

Ordine operatori e valutazione lazy

La valutazione avviene solo se necessario(lazy).

Ad esempio con $3 < 5 \mid\mid 10 == 7 + 3$

- Valuta $3 < 5$ ed è true

Non va avanti con la valutazione

Ad esempio con

```
bool b9 = false && (10 / 0) == 1;
```

Non viene dato errore della divisione con 0 mentre:

```
bool b9 =  (10 / 0) == 1 && false;
```

Dà errore.

If-Else

- Se le condizioni sono onerose(I/O, ecc...) o dipendono dall'istante dove vengono eseguite, al posto di metterle direttamente nell'if è meglio metterle in una variabile esterna

Attenzione alle operazioni matematiche

Inoltre per $3 < 5 < 7$ succede che

- Viene valutato $3 < 5 = 1$
- Viene valutato $1 < 7$

Quindi la valutazione matematica viene cambiata

Goto

Salto incondizionato. NON VA USATO in quanto non sarebbe programmazione strutturata come return e break per interrompere un ciclo while o for.

Scanf

Scanf ritorna il numero di argomenti letti con successo.

Note

- Usare sempre graffe in if-else anche se c'è una sola riga
- Non terminare mai con un else if

Floating point

Secondo lo standard IEEE 754

Divisione in C

In C per evitare errori di troncamento, le divisioni con la virgola vanno fatte con tutti tipi float.

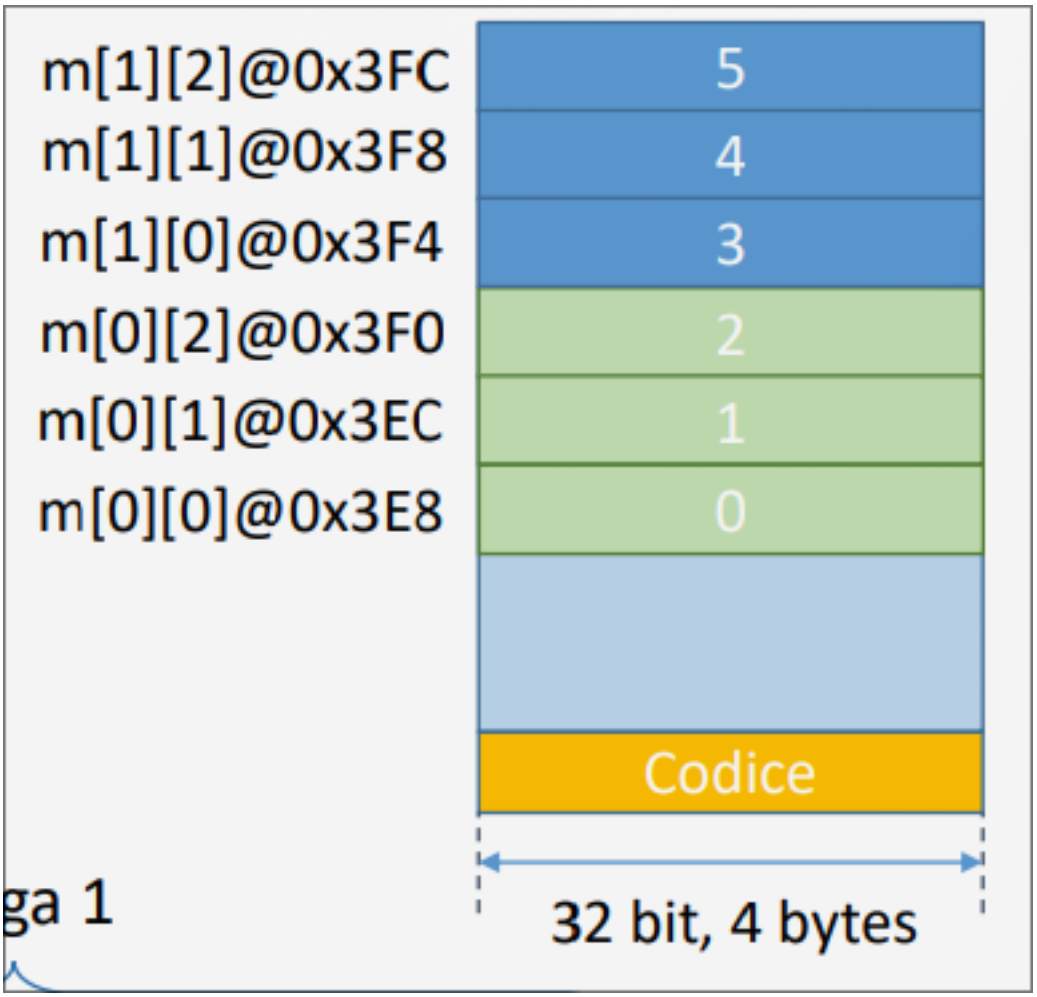
Ad esempio:

```
int x = 5;
int y = 2;
float res = x / y;  // Avrà valore 2.00
```

