SINTASSI P-LIKE

Università degli studi di Napoli

Federico II

Indice

1 Sintassi

1.1 Definizioni e inizializzazione

- `Begin` → per iniziare il programma;
- `Read` → istruzione di input (read R, significa che sono io che inserisco R) oggetto variabile;
- `Print` → istruzione di output (print C, significa che il programma mi dovrà stampare il valore C);
- `End` → per terminare il programma ;
- `:=` → assegnazione (C:=a+b , vedi quanto è a e vedi quanto è b, sommali e dai quel valore a C);
- `Var:R:real` → Variabile reale;
- `Var:Nome:Character` → Variabile alfanumerica;
- `Var:L:logical` → Variabile logico;
- `Var:L: integer` → Variabile intero;
- `var:nome array (dimensione):array of real` → Definizione di un array

1.2 Costanti

Esistono vari tipi di costante:

1. Alfanumeriche cioè `'costante'`, la costante racchiusa in due apici;
2. Logiche cioè quelle che vengono indicate con `.TRUE.` oppure `.FALSE.`;
3. Intere cioè cifre decimale precedute dal segno;
4. Reali cioè cifre decimali contenenti la virgola e precedute dal segno.

1.3 Cast

- Quando voglio fare dei cambi solo su quella riga del codice e non in tutto il codice;
- Quando do il valore ad una costante (base=5.) metto un punto davanti al valore.

1.4 Struttura di selezione

```
if (condizione) then
   istruzione 1
else
   istruzione 2
endif
```

1.5 Cicli

```
// DA UTILIZZARE SOLO QUANDO SI SA QUANTE VOLTE IL BLOCCO DI ISTRUZIONI DEVE ESSERE RIPETUTO
for (condizione) do
    istruzione
endfor

// Alternativa al for può anche essere il do-while
// Quando so quante volte deve essere ripetuta 'listruzione
while (condizione) do
    istruzione
endwhile
```

Se si vuole usare una sorta di Do-while, bisogna utilizzare il `Repeat-until`:

```
repeat
    istruzione
until
```

1.5.1 Osservazioni

Per il ciclo **`for`** bisogna sapere a priori il numero di interazioni (non necessario per ciclo **`while`** e **`repeat`**).

Nel **`while`** se l'istruzione è falsa allora il ciclo si blocca mentre nel **`repeat`** le istruzioni vengono ripetute almeno una volta

1.6 PROCEDURE

Stabilire le regole per la comunicazione tra i due algoritmi

1. Scrivo algoritmo generale;
2. Do un nome a tale algoritmo;
3. Riferirsi a tale algoritmo.

Quando dobbiamo usare una function o una procedure: si scrive prima **`function`** o **`procedure`**, poi si dichiarano le variabili e poi il begin con le varie strutture iterative.

1.7 Leading Dimension

Problema che nasce con la memorizzazione degli elementi di un array 2D per righe in quanto posso avere un disallineamento tra gli elementi dell'array chiamante e array procedure:

A(1,1)	TAB(1,1)
A(1,2)	TAB(1,2)
A(1,3)	TAB(2,1)
A(2,1)	TAB(2,2)
A(2,2)	TAB(3,1)
A(2,3)	TAB(3,2)
A(3,1)	

Quindi con LDA vado a fornire il numero di colonne cosi da poter far allineare l'array procedure (aggiungo all'array chiamante LDA:= numero di colonne).

A(1,1)	TAB(1,1)
A(1,2)	TAB(1,2)
A(1,3)	
A(2,1)	TAB(2,1)
A(2,2)	TAB(2,2)
A(2,3)	

Lascio una riga vuota in modo tale da potermi trovare con l'allineamento

1.8 Algoritmo di Hormer

Da utilizzare nella valutazione di un polinomio

$$nM + nA$$

o nella forma generale

```
p:=an
For i:=n-1 to 0, step-1 do
   p:=p*x+ai
Endfor
```

1.9 Matrice sparsa

Matrice che contiene molti 0 (quasi il 95%)

1.10 Merge sort

Lo utilizzo quando ho due array. Vado ad ordinare gli elementi dei due array, inserendoli ordinati in un terzo array.

1.11 Array bidimensionale

Matrice con tutti gli elementi uguali (ovvero tutti dello stesso tipo)

1.12 Costo Computazionale

- Complessità di tempo `T(n)`
- Complessità di spazio `S(n)`

1.13 Binary search (Ricerca binaria)

Parto da un array che ho già ordinato.

Dopodichè vado a cercare un elemento all'interno di questo array confrontandolo con il valore di mezzo.

A seconda di come questo elemento sia < o > dell'elemento di mezzo, considero solo una parte di array (quella di destra o di sinistra) e continuo con questo procedimento fin quando non trovo il mio elemento.

1.14 Epsilon macchina

Il numero macchina più piccolo che se anche va sommato non porta conseguenze

1.15 Trasposta di una matrice:

Quando trasformo le righe di una matrice in colonne e viceversa. Con la matrice quadrata è più semplice rispetto ad una matrice rettangolare.

Mi trovo in uno spazio 2D.

