Pseudocodifica

La pseudocodifica è un linguaggio semplice che non viene né compilato né interpretato, ma che può essere implementato con un qualsiasi linguaggio di programmazione.
È un linguaggio universale, che viene letto dal programmatore (e non dalla macchina), in cui esiste un numero fisso di tipi di variabili, ma non sono ammessi casting di tipo e esistono i puntatori, ma la loro algebra non è supportata.

In questo linguaggio è **molto importante l'indentazione** del codice, in quanto non fa uso delle parentesi.

Variabili e assegnazioni

Le variabili hanno un tipo ma non necessitano di una dichiarazione.
Le assegnazioni sono eseguite normalmente `<left-value>=<right-value>`.

Tipo di variabili

Lo pseudocodice **non prevede definizioni di tipo**, perciò si suppone che le variabili vengano allocate quando vengono chiamate per la prima volta. Se necessario si possono usare i commenti per esplicitare il tipo quando non evidente.

Array

Gli array possono essere di qualunque tipo e i loro elementi sono identificati da `A[i] con i=0⋯A.lenght`. Nella variabile è contenuta anche l'informazione sulla lunghezza dell'array (all'inizio).

Oggetti

Gli oggetti sono dati più complessi, formati da attributi, detti campi, ai quali si accede con l'operatore `·`. La variabile che rappresenta l'oggetto è un puntatore ad esso, quindi un riferimento.

Espressioni booleane

Le costanti booleane sono `true`, `false` e gli operatori booleani sono `and`, `or`, `not`.

È valida la ***legge di De Morgan***, per cui

`not( ⋯ and ⋯) → (not⋯)or(not⋯)`
`not( ⋯ or ⋯) → (not⋯)and(not⋯)`

Istruzioni Condizionali

Le istruzioni condizionali sono rappresentate dal costrutto

```
IF condizione THEN
        operazioni
ELSE
        operazioni
```

Ripetitive

Le istruzioni ripetitive sono rappresentate dai costrutti

```
FOR inizializzazione TO valore max
        operazioni

WHILE condizione DO
        operazioni
```

```
FOR inizializzazione DOWN TO valore min
        operazioni

REPEAT
        operazioni
UNTILL condizione
```

Procedure

Le procedure, cioè le funzioni, possono o meno ricevere parametri e ritornare valori, espressi o meno in una sola variabile. **Il passaggio dei parametri avviene solo per valore**, quindi la funzione riceve il valore dei parametri formali tramite i parametri attuali. La modifica dei parametri formali non influisce sui parametri attuali, tranne quando il parametro è un oggetto o un array, perché essi sono rappresentati da un riferimento (cioè si accede direttamente alla cella di memoria). Si dice che **non c'è side effect**.

Linguaggio c

Esecuzione di un programma

Trasformazione dei file

Ogni programma è composto da uno o più file di codice, ma di questi solo uno può contenere la funzione principale `main`. L'insieme di questi file subisce delle trasformazioni prima di essere eseguito. Tali trasformazioni si dividono in due fasi:

- **fase di compilazione**, in cui tutti i file `.c` vengono trasformati in file oggetto `.o`
- **fase di linkaggio**, in cui i file oggetto vengono trasformati in file eseguibili.

Errori

Esistono perciò due tipologie di errori che si possono presentare:

- **errori di compilazione**, che si verificano in presenza di errori di sintassi
- **errori di linkaggio**, che si verificano in presenza di errori di semantica, quando ad esempio manca la funzione principale oppure ne è stata creata più di una, oppure quando le funzioni non sono state dichiarate o lo sono state più volte.

Dichiarazioni e definizioni

Variabili e funzioni

La **dichiarazione** annuncia l'esistenza e la possibilità di utilizzo della **variabile o funzione**; può essere iterata, ma *non produce codice nel file oggetto*.

La **definizione**, che include la dichiarazione, descrive in dettaglio la **variabile o funzione**; non può essere iterata, ma *introduce codice nel file oggetto*.

Tipi

La **dichiarazione** annuncia l'esistenza e la possibilità di utilizzo di un **tipo**; può essere iterata, ma *non produce codice nel file oggetto*.

La **definizione**, che include la dichiarazione, descrive in dettaglio il **tipo**; può essere iterata, ma *non introduce codice nel file oggetto*.

Costrutto typedef

Il costrutto typedef **permette di ridefinire un tipo o una struttura** assegnandogli un nuovo nome (non esistente).

File Header

Generalmente, **tutte le dichiarazioni vengono messe in un file header** .h, che poi può essere importato (come libreria) in altri file .c.
Tutto ciò che è dichiarato in questo file deve essere **definito in un corrispondente file .c**, così che venga prodotto l'adeguato codice nel file oggetto e la fase di linking vada a buon fine.

Standard ANSI C

Per il linguaggio C è stato definito uno **standard ANSI C**, chiamato anche C89 o C90, che non viene però usato dai compilatori e quindi per far si che questi vi si attengano, è necessario **compilare con l'opzione** `-pendantic-errors`.