Invece è corretto:

```
float x = 5;
float y = 2;
float res = x / y;  // Avrà valore 2.50
```

Matrici



Per ottimizzare gli accessi alle matrice, dobbiamo accedere agli elementi per righe e poi per colonne.(ordine row-major).

Notazione for:

- i ciclo esterno
- j ciclo interno

Matrici rugged

Matrici che sono quadrate ma su alcune celle ci sono valori privi di valore.

Le colonne con valori validi sono contenute nel vettore rags.

Funzioni

Programmazione modulare, scriviamo porzioni di codice in parti separate e le richiamiamo nel codice. Queste sono le funzioni.

Bisogna mettere solo un return nella funzione

Bisogna usare l'approccio bottom-up, prima scrivere il prototipo e poi la funzione:

```
#include <stdio.h>
int square(int a);

void main(void) {
    int x=2, asq=0;
    asq = square(x);
    printf («%d al quadrato: %d\n», x, asq);
}

int square(int x) {
    int x2 = x * x;
    return x2;
}
```

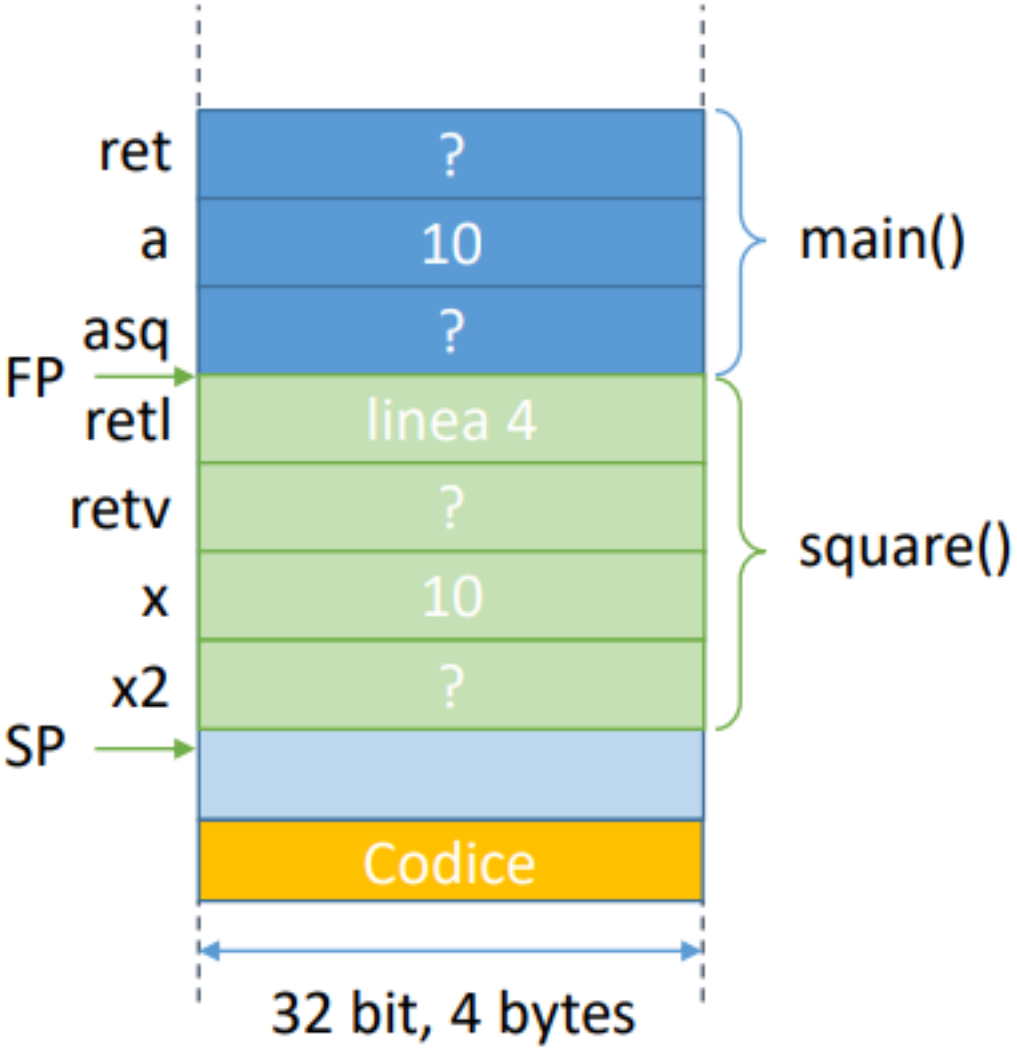
Funzioni per valore e occupazione memoria.