1.16 Prodotto scalare tra due matrici

Posso usare sia una function che una procedure

1.17 Saxpy (somma di un multiplo di un vettore con un altro vettore)

$$Z = \alpha x + y$$

Con `x` e `y` vettori di `n` componenti e `alpha` scalare. Il calcolo avviene in place ovvero il risultato va nello stesso vettore y (y=alphax+y).

```
Procedure saxpy (in:alpha, x, n; in/out:y)
  Var:i,n:integer
  Var:alpha:real
  Var:x,y:array …[1n] of real
  Begin
    For i:=1 to n do
        Y(i):=y(i)+alphax(i)
    Endfor
  End
End saxpy
```

1.18 Gaxpy (general saxpy)

Si utilizza quando l'alpha della saxpy è una matrice `m x n`

1.19 Back substitution

Uso questo algoritmo quando voglio risolvere una matrice triangolare superiore.

$$\text{A(n)} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & a_{2,n} \\ 0 & 0 & a_{3,3} & a_{3,n} \\ 0 & 0 & 0 & a_{m,n} \end{pmatrix}$$

1.20 Diagonal substitution

Uso questo algoritmo quando voglio risolvere una matrice diagonale

$$\text{A(n)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & -3 \end{bmatrix}$$

1.21 Upper substitution

Uso questo algoritmo quando voglio risolvere una matrice triangolare inferiore.

$$\text{A(n)} = \begin{pmatrix} a_{1,1} & 0 & 0 & 0 \\ a_{2,1} & a_{2,2} & 0 & 0 \\ a_{3,1} & a_{3,2} & a_{3,3} & 0 \\ a_{m,1} & a_{m,2} & a_{m,3} & a_{m,n} \end{pmatrix}$$

1.22 Insertion sort

Tecnica del giocatore che si sistema le carte in ordine non decrescente.

1.23 Algoritmo di Shellsort

Diminuzione di incrementi

1.24 Record

Generalizzazione di un array.

In un dato record gli elementi possono anche essere di tipo diverso (cosa non possibile per un array).

1.24.1 Dichiarazione

```
var: nomevariabile: record
```

Dopo aver dichiarato il record, pass a dichiarare tutti i selettori del record; ad esempio:

```
var:data:record

Giorno:integer

Mese:character
```

Invece di scrivere tutti i selettori possono anche creare un nuovo tipo di dato:

```
type data_m_g_a:record

Var:data_nascita,data_laurea:data_g_m_a
```

1.25 Dato complex

Utilizzato per esprimere numeri complessi.

```
1 Var:x,y,z:record
2
3 Re:real
4
5 Im:real
6
7 End
```

Oppure:

```
var:x,y,z:complex

Procedure somma_complex (in:x,y;out:z)……

  z.Re:=x.Re+y.Re
  z.Im:=x.Im+y.Im…
  .
End
```

1.26 Stack

Struttura lineare aperta che permette di inserire ed estrarre solo a partire dall'inizio della struttura.

2 Esercizi e Soluzioni

2.1 Calcolare massimo di un array:

```
begin massimo_array
  var:a[100]:array of real
  var:i,n:integer
  read n
  for i=2, n do
    if (max a < a(i)) then
        max = a(i)
    endif
  endfor
end
```

2.2 Sequential search

Ricerca sequenziale all'interno di un array già definito (al quale già sono stati assegnati dei valori):

```
logical function sequential_search (in: n,A,x)
  var:i,n,x:integer
  var:A [1..n]:array of integer
  begin
    i:=1
    while (A(i) !=x .AND. i<=n) do
        i:=i+1
    endwhile
    if (i>n) then sequential_search
    else
        sequential_search:= .TRUE.
```

```
    endif
  end
end sequential_search
```

In alternativa:

```
var:i,n,x:integer
Var:A…[1100]:array of integer
Var:sequential_search(n,A,x):logical function
Begin
  Read n,x
  For i:=1 to n do
    Read A(i)
  Endfor
  If (sequential_search(n,A,x))=.TRUE. then
    Print ‘(il ’numero, x ’, è presente ’’nellarray)
  Else
    Print ‘(il ’numero, x, ‘non è presente ’’nellarray)
End
```

2.3 Selection sort

Ordinamento di un array completo dunque posiziona i numeri in ordine crescente o decrescente

```
Min:=A(1)
P:=1
For j:=2 to n do
    If (A(j)<min) then
        Min:=A(j)
        P:=j
    Endif
endfor
```

`Scambio tra A(1) e A(p)` ripetuta più volte

```
procedure selection_sort (in:n; in/out:A)
  Var:i,j,n,p:integer
  Var:A…[1n]:array of real
  Var:min:real
  Begin
    For i:=1 to n-1 do
      Min:=A(i)
      P:=i
```

```
      For j:=i+1 to n do
        If (A(j)<min) then
          Min:=A(j)
          P:=j
        Endif
      Endfor"
      Scambio tra A(i) e A(p)"
    Endfor
  End
End selection_sort
```

2.4 Exchange sort (o Bubble sort)

Ordinamento per scambio → gli elementi vengono confrontati a due a due.

Confrontando a due a due e mettendo i più piccoli come primi in automatico quelli più grandi slitteranno verso la fine (proprio come delle bolle).

L'ultimo, infatti, non viene considerato in quanto si trova già nella sua posizione

```
For j:=1 to n - 1 do
  If (A(j) > A(j + 1)) then
    "scambio"
Endfor
```