Materiale didattico preparato dal
dott. Stefano Ferilli

Corso di Programmazione
Linguaggio Pascal

Dott. Pasquale Lops
lops@di.uniba.it

Pascal
Standard ISO 7185

- Standard
  - Nucleo di caratteristiche garantite da qualunque implementazione
- ISO
  - International Standard Organization
- ***IMPORTANTE!***
  - Usare soltanto le caratteristiche del linguaggio definite dallo standard
  - Non fare affidamento su caratteristiche definite da specifiche implementazioni

Regole di Scrittura

- Non si distingue fra maiuscole e minuscole
  - Tranne che nelle stringhe di caratteri
- Gli spazi superflui sono ignorati
  - Non possono spezzare parole chiave o simboli composti
- Convenzione
  - Parole chiave in maiuscolo
  - Identificatori in minuscolo
  - Indentazione che evidenzi le sequenze di istruzioni

Parole Riservate

- Parole Chiave

div	mod	nil	in	or	and	not
if	then	else	case	of	repeat	until
while	do	for	to	~~goto~~	downto	begin
end	with	const	var	type	array	record
set	file	function	procedure	~~label~~	packed	program

- Simboli speciali

+ – * / = <> < <= > >= ( ) [ ] := . .. : ; ↑ ‘ { }

Parole Riservate

- Costanti

  false true maxint

- Tipi

  integer real boolean char text string

- Variabili

  input output

- Direttive

  forward

Parole Riservate

- Funzioni

abs	arctan	chr	cos	eof	eoln
exp	ln	odd	ord	pred	round
sin	sqr	sqrt	succ	trunc	

- Procedure

get	new	pack	page	put	read
readln	reset	rewrite	unpack	write	writeln

Parole Riservate

- Rappresentazioni alternative:
  - (. per [
  - .) per]
  - @ o ^ per ↑
  - (* per {
  - *) per }

Commenti

- Qualunque sequenza di caratteri inclusa fra
  - { e }
  - (* e *)
- Ignorati dal compilatore
  - Equivalenti ad un carattere di spaziatura
  - Utili alla comprensione del programma
    - Scopo del programma o di un sottoprogramma
    - Scopo di un frammento di codice
    - Uso di un identificatore

Programma Pascal

- Assimilabile ad una procedura che trasforma un input in un output
  - Intestazione simile a quella di una procedura
    - Nome
    - Argomenti di ingresso/uscita

Programma Pascal

- Parte dichiarativa
  - Intestazione di programma
  - Definizione delle costanti
  - Definizione dei tipi
  - Dichiarazione delle variabili
  - Dichiarazione di sottoprogrammi
- Corpo
  - Insieme delle istruzioni da eseguire

Programma Pascal
Schema

program *nome* (**input**, **output** {, … })**;**

{ sezione dichiarativa }

begin

{ corpo }

end.

- Sezione dichiarativa
  - Risorse usate dal programma
- Corpo
  - Sequenza di istruzioni del programma

Programma Pascal
Il più piccolo concepibile

PROGRAM nullo**;**

BEGIN

END.

- Non fa nulla

Sezione Dichiarativa

- Costanti
  - Introdotte dalla clausola **const**
- Tipi
  - Introdotti dalla clausola **type**
- Variabili
  - Introdotte dalla clausola **var**

Sezione Dichiarativa

- Sottoprogrammi
  - Procedure
    - Introdotte dalla clausola **procedure**
  - Funzioni
    - Introdotte dalla clausola **function**

Istruzioni di uscita

write(*lista_di_output*)

- Emette il valore di ogni variabile o costante della lista

writeln(*lista_di_output*)

- Come sopra
  - La lista di output può essere vuota
- Il cursore avanza automaticamente alla riga successiva

- Esempi
  - write(‘Salve!’)
  - writeln(‘risultato = ’, max)

Istruzione di Uscita
Esempio

```
PROGRAM ciao(output);
    BEGIN
        writeln(‘Salve!’)
    END.
```

- Stampa a video un saluto

Istruzione di Uscita
Esercizi

- Produrre dei programmi per stampare:
  - Il messaggio “buon lavoro” una parola per rigo
  - La parola “programmazione” una lettera per rigo
  - Il messaggio “Salve, mondo!” sullo stesso rigo
    - Usare una **write** per ciascuna parola
  - Il messaggio “ciao a tutti” una parola per rigo
    - Allineare le parole a destra

  andando a capo con un rigo vuoto al termine della stampa
- Inserire commenti nei programmi prodotti

Tipi

- Tipizzazione forte
  - Rigida distinzione fra tipi
  - Rigoroso controllo di compatibilità nelle operazioni su variabili
- Tipizzazione statica
  - Ogni variabile è associata ad un unico tipo durante tutta l’esecuzione del programma

Tipi Semplici
Predefiniti

- Scalari
  - Integer
  - Boolean
    - **False**, **True**
  - Char
- Real

Tipi Semplici
Predefiniti

- Integer
  - Sequenza di cifre, preceduta o meno dal segno
    - Massimo intero rappresentabile: **maxint**
      - Dipendente dall'implementazione
      - Visualizzabile tramite l'istruzione
        writeln('Il massimo intero rappresentabile e'' ', maxint)
- Boolean
  - False < True
  - ord(False) = 0

Identificatori

- Nomi delle risorse usate dal programma
  - Costanti, Tipi, Variabili, Procedure, Funzioni
- Iniziano con una lettera (non accentata), eventualmente seguita da una sequenza di
  - Lettere (non accentate)
  - Cifre
  - Caratteri di sottolineatura singoli (_)
- Interamente significativi

Dichiarazioni di Variabili

x **:** T;

- Dove *x* è l'identificatore e *T* è il tipo della variabile

x, y, z **:** T;

- Forma abbreviata nel caso che più variabili abbiano il medesimo tipo

x, y, z **:** T_1;

u, v **:** T_2;

- Forma abbreviata che raggruppa più variabili del medesimo tipo

Dichiarazioni di Costanti

- c = v;
  - Dove *c* è l'identificatore e *v* è il valore della costante
  - Tipo individuato dalla forma della costante
    - (±) Numero senza segno
    - (±) Identificatore di costante
    - (sequenza di) caratteri fra apici
    - **nil**
  - Non ammesse espressioni