Ogni chiamata a funzione, comporta l'allcazione di un nuovo frame nello stack.

L'indirizzo alla prima istruzione del frame è il frame pointer della CPU.

L'indirizzo dell'ultima istruzione del frame è nello stack pointer della CPU.

Ogni funzione ha il proprio stack che occupa memoria con indirizzi decrescenti.



Dove:

- retl è la riga da eseguire dopo la fine della funzione
- retv è il valore restituito dalla funzione

I parametri e le variabili locali sono allocate nello stack.

Lo stack viene inizializzato quando le funzioni vengono chiamate.

Funzioni con passaggio per riferimento

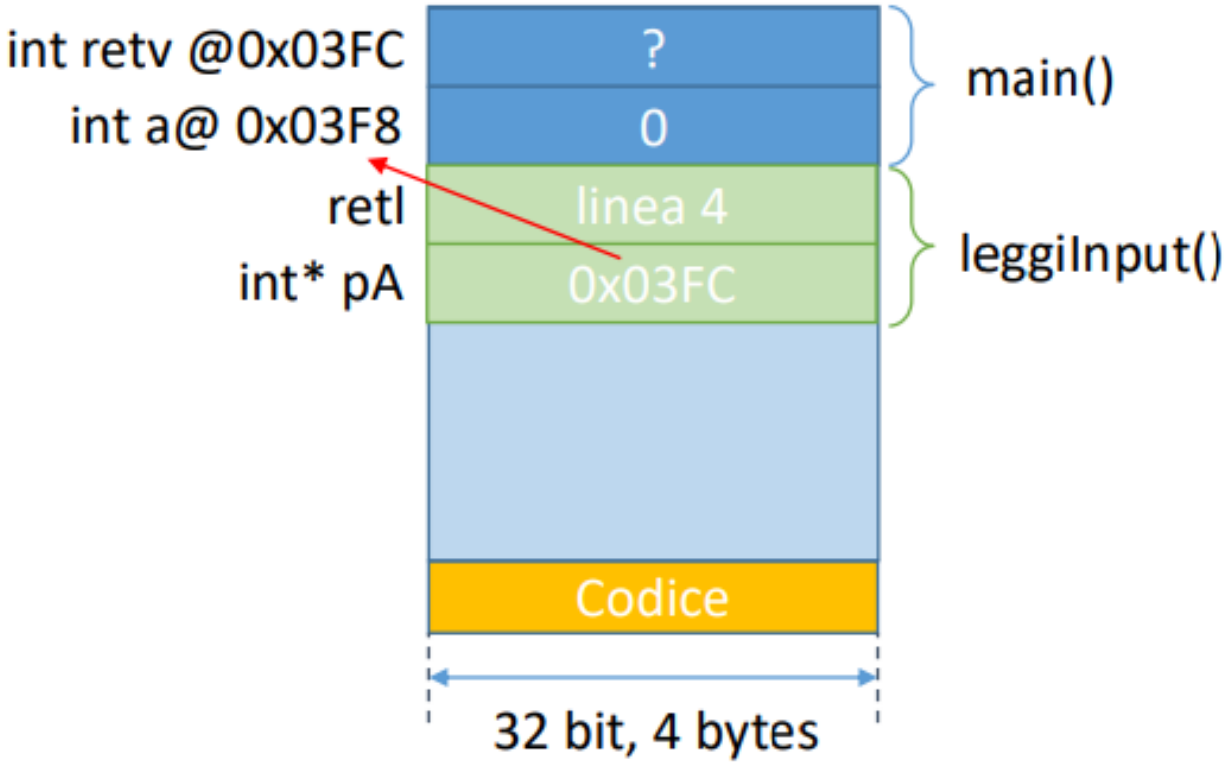
La funzione si aspetta come tipo un puntatore di tipo, cioè l'indirizzo della variabile.