Puntatori

I puntatori, in altri linguaggi chiamati *riferimenti*, **sono variabili che contengono l'indirizzo di un'altra cella di memoria**.

Dichiarazione

I puntatori si dichiarano, come le variabili, **indicando nome e tipo.**
Per inizializzare un puntatore che non contiene un indirizzo significativo, è possibile usare la **costante NULL**, in quanto corrisponde al valore zero, che non è legittimo per un indirizzo di memoria accessibile dall'utente.
Tale inizializzazione può essere omessa in quanto, nella dichiarazione senza inizializzazione, risulta implicito che il puntatore non abbia valore significativo.

Operatori

I puntatori vengono utilizzati attraverso due **operatori unari: star *** (contenuto di) e **and** & (indirizzo di).

Possono essere combinati tra loro per assegnazioni ed utilizzati nelle funzioni per passaggi e restituzioni di valori, attivando il side effect.

Operatore star

L'operatore star si compone con i tipi per indicare che la variabile puntatore punta ad una variabile di quel tipo.
Questo operatore può essere *interpretato in due modi*:

- la variabile `var` è di tipo `tipo*`
- `*var` è l'oggetto puntato da `var` ed è di tipo `tipo`.

Si può, inoltre, usare anteposto a un indirizzo di memoria (espresse da variabili o operazioni tra esse), in modo da identificarne il contenuto.

Operatore and

L'operatore and, opposto a star, **si applica ad una variabile per estrarne l'indirizzo di memoria**.

Non ha senso usarlo con le espressioni generiche, perché significherebbe anteporlo ad un valore e non ad una variabile.

Array

Definizione

Un array è una sequenza omogenea di variabili, memorizzate in celle contigue di memoria e indicizzate da un indice intero.
L'array è quindi **un puntatore**, in quanto **rappresenta l'indirizzo della prima cella di memoria** della sequenza, ma è *costante*, cioè non ne è permessa la sovrascrizione o modifica dell'indirizzo.

Dichiarazione e accesso

La dichiarazione di un array, in C, **è possibile solo indicando il numero di elementi costante** e non una variabile.

L'accesso agli elementi di un array è consentito attraverso l'uso delle parentesi quadre con l'indice dell'elemento a cui si vuole accedere (variabile o costante). Questa operazione corrisponde all'utilizzo dell'operatore star dei puntatori davanti un indirizzo di memoria. È per questo che il primo elemento dell'array si trova nella posizione 0 e non 1.

Allocazione dinamica

Un array può essere reso dinamico utilizzando le apposite funzioni malloc **e** calloc (azzera i valori delle celle) per **l'allocazione della memoria nell'heap**, sezione della memoria gestibile dal programmatore (gli array statici vengono allocati nello stack).

Una volta allocato l'array, si può accedere agli elementi con i classici costrutti ed è inoltre possibile **riallocarne la memoria con la funzione** realloc.

Quando si alloca dinamicamente la memoria, se non si libera la memoria, con la funzione free, si ha una costante perdita di memoria, **memory leak**, che comporta un rallentamento della macchina.

Strutture

Definizione

La dichiarazione di una struttura, in C, solitamente si effettua direttamente con la **definizione**, anche se a volte può essere necessaria una dichiarazione

in precedenza. La struttura non ha tipo in quanto è lei stessa a definire un tipo, quindi è sufficiente anteporre al nome il costrutto `struct` ed eventualmente anche il costrutto `typedef`, per ridefinirne il nome.
È possibile rinunciare al nome della struttura, ma ciò ne impedirà l'utilizzo nel resto del codice.

Ogni struttura è costituita da **campi** (attributi), che possono essere di tipo qualsiasi e ai quali si accede attraverso l'**operatore freccia** `var→attr`, che ha la stessa funzione dell'operatore star `(*var).attr`.

La dichiarazione delle variabili del tipo definito dalla struttura è possibile sia in seguito alla definizione di questa, che contestualmente.

Definizione ricorsiva

I suoi campi possono essere anche del tipo definito dalla struttura stessa, che quindi risulta contenere **riferimenti a sé stessa**.

Liste

Definizione

Una lista, in C, viene costruita attraverso **due strutture**, una rappresentante la **lista** e una i suoi **elementi**. La dichiarazione è quindi, inclusa nella definizione.

Definizione semplificata

È possibile però, fare una **definizione semplificata**, con cui si fa coincidere la lista con il suo primo elemento. Questa definizione *non è utilizzabile in pseudocodifica*, in quanto chiamando le funzioni che svolgono le operazioni, verrebbe passato un parametro nullo e le modifiche avverrebbero solo a livello locale, lasciando invariata la lista. In C, si può ovviare a questo problema usando *l'operatore and nel passaggio della lista come parametro*, così da permettere alla funzione di accedere direttamente all'indirizzo di memoria.

Sommario