Istruzioni di Uscita

- La stampa dei boolean produce le stringhe corrispondenti
- Per inserire un apice nel testo da stampare
  - Raddoppiarlo
    - Evita confusione con quello di chiusura

      write('…''…')
- Per inserire un 'a capo' senza scrivere nulla
  - Lista di output vuota
    - Si omette

      writeln

Istruzioni di Uscita
Formattazione

- e[: w[: f]]
  - *e* valore da scrivere
    - Espressione intera, reale, booleana o stringa
  - *w* larghezza di campo minima
    - Espressione intera positiva
    - Indica il numero di caratteri da scrivere
      - Spazi iniziali se necessario
      - Valore di default in base al tipo di *e*
  - *f* lunghezza di frazione
    - Espressione intera positiva
    - Applicabile solo ai real

Istruzione di Uscita
Formattazione

- Impostazioni predefinite per *w*
  - integer: Dipendente dall'implementazione
    - Riempiti con spazi i caratteri superflui
      - Numero allineato a destra
    - Se è troppo corto, il numero viene comunque scritto tutto
  - stringhe: Lunghezza della stringa

Istruzione di Uscita
Formattazione

- Impostazioni predefinite per *w*
  - boolean: Dipendente dall'implementazione
    - Maiuscole e minuscole dipendenti dall'implementazione
  - real: Dipendente dall'implementazione
    - Se è troppo corto, il numero viene comunque scritto tutto
      - Se f è specificata, viene scritto in virgola fissa
      - Altrimenti viene scritto in virgola mobile (forma esponenziale)

Istruzione di Uscita
Esempi Formattazione

- *A* variabile intera con valore 10
- *Writeln(A)* scrive 10
- *Writeln(A:5)* scrive _ _ _10
- *Writeln(A:10)* scrive _ _ _ _ _ _ _ _10

Istruzione di Uscita
Esempi Formattazione

- *A* variabile real con valore 10
- *Writeln(A)* scrive 1.0000000..0E+001
- *Writeln(A:5:3)* scrive 10.000
- *Writeln(A:10:5)* scrive _ _10.00000

Istruzione di Uscita
Esempi Formattazione

- Stampare una stringa centrata rispetto alla lunghezza della linea (80 caratteri)
  - *writeln* (stringa: 40 – length(stringa)/2)

Istruzioni di ingresso

read(*lista_di_input*)

- Immette i dati nelle variabili della lista
  - Dati separati da caratteri di spaziatura
- L'esecuzione del programma è sospesa finché non sono immessi i dati richiesti

- Si possono acquisire i tipi semplici predefiniti
  - Non si possono acquisire i boolean

Istruzioni di Ingresso

readln(*lista_di_input*)

- Come read
- Salta il resto della linea in corso dopo l'ultimo dato
  - Ignorati eventuali ulteriori dati fino alla fine della linea

- Per fermare l'esecuzione del programma in attesa di un ingresso
  - **readln**

Istruzioni di ingresso
Esempi

- **read**(max)
- **read**(giorno, mese, anno)
  - 12 10 1972 333 4562
- **readln**(prezzo, sconto)

  30000 15 ~~465 33~~ ¶
- **readln**

Assegnamento

- Indicato dal simbolo **:=**
  - A sinistra: variabile da assegnare
  - A destra: espressione da calcolare ed assegnare alla variabile
- Richiesta la compatibilità di tipi fra variabile a sinistra e risultato dell'espressione a destra

Operatori per i Tipi Predefiniti

- Integer
  - Risultati integer
    - Somma **+**
    - Differenza **–**
    - Prodotto *****
    - Quoziente intero **DIV**
    - Resto **MOD**
- Real
  - Risultati real
    - Somma **+**
    - Differenza **–**
    - Prodotto *****
    - Divisione **/**

Operatori per i Tipi Predefiniti

- Boolean
  - Risultati boolean
    - Negazione **NOT**
    - Congiunzione **AND**
    - Disgiunzione **OR**
- Char
  - Nessun operatore
- Operatori relazionali
  - Producono boolean
    - Minoranza stretta <
    - Minoranza <=
    - Uguaglianza =
    - Maggioranza >=
    - Maggioranza stretta >
    - Disuguaglianza <>

Tipo Integer
DIV e MOD

- Operazioni esatte
  - DIV
    - Quoziente intero della divisione
  - MOD
    - Resto intero della divisione intera
      - x MOD y = x – ((x DIV y) * y)
    - Indefinita se il secondo operando è negativo o nullo

Tipo Integer
DIV e MOD

- Esempi

8 DIV 2 = 4	8 MOD 2 = 0
4 DIV 5 = 0	4 MOD 5 = 4
3 DIV 3 = 1	3 MOD 3 = 0
28 DIV 3 = 9	28 MOD 3 = 1
–13 DIV 6 = –2	–13 MOD 6 = 5
9 DIV –4 = –2	9 MOD –4 = indefinito

Applicazione degli operatori DIV e MOD ad un problema reale

- Possono essere utilizzati per scomporre un valore intero *n* nelle cifre che lo compongono
- Esempio
  - numero 126
    - *n mod 10* consente di ottenere le unità (cifra più a destra)
    - *(n div 10) mod 10* dà la cifra delle decine ovvero 2
    - *n div 10* consente di ottenere la cifra più significativa, ovvero quella più a sinistra

Programma Pascal
Somma

```
PROGRAM somma (input, output);
   (* legge due numeri interi e ne stampa la somma *)
VAR   x, y: integer; (* numeri da sommare *)
         risultato: integer; (* risultato della somma *)
BEGIN (* qui x, y e risultato hanno valore indefinito*)
   writeln(‘ inserisci due numeri interi’);
   read(x, y);
   risultato := x + y;
   writeln(‘ la somma di ‘, x, ’ e ‘, y, ’ e’’ ‘, risultato)
END.
```

Strutture di Controllo
Sequenza

begin *istruzione*; …; *istruzione* **end**

- BEGIN ed END fungono da parentesi sintattiche
- Istruzioni separate da ;
  - Separatore, non terminatore
  - Usare con cautela per evitare che il compilatore assuma erroneamente il termine di un'istruzione

Strutture di Controllo
Selezione

if *condizione* **then** *istruzione* [**else** *istruzione*]

- La condizione è un'espressione booleana
- L'istruzione dopo il THEN è eseguita se la condizione è verificata
- L'istruzione dopo l'ELSE è eseguita se la condizione non è verificata
  - Opzionale
  - In caso di nidificazione, senza ulteriori specificazioni si assume riferito all'ultimo IF

Strutture di Controllo
Selezione

case *espressione* **of**
lista_di_costanti **:** *istruzione***;**
…
lista_di_costanti : *istruzione*
end