Per prendere l'indirizzo usiamo l'operatore di referenziamento &

```
int main(void) {
    int a = 0;
    leggiInput (&a);
    printf("a vale %d\n", a );
```

```
}
void leggiInput (int* pA) {
    printf("Inserisci un intero: ");
    scanf("%d ", pA);
}
```

Nello stack al posto dei valori verrà salvato l'indirizzo delle variabili.



NB: L'indirizzo dovrebbe essere 0x03F8

In questo modo la funzione opererà direttamente sulle variabili dereferenziate.

Per convertire l'indirizzo nel suo valore si usa l'operatore di dereferenziamento *

Passaggio di array a funzioni

La lunghezza va passata come parametro e per i VLA bisogna passare prima la lunghezza poi l'array per referenza.

```
void func(size_t len, int a[]);
```

a[] punta alla prima cella dell'array.

L'operatore [] dereferenzia l'indirizzo all'elemento di a.

Bubble sort

Per ordinare dal minore al maggiore

```
void sort_array(size_t len, int v[]) {
    bool eseguiCiclo = true;
    while (eseguiCiclo == true) {
        eseguiCiclo = false;
        for (size_t i = 0;i < len-1; i++) {
            if (v[i] > v[i+1]) {
                int tmp = v[i];
                v[i] = v[i+1];
                v[i+1] = tmp;
                eseguiCiclo = true;
            }
        }
    }
}
```

Codifica del testo

ASCII Contiene solo un set di lettere ma non tutte(ad esempio quelle accentate o i simboli matematici). Inoltre non bastavano 28 valori per tutte le lingue del mondo(soprattutto se hanno ideogrammi)

Codepages Unisce l'ASCII e ci aggiunge delle flag per rappresentare più simboli

Unicode 1.0 Codifica a 16 bit per carattere.

Non basta comunque per tutti gli ideogrammi

Unicode 2.0 Codifica a 21 bit che è poco efficiente in termini di memoria.

Quindi la codifica viene divisa in due:

Codifiche wide-character

- UCS-16, 16 bit per carattere
- UCS-32, 32 bit per carattere

Codifiche multi-byte

- UTF-8, codifica a 8 bit.

Tipi di carattere in C

- single-byte: char, va bene per ASCII
- wide: wchar_t
- multi-byte: char che implementa un'elaborazione più avanzata per riconoscere i caratteri

Quantificazione universale su array

```
bool tutti = true;
for (size_t i=0; i<len && tutti; i++) {
 if (! Cond valutata su a[i] )
 tutti = false;
}
```

Quantificazione esistenziale su array

```
bool esiste = false;
for (size_t i=0; i<len && !esiste; i++) {
 if (Cond valutata su a[i])
 esiste = true;
}
```

Funzioni su caratteri(ctype.h)

- isvowel(): Controllo se è vocale
- isalpha(): Controllo se è lettera
- isalnum(int c): Controllo se è una lettera o una cifra
- isalpha(int c): Controllo se è una lettera
- iscntrl(int c): Controllo se è un carattere di controllo
- isdigit(int c): Controllo se è una cifra
- isgraph(int c): Controllo se è stampabile eccetto lo spazio
- islower(int c): Controllo se è una lettera minuscola
- isprint(int c): Controllo se è uno stampabile (compreso lo spazio)
- ispunct(int c): Controllo se è un carattere di punteggiatura
- isspace(int c): Controllo se è uno spazio, tabulazione, ritorno a capo, avanza pagina, avanzamento di riga o avanzamento di tabulazione
- isupper(int c): Restituisce un valore diverso da zero se il carattere è una lettera maiuscola
- isxdigit(int c): Restituisce un valore diverso da zero se il carattere è una cifra esadecimale
- tolower(int c): Converte il carattere in minuscolo, se possibile.
- toupper(int c): Converte il carattere in maiuscolo, se possibile.