- Espressione di tipo scalare
  - Costanti di tipo compatibile con l'espressione
- Liste di costanti mutuamente esclusive
  - Deve verificarsi sempre la corrispondenza con l'espressione

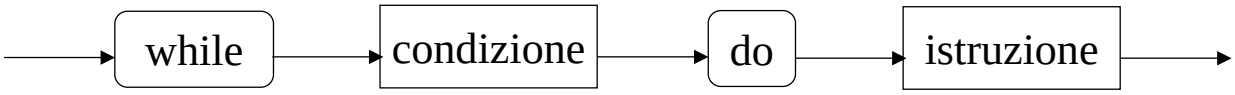
Strutture di Controllo
Iterazione

while *condizione* **do** *istruzione*

- La condizione è un'espressione booleana
  - Il ciclo viene intrapreso quando viene verificata
- Se l'istruzione da eseguire è una sequenza, necessita delle parentesi BEGIN … END

Strutture di Controllo
Iterazione – Carta sintattica

while *condizione* **do** *istruzione*



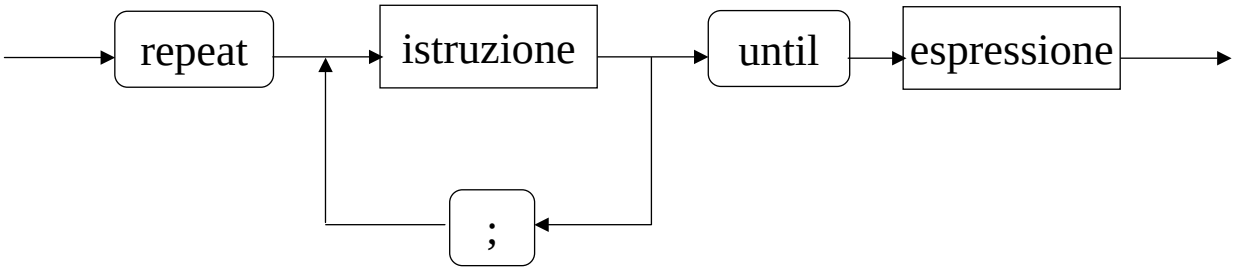
Strutture di Controllo
Iterazione

repeat *istruzione* **until** *condizione*

- Se l'istruzione da ripetere è una sequenza, non necessita delle parentesi BEGIN … END
- La condizione di terminazione è un'espressione booleana
  - Il ciclo termina quando viene verificata

Strutture di Controllo
Iterazione – Carta sintattica

repeat *istruzione* **until** *condizione*



Strutture di Controllo
Iterazione Limitata

for *indice* **:=** *espressione*
[**down**]**to** *espressione* **do** *istruzione*

- L'indice è uno scalare
  - Incrementato di 1 ad ogni ciclo con TO
  - Decrementato di 1 ad ogni ciclo con DOWNTO
- Espressioni di un tipo compatibile con l'indice
- VIETATO agire sull'indice nell'istruzione

Definizione di Tipi

nometipo = descrizione_tipo

- Tipi scalari
  - Enumerativi

    identificatore = (*lista_valori*)
    - Esempio:
      giorno = (lun, mar, mer, gio, ven, sab, dom)
  - Subrange

    identificatore = val_min .. val_max
    - Esempio:
      feriale = lun .. sab

Tipi Enumerativi

- Ciascuna costante può apparire in un solo tipo enumerativo
  - Non stampabili
- Gli ordinali associati alle costanti partono da 0
  - Ordinamento definito dalla sequenza di elencazione
- Funzioni predefinite:
  - Pred(X) costante elencata prima di X
  - Succ(X) costante elencata dopo X
  - Ord(X) ordinale di X

Tipi Sottocampo

- Risparmio di memoria
- Consentono controlli nell'esecuzione
  - Compatibili con il tipo ospite
    - Possibilità di errori di assegnazione
      - Assegnazione fuori dall'intervallo

Tipi Strutturati
Predefiniti

- Array (vettori)
- Record
- File
  - Text
- Set

Array

array[*tipo_indice*] **of** *tipo_base*

- Indici di tipo scalare
- Accesso

  nome_array[*indice*]
- Allocazione statica alla compilazione
  - Tipo fissato
  - Dimensione fissata

Array Multidimensionali

- Array aventi come componenti altri array
  - Dichiarazione

    array[*tipo_indice*] **of array** [*tipo_indice*] … **of** *tipo_base*
  - Accesso

    nome_array[i][j]…[k]
  - Abbreviazioni

    array[*tipo_indice*, …, *tipo_indice*] **of** *tipo_base*

    nome_array[i, j, …, k]

Array

- Notevole occupazione di memoria centrale
  - Possibilità di ridurla al minimo compattando gli elementi
    - Boolean, Char, Subrange
- Effetto dipendente
  - Dalla natura degli elementi
  - Dall'implementazione

Array
Impaccamento

- Compattazione di più elementi dell'array in una singola voce di memoria
  - Esempi
    Voci di memoria da 4 byte
    - Char
      - 7 +1 bit (ASCII) $\rightarrow$ 4 caratteri
    - Boolean
      - 1 bit $\rightarrow$ 32 valori logici
    - Subrange
      - 0..127 (4 bit) $\rightarrow$ 4 valori

Array
Impaccamento

packed array[*tipo_indice*] **of** *tipo_base*

- Compattamento delle componenti in memoria centrale
  - Ottimizzazione dell'uso di memoria
  - Inefficienza nell'accesso
    - Possibilità di perdere l'indicizzazione
- Esempi
  - A : **packed array**[-16..+15] **of boolean**
  - Cognome : **packed array**[1..32] **of char**
  - Esame : **packed array**[1..24] **of** voto
    - **type** voto = 0..30

Array
Impaccamento

- Procedure predefinite
  - **pack**(U, i, P)
    - Impacca l'array *U* nell'array impaccato *P* a partire dall'*i*-esimo elemento di *U*
  - **unpack**(P, U, i)
    - De-impacca l'array impaccato *P* a partire dal suo primo elemento, nell'array *U* a partire dall'*i*-esimo elemento

Record

record

identificatore : *tipo*;

…

identificatore : *tipo*;

end;

- Campi dello stesso tipo possono essere raggruppati

Record

- Accesso

  *nome_record***.***nome_campo*

  – Specificare il nome del record seguito da quello del campo desiderato
- Operazioni
  – Sui singoli campi
    - Operatori consentiti dal tipo del campo
  – Globali sui record
    - Solo in caso di strutture identiche impaccate

Record
Esempi

```
libro :  record
         autore : array[1..50] of char;
         titolo : array[1..70] of char;
         anno : integer;
         prezzo : real;
         prestato : boolean
         end;
complesso :   record
              parte_reale, parte_immaginaria : real
              end;
```


Record

with *nome_record* **do begin** *istruzioni* **end**

- Consente di manipolare i campi di un record specificando solo i loro nomi
  - Evita di specificare ogni volta il nome del record
  - Utile per operazioni multiple su uno stesso record

Istruzione With
Esempio

```
type    data = record
                 giorno : 1..31;
                 mese : 1..12;
                 anno : 1999..2500
               end;

var compleanno : data;
```

```
compleanno.giorno := 12;
compleanno.mese := 10;
compleanno.anno := 1972
```

equivale a

```
with compleanno do
    begin
        giorno := 12;
        mese := 10;
        anno := 1972
    end
```


Record con Varianti

- Schede aventi
  - Una parte in comune
    - Eventualmente nulla
  - Una parte che può contenere campi diversi
    - Utilizzabili in situazioni diverse
- Possibilità di riunirle in un unico tipo
  - Lettura di un valore qualunque sia il sottotipo

Record con Varianti
Esempio

```
contribuente : record
    imponibile : intero;
    categoria : (persona, ditta);
    (  persona: (       nome : array[1..30] of char;
                        cod_fiscale : array[1..16] of char      )
    )
    (  ditta : (        ragione_sociale : array[1..40] of char;
                        partita_iva : array[1..11] of char      )
    )
    end
```


Sottoprogrammi

- 2 tipi:
  - Procedure

procedure *nome_procedura***(***lista_parametri***);**

  - Funzioni

function *nome_funzione***(***lista_parametri***):** *tipo_funzione***;**

- Entrambe vanno dichiarate prima del loro uso
  - Stesse regole generali dei programmi Pascal
  - Le istruzioni saranno eseguite quando vengono invocate

Sottoprogrammi

- Procedure
  - Assimilabili a nuove istruzioni che si aggiungono al linguaggio
    - Ritagliate sulle esigenze dell'utente
- Funzioni
  - Come sopra, ma
    - Consentono di esprimere trasformazioni di dati direttamente nelle espressioni
    - Restituiscono un valore di un tipo semplice
      - Assegnamento all'identificatore della funzione

Function in Pascal
BNF

<DichiarazioneFunzione>::=<IntestazioneFunzione> ";" Blocco|
<IntestazioneFunzione>";" Direttiva|
....
<IntestazioneFunzione>::="function" Identificatore
[ListaParametriFormali] ":" TipoRisultato
<ListaParametriFormali>::="("<SezioneParametriFormali>
{";"<SezioneParametriFormali>}")"
…

Procedure Predefinite

- Pack(u,i,p)
  - Compressione di un array
- Read(f,v)
- Read(f,v1,…,vn)
  - Lettura dati
- Readln
- Readln(f,v1,…,vn)
  - Lettura dati ignorando il resto della linea
- Unpack(p,u,i)
  - Decompressione di un array
- Write(f,v)
- Write(f,v1,…,vn)
  - Scrittura dati
- Writeln
- Writeln(f,v1,…,vn)
  - Scrittura dati andando a capo al termine

Funzioni Predefinite

- Abs(x)
  - Valore assoluto
- Arctan(x)
  - Arcotangente
- Sin(x)
  - Seno
- Exp(x)
  - Esponenziale
- Ln(x)
  - Logaritmo naturale
- Odd(i)
  - Parità
- Round(r)
  - Arrotondamento
- Trunc(r)
  - Troncamento
- Sqr(x)
  - Quadrato
- Sqrt(x)
  - Radice quadrata

Funzioni predefinite

- Ord(i)
  - Ordinale
- Pred(x)
  - Ordinale precedente
- Succ(x)
  - Ordinale successivo
- Chr(i)
  - Carattere corrispondente ad un ordinale

Nidificazione di Sottoprogrammi

- Ciascun sottoprogramma può contenere, nella propria sezione dichiarativa, la dichiarazione di altri sottoprogrammi
  - Insieme dei sottoprogrammi strutturato ad albero
- Visibilità
  - Ciascun sottoprogramma può richiamare soltanto:
    - I sottoprogrammi che esso dichiara (figli)
    - I sottoprogrammi dichiarati dal sottoprogramma che lo dichiara (fratelli)

Schema di Programma
Dichiarazione di Sottoprogrammi

```
program nome_programma (input, output);
    { dichiarazioni di tipi e variabili }
    dichiarazione di sottoprogrammi
        blocco programma 1
                sotto blocco programma 1.1
                sotto blocco programma 2.1
                        sottoblocco programma 1.2.1
                        sottoblocco programma 1.2.2
        blocco programma 2
    begin
        { corpo programma principale }
    end.
```


Sottoprogrammi
Comunicazione

- I dati calcolati possono essere restituiti al programma chiamante
  - Assegnando valori a variabili comuni al programma chiamante e al programma chiamato
    - Variabili non locali
  - Alterando i parametri effettivi
    - Devono essere variabili

Passaggio di Parametri

- Il Pascal consente passaggi:
  - Per valore
  - Per riferimento completo (read/write)
- ed inoltre consente
  - Di passare come parametri formali procedure o funzioni
  - Di passare parametri stringa senza una definita lunghezza

Passaggio di Parametri
per Valore

- Non sono introdotti da nessuna specificazione
- Possono essere pensati come variabili locali inizializzate dal programma chiamante
  - Il sottoprogramma invocato può modificarli al suo interno
  - Eventuali cambiamenti non si riflettono all'indietro sul programma chiamante

Passaggio di Parametri
per Riferimento Completo

- Introdotti dalla clausola **var**
- Riflettono le modifiche, avvenute nel programma chiamato, indietro nel chiamante
  - Possono essere usati come input e output
  - Applicabile solo alle variabili
    - NO espressioni, costanti, elementi di variabili strutturate impaccate, …

Passaggio di Parametri
Gestione

- Per valore
  - Al momento della chiamata di procedura
    - Viene allocata una nuova area di memoria
    - Il valore corrente del parametro effettivo è copiato in questa locazione
  - L'area viene rilasciata all'uscita dalla procedura
- Per riferimento
  - Al momento della chiamata di procedura
    - Vengono eseguiti tutti i calcoli degli indirizzi
  - Ogni operazione che coinvolge il parametro formale è in realtà fatta direttamente sul parametro effettivo

Variabili Locali e Non Locali
Esempio

```
program S (input, output);
    var x: integer;
    procedure cambia;
        begin
            x := 1
        end;
begin
    x := 0;
    cambia;
    write(x)
end.
```

→ *1*

```
program S (input, output);
    var x: integer;
    procedure cambia;
        var x: integer;
        begin
            x := 1
        end;
begin
    x := 0;
    cambia;
    write(x)
end.
```

→ *0*

Parametri
Esempio

```
program S (input, output);
    var x: integer;
    procedure cambia(var y: integer);
        begin
            y := 1
        end;
begin
    x := 0;
    cambia(x);
    write(x)
end.
```

→ *1*

```
program S (input, output);
    var x: integer;
    procedure cambia(y: integer);
        begin
            y := 1
        end;
begin
    x := 0;
    cambia(x);
    write(x)
end.
```

→ *0*

Effetti Collaterali
Esempio

```
program side_effect(input, output);
    var b: integer;
    function f(var a: integer): integer;
            begin
                    a := 2 * a;
                    f := a;
            end;
begin
    b := 1;
    writeln(2 * f(b));          {1}
    b := 1;
    writeln(f(b) + f(b))        {2}
end.
```

- La funzione *f*, attivata sul parametro effettivo *b*, ne calcola il doppio
  1. Stampa 4, il quadruplo di *b*
  2. Stampa 6, perché la seconda applicazione di *f* ha in ingresso un valore di *b* modificato a 2, quindi fornisce il valore 4

Tipo Stringa

- Il Pascal prevede un tipo denominato **string**
  - Packed array of char
- NON sono disponibili operatori su stringa
  - Concatenazione, estrazione di sottostringhe, …

Passaggio di Parametri
Stringhe

- Parametri di tipo specificato **string**
  - Lunghezza nel parametro formale non specificata
    - Consentono di chiamare un sottoprogramma passandogli una stringa che può avere una lunghezza di volta in volta differente
  - Utilizzabile sia che si usi il metodo di passaggio **var** sia che si usi il metodo di passaggio **const**

Passaggio di Parametri
Procedure e Funzioni

- Per definire parametri formali che siano a loro volta sottoprogrammi va specificata la forma della loro intestazione
  - Nomi fittizi per il sottoprogramma ed i parametri
  - Numero, tipo e ordine dei parametri devono coincidere con quelli dei parametri reali
- Attenzione!
  - Non possono apparire come parametri formali procedure e funzioni standard (*built-in*)

Sottoprogrammi come Parametri

procedure P (**procedure** A);

- Procedura *P* con un parametro formale *A* che non ha parametri
  - Nel corpo di *P* l'identificatore *A* è usato come un qualunque altro identificatore di procedura non parametrica

procedure Q(**function** F(x: **integer**): **integer**);

- Procedura *Q* con un parametro formale *F* che è una funzione intera di variabile intera
  - Nel corpo di *Q* l'identificatore *F* è usato come un qualsiasi altro indicatore di funzione di quel tipo, cioè come designatore di una funzione

Sottoprogrammi come Parametri
Esempio

```
procedure tabula(function f(X: real): real; lower, upper, step: real);
    var   x: real;
          j: integer;
begin
    x := lower;
    for j := 0 to trunc((upper-lower)/step) do
          begin
                writeln(x:13, f(x):20);
                x := x + step
          end
end;
```

Sottoprogrammi come Parametri
Esempio

- La procedura *tabula*
  - Genera una successione di valori per *x*
    - Determinati sui valori dei parametri *lower*, *upper* e *step*
  - Calcola e stampa *f(x)* per ognuno dei valori
    - La definizione formale di tabula non cambia a seconda della funzione *f*
- tabula(f(1), 0.0, 1.0, 0.01)
  - Tabulazione di valori di *x* e *f(x)*, compresi nell'intervallo 0÷1 con passo 0.01
    - Possibili funzioni: radice quadrata, potenza, coseno, …

Ricorsione
Calcolo del Fattoriale

- Versione iterativa

```
funcion fatt(n: integer): integer;
    var   fattore, accum: integer;
begin
    accum := 1;
    for fattore := 2 to n do
            accum := accum * fattore;
    fatt := accum
end;
```

- Versione ricorsiva

```
function fatt(n: integer): integer;
    var accum: integer;
begin
if n=0 then
            accum := 1
    else
            accum := n * fatt(n-1);
fatt := accum
end;
```

Direttiva Forward

- Ogni sottoprogramma vede solo quelli che lo precedono nella lista delle dichiarazioni
  - Necessità di tenerne conto nell'ordine di dichiarazione
- Capita che un sottoprogramma debba richiamarne un altro dichiarato in seguito
  - Possibile solo fra sottoprogrammi dichiarati da uno stesso sottoprogramma

Direttiva Forward

- Dichiara al compilatore l'esistenza di un sottoprogramma prima che esso sia effettivamente definito
  - Utile quando è necessario chiamare un sottoprogramma all'interno di altri precedenti
- Al fine dei controlli sintattici sull'uso, deve specificare esattamente l'interfaccia del sottoprogramma in questione
  - Identificatore
  - Lista e tipo degli argomenti
  - Tipo di ritorno (per le funzioni)

Direttiva Forward
Uso

- L'intestazione del sottoprogramma è dichiarata senza dichiararne la parte dichiarativa ed il corpo
  - Seguita dalla direttiva **forward**
- Si dichiarano altri sottoprogrammi che possono chiamare il sottoprogramma in questione
- Al momento della dichiarazione completa del corpo della procedura la dichiarazione avviene normalmente solo che si tralascia la lista dei parametri formali

Direttiva Forward
Esempio

- Siano *P* e *Q* due sottoprogrammi che si richiamano a vicenda (*mutua ricorsione*)
- La dichiarazione assumerà la forma seguente
  - Intestazione del sottoprogramma *P*
    - Completa della lista dei parametri formali
  - **forward;**
  - Definizione del sottoprogramma *Q*
    - All'interno richiama *P*
  - Definizione del sottoprogramma *P*
    - Si tralascia la lista dei parametri formali

Sottoprogrammi ricorsivi ed iterativi

Procedura di ricerca binaria iterativa

```
procedure bsit (v: vettore; x: real; a,z: integer; VAR esito:boolean);
var m:integer;
begin
if a > z then esito := false
else
  begin
    repeat
        m := (a+z) div 2;
        if x < v[m] then z := m-1 else a := m+1
    until (a>z) or (v[m]=x);
   esito := (v[m]=x)
 end
end;
```


Sottoprogrammi ricorsivi ed iterativi

Funzione di ricerca binaria iterativa

```
function bsit (v: vettore; x: real; a,z: integer):boolean;
var m:integer;
begin
if a > z then bsit := false
else
  begin
    repeat
        m := (a+z) div 2;
        if x < v[m] then z := m-1 else a := m+1
    until (a>z) or (v[m]=x);
   bsit := (v[m]=x)
 end
end;
```

Sottoprogrammi ricorsivi ed iterativi

Procedura di ricerca binaria ricorsiva

```
procedure bsr (v: vettore; x: real; a,z: integer; VAR esito:boolean);
var m:integer;
begin
if a > z then esito := false
else
  begin
     m := (a+z) div 2;
     if  v[m] = x then esito:= true
     else
        if x < v[m] then bsr(v, x, a, m-1, esito)
        else bsr (v, x, m+1, z, esito)
   end
end;
```


Sottoprogrammi ricorsivi ed iterativi

Funzione di ricerca binaria ricorsiva

```
function bsr (v: vettore; x: real; a,z: integer):boolean;
var m:integer;
begin
if a > z then esito := false
else
  begin
     m := (a+z) div 2;
     if  v[m] = x then bsr:= true
     else
        if x < v[m] then bsr:=bsr(v, x, a, m-1)
        else bsr:=bsr (v, x, m+1, z)
   end
end;
```

Unità di compilazione

- Permettono di suddividere i programmi in parti più semplici e maneggevoli, favorendo così la *programmazione modulare*
- Una *unit* è costituita da un insieme di costanti, tipi, variabili e sottoprogrammi che può essere memorizzato in un file e compilato separatamente

Unità di compilazione

- Estensione del Pascal standard
- Unità compilabili separatamente
  - Keyword *UNIT*
- Si dividono in due parti
  - Interfaccia
    - Dichiarazioni esportabili (globali a tutta la unità)
  - Implementazione
    - Definizione dei sottoprogrammi dichiarati e di costanti, tipi, variabili e sottoprogrammi non esportabili (locali)
- Opzionalmente può comparire una sezione di inizializzazione, racchiusa in un blocco *begin-end*

Unità di compilazione
Struttura

unit <ID-Unit>
interface
  <dichiarazione-costanti>
  <dichiarazione-tipi>
  <dichiarazione-variabili>
  <definizione-sottoprogrammi>
implementation
  <dichiarazione-costanti>
  <dichiarazione-tipi>
  <dichiarazione-variabili>
  <definizione-sottoprogrammi>
[**begin** {inizializzazioni} **end**]

Unità di compilazione
Esempio

unit Servizio_Vettori;

interface
procedure shell (var a:Tvettore; n:integer);
procedure bubble (var a:Tvettore; n:integer);
procedure leggi (var a:Tvettore; var n:integer);
procedure stampa (a:Tvettore; n:integer);
function cerca (x:Tbase; var a:Tvettore;
n:integer): boolean;

Unità di compilazione
Esempio

```
implementation
procedure shell      (var a:Tvettore; n:integer);
begin . . . end;
procedure bubble     (var a:Tvettore; n:integer);
begin . . . end;
procedure leggi      (var a:Tvettore; var n:integer);
begin . . . end;
procedure stampa     (a:Tvettore; n:integer);
begin . . . end;
function cerca              (x:Tbase; a:Tvettore;
                  n:integer): boolean;
begin . . . end;
```


Unità di compilazione
Utilizzo

```
program import_unit(input, output);
   uses Servizio_Vettori;
   …
begin
   bubblesort (…)
end.
```

Insiemi

- Il Pascal consente di dichiarare tipi insieme
  - Denominati **set**
- L'identificatore tipo set è definito sui valori specificati in un altro tipo
  - *Tipo base*
- La definizione dei valori può avvenire
  - Elencandoli singolarmente
  - Specificando un intervallo del tipo base

Set

set of *tipo_base*

- Una variabile dichiarata di tipo set è un insieme i cui elementi sono scelti dai valori in tipo base
  - Restrizioni sui tipi base
    - Deve essere enumerabile
      - Ordinale
    - Alcune implementazioni impongono un limite sul numero dei valori del tipo base

Set
Definizione

- Il tipo base può essere definito direttamente nella dichiarazione di set
  - Per enumerazione

```
type    carte = (c1,c2,…,ck,q1,q2,…,qk,p1,p2,…pk,f1,f2,…,fk);
        mano = set of carte;
```

  - Come sottocampo

```
type    maiuscole = set of ‘A’..’Z’
```


Set
Esempi

```
type    mese = (gen,feb,mar,apr,mag,giu,lug,ago,set,ott,nov,dic);
        setmese = set of mese;
var     inverno, estate: setmese;
begin
   inverno := [dic, gen, feb];
   estate := [giu..ago]
end.
```

Set
Operazioni Disponibili

- Assegnamento **:=**
- Confronti
  - Eguaglianza =
  - Differenza < >
  - Inclusione <=
    >=
  - Inclusione stretta <
    >
- Operatori (Priorità decrescente)
  - Intersezione *****
  - Unione **+**
  - Differenza **–**
  - Appartenenza **IN**
    - Si applica ad un elemento e un insieme

Set
Operazioni Disponibili

- Esempio

```
type    ins = set of (a, b, c);
var     x, y: ins;

    x := [a, c]
    y := [a, b]
```

x = y	false
x < [a,b,c]	true
x < [a]	false
x > [c]	true
x < y	false
x * y è l'insieme	[a]
x + y è l'insieme	[a,b,c]
x – y è l'insieme	[c]
if (a IN x) …	true
if (b IN x) …	false

Set
Costruzione

- Inizializzazione
  - Specifica degli elementi (o *membri*)

```
type      colore = (rosso,giallo,blu);
var       colorecaldo: set of colore;
 begin
    col := [rosso,giallo]
end.
```

- Aggiunta di un elemento
  - Somma dell'insieme unitario contenente l'elemento da inserire

```
type      numeri = 1..10;
var       primi: set of numeri;
 begin
    primi := [1, 2, 3, 5] + [7]
end.
```


Operazioni su Set
Esempi

- * (intersezione o prodotto insiemistico)
  - [a, b, c] * [a, c] = [a, c]
  - [a, c] * [a, b] = [a]
  - [a, c] * [b] = [ ]
- + (unione o somma insiemistica)
  - [a, b] + [a, c] = [a, b, c]
  - [a, c] + [b, d] = [a, b, c, d]
  - [a, b] + [b] = [a, b]
- – (differenza)
  - [a, b, c] – [a, c] = [b]
  - [a, b] – [c] = [a, b]
  - [a, b] – [a, b, c] = [ ]
- IN (appartenenza)
  - 10 IN [5..25] true
  - 'r' IN ['a', 'e', 'i', 'o', 'u'] false

Confronti fra Set
Esempio

- = Uguaglianza
  - [a, b] = [a, b] true
  - [a, b] = [a, c] false
  - [a, b] = [b, a] true
  - [] = [b] false
- < Sottoinsieme proprio
  - [a, b, c] < [a, b] false
  - [a, b] < [a, b, c] true
  - [a, b] < [a, c] false
  - [a, b] < [a, b] false
- <= Sottoinsieme
  - [a, b] <= [a, b] true
- > Sovrainsieme proprio
  - [a, b] > [a, b, c] false
  - [a, b, c] > [a, b] true
  - [a, c] > [a, b] false
  - [a, b] > [a, b] false
- >= Sovrainsieme
  - [a, b] >= [a, b] true
- < > Disuguaglianza
  - [a, b] < > [a, b] false
  - [a, b] < > [b, a] false
  - [b] < > [a, b] true

Set
Usi

- Assegnamento
  - Inizializzazione
    S := []
  - Aggiornamento
    - Aggiunta
      S := S + [elemento]
    - Rimozione
      S := S – [elemento]
  - Operatore globale
    S1 := S2
- Appartenenza
  - Facilita la scrittura di espressioni booleane

c = ‘a’ OR c = ‘e’ OR c = ‘i’ OR
c = ‘o’ OR c = ‘u’

equivale a

c IN [‘a’, ‘e’, ‘i’, ‘o’, ‘u’]

Set
Esempi

- Insiemi dei numeri pari e dispari nell'intervallo da 1 a *maxnum*

```
program pari_dispari(input,output);
    const       maxnum = 60;
    type        intset = set of 1..maxnum;
    var         pari, dispari: intset;
                i: integer;
begin
    dispari := []; i := 1;
    while i <= maxnum do
        begin   dispari := dispari + [i]; i := i + 2    end;
    pari := [1..maxnum] – dispari
end.
```


File

- Insiemi di valori, indicati da un unico nome e strutturati
  - Indicano successioni di dati del medesimo tipo
  - Come per tutte le variabili strutturate, nella dichiarazione è necessario indicare
    - Il tipo di struttura
    - Il tipo delle componenti

File

file of *tipo_base*

- Un file le cui componenti sono tutte di tipo *tipo_base*

- Il Pascal prevede solo file sequenziali
  - Implementazione del concetto astratto di sequenza
    - Manca l'operatore di concatenazione
  - Astrazione dei supporti di memoria sequenziale

File

- La dichiarazione di una variabile di tipo file alloca in memoria lo spazio necessario a contenere un singolo elemento del tipo base
  - Unico elemento direttamente accessibile in ogni istante
  - Tutte le operazioni sul file devono passare da tale variabile
- Buffer tampone
  - Identificato da *nome_file*^
    - Puntatore all'area di memoria corrispondente

File
Operatori di Apertura

- Rewrite(x)
  - Costruisce un file vuoto di nome *x*
    - Se, prima dell'istruzione, esisteva già un file con eguale nome, questo viene sprotetto (cancellato)
- Reset(x)
  - Consente di posizionarsi sul primo elemento del file *x* per iniziare la scansione
- Il tipo dell'ultima apertura di un file vincola il tipo di operazione che vi si può effettuare

File
Scrittura

- Put(x)
  - Concatena al file il contenuto del buffer tampone ad esso associato
    - Il file deve essere stato aperto in scrittura
- Aggiunta di un valore in coda al file *x*
  - Si riempie il buffer tampone col valore da inserire
  - Si aggiunge il buffer tampone in coda al file

x^ := y put(x)

File
Lettura

- Get(x)
  - La scansione passa all'elemento successivo, che viene copiato nel buffer tampone
    - Il file deve essere stato aperto in lettura
- Lettura del valore successivo nel file *x*
  - Si sposta la "testina" sull'elemento successivo, che viene inserito nel buffer tampone
  - Si copia il buffer tampone in una variabile

get(x) y := x^

File
Fine Sequenza

- Eof(x)
- Funzione booleana predefinita
  - Applicata ad un file *x*

Eof(x) = true sse x_d = < >

  - Restituisce i valori
    - **true** se il file sequenziale è finito
    - **false** altrimenti

File
Schemi di Programma

- Scrittura di un file *x*

```
rewrite(x);
while condizione do
   begin
      x^ := …;
      put(x)
   end;
```

- Lettura di un file *x*

```
reset(x);
while not eof(x) do
   begin
      operazione(x^)
      get(x);
   end;
```


File
Esempio

- Determinazione della lunghezza L di un file *x*

```
reset(x);
L ← 0;
while not eof(x) do
   begin
      L ← L + 1;
      get(x);
   end;
```

File

- Solitamente usati per contenere grandi quantità di dati
  - Troppo grandi per essere contenute in memoria centrale
- Non esiste un'istruzione di assegnamento tra file
  - Impossibilità di fare una copia locale di un intero file in memoria centrale

Aprire in lettura il file originale ed in scrittura il file copia
Finché non si è raggiunta la fine del file originale
    leggere un elemento dal file originale ed inserirlo in quello copia

File
come Parametri di Sottoprogrammi

- Ogni file specificato come parametro di procedura va passato sempre per referenza (**var**)
  - Viene in realtà passato l'indirizzo (di memoria centrale) del buffer associato al file

  indipendentemente dal fatto che sia usato come ingresso o come uscita
  - Il passaggio per valore implicherebbe un assegnamento del parametro (file) attuale al parametro formale corrispondente
    - Impossibile

File Esterni

- assign(*nome_file_logico*,*nome_file_fisico*);
  - nome_file_logico
    - Identificatore del file come dichiarato nel programma
  - nome_file_fisico
    - Stringa contenente fra apici singoli il nome del file come individuato dal file system
      - Eventualmente inclusiva di percorso

File Esterni

- I parametri dell'intestazione di programma possono includere identificatori di file logici
  - Non va dichiarato il tipo nell'intestazione
    - Va dichiarato nella sezione dichiarativa
  - Necessità di associargli un file fisico all'interno del programma
- Esempio

```
program file_esterni(input, output, f, g);
    type  tipo_file = file of integer;
    var   f, g: tipo_file;
```

File di Record
Esempio

```
const   dimstring = 20;
type    intcod = 1111..9999;
        string = packed array[1..dimstring] of char;
        libro = record
                    codnum: intcod;
                    autore, titolo: string;
                    prezzo: real;
                    quantita: integer
                    end;
        biblio = file of libro;
var     inventario, invsag : biblio;
        saggio: libro;
```


File di Record
Esempio

- Ricerca di un definito elemento
  - Va aggiunta una variabile

```
    elemento: libro;
    trovato: boolean;
begin
    trovato := false;
    read(elemento);
    reset(inventario);
    while (not eof(inventario)) and (not trovato) do
        if inventario^ = elemento then
                trovato := true
        else
                get(inventario)
end.
```

File Testo

text

- Tipo predefinito del Pascal
  - File le cui componenti sono caratteri
    - Raggruppati in sequenze, definibili come linee
      - Marcate da uno speciale carattere di fine linea
  - Differente da *file of char*
    - Prevede uno speciale carattere di fine linea

File Testo

eoln(·)

- Funzione booleana che testa la condizione di fine linea
  - **true** se la "testina" di lettura-scrittura del file è alla fine della linea
  - Usata per scandire le linee di un file text
- Possono essere creati usando l'editor di sistema così come per i programmi
  - I caratteri componenti sono immessi da tastiera
    - Carattere di fine linea ottenuto premendo

      <*return*> o <*invio*>
    - Carattere di fine file ottenuto premendo

      <*control*> + Z

File Testo

- Tipica struttura di programma

```
while not eof(x) do
    begin   azione1;
        while not eoln(x) do
            azione2;
        azione4
    end;
```


File Testo
Procedure di creazione/scansione

- Sui file di tipo text sono disponibili degli speciali operatori per la creazione e la scansione
  - Oltre ai normali operatori di base get(·) e put(·)
- Il file va opportunamente inizializzato in lettura o scrittura prima di poterli usare

File Testo
Procedure di creazione/scansione

- **write**(*outfile*, *dato*)
  - I caratteri che costituiscono il dato specificato devono essere tipi standard o stringhe
    - **char**
      - Singolo carattere
    - stringhe
      - Sequenze di caratteri racchiuse fra apici singoli
    - **integer**, **subrange**, **real**, **boolean**
      - Sequenza di caratteri numerici, poi convertita in binario

File Testo
Procedure di creazione/scansione

- **read**(*infile*, *dato*)
  - Legge da *infile* una sequenza di caratteri nella variabile specificata *dato*
    - Deve essere di un tipo standard
    - **char**
      - Viene letto un solo carattere alla volta
    - **integer**, **subrange**, **real**
      - Viene letta una sequenza di caratteri numerici, convertiti in binario e poi memorizzati nella variabile
  - La testina avanza dopo l'ultimo carattere letto

File Testo
Procedure di creazione/scansione

- Generalizzazioni
  - **write**(*outfile*, *lista_output*)
    - Abbreviazione di una sequenza di **write**
      - Una per ciascun elemento di *lista_output*
  - **read**(*infile*, *lista_input*)
    - Abbreviazione per una sequenza di **read**
      - Legge un dato per ciascun elemento di *lista_input*
      - Dati separati da caratteri di spaziatura o fine linea

File Testo
Procedure di creazione/scansione

- Versioni che gestiscono la fine della linea

  writeln(*outfile*, *dato*)

  writeln(*outfile*, *lista_output*)

  writeln(*outfile*)

  – Impone a vera **eoln**(*outfile*)

  readln(*infile*, *dato*)

  readln(*infile*, *lista_input*)

  readln(*infile*)

  – La testina avanza al primo carattere della riga successiva

File Testo
Standard

- I file **input** e **output** sono file standard di tipo text

  var input, output: **text**

- Gli operatori standard, nel caso non appaia esplicitamente il nome del file, si intendono applicati ai file di testo standard
  - Read applicata ad Input
  - Write applicata ad Output

File Testo

- Supponiamo di indicare con *ch* una variabile di tipo **char**
  - **write**(*ch*) equivale a **write**(output, *ch*)
  - **read**(*ch*) " **read**(input, *ch*)
  - **writeln** " **writeln**(output)
  - **readln** " **read**(input)
  - **eof** " **eof**(input)
  - **eoln** " **eoln**(input)

File Testo
Programma per la Copia

```
program copia(infile, outfile);
    var   infile, outfile: text;
          nextchar:char;
begin
assign(infile, ‘in1.txt’); assign(outfile, ‘out1.txt’);
reset(infile); rewrite(outfile);
while not eof(infile) do
    begin
    while not eoln(infile) do
          begin     read(infile, nextchar); write(outfile, nextchar)          end;
          (*        outfile^ := infile^; put(outfile); get(infile);    *)
    writeln(outfile); readln(infile)
    end;
end.
```


Tipo Puntatore

- Rappresenta indirizzi di memoria
  - Variabili puntatore distinte dal simbolo ^
  - Necessità di indicare il tipo di dato che sarà contenuto nell’indirizzo puntato
- Consente di realizzare l’allocazione dinamica della memoria
  - Spazio riservato durante l’esecuzione
    - Dipendente dalle necessità
    - Dimensione variabile

Tipo Puntatore

identificatore **:** ^*tipo***;**

- Dichiarazione di una variabile puntatore
  - Contenente l'indirizzo di un dato del tipo specificato
- *identificatore*
  - Indirizzo di memoria
- *identificatore*^
  - Contenuto dell'indirizzo di memoria
- **NIL**
  - Indirizzo nullo (fittizio)

Allocazione Dinamica

- Basata sull'uso di puntatori
- **new**(*puntatore*)
  - Alloca un'area di memoria
    - Dimensione dell'area allocata sufficiente a contenere un dato del tipo contenuto in *puntatore*
    - Pone in *puntatore* l'indirizzo dell'area allocata
- **dispose**(*puntatore*)
  - Rilascia l'area di memoria puntata da *puntatore*

Allocazione Dinamica
Esempio

```
program puntatore (input, output)
   var  a: ^integer;
begin
  new (a);  (*allocazione memoria per la variabile intera a *)
  a^:=5;
  writeln(a);
  dispose (a); (* deallocazione memoria *)
end.
```


Ambiente Dev-Pascal