



# Fondamenti d'Informatica: Linguaggio While

Barbara Re, Phd

# Il linguaggio WHILE

---

- ▶ La computabilità come supporto ai moderni linguaggi di programmazione
- ▶ Si può generalizzare rispetto ad una vasta famiglia di linguaggi come ALGOL, PASCAL, C, C++, Java
- ▶ Di riferimento un semplice linguaggio di programmazione imperativo chiamato **WHILE**
  - ▶ E' adeguato per rappresentare tutte le **funzioni calcolabili**
  - ▶ **Tutte le variabili, così come i parametri di input ed output dei programmi sono interi non negativi**
    - ▶ Per non perdere di generalità qualunque parametro o variabile di un qualunque programma è sempre rappresentabile come una sequenza di bit in memoria e prendendo la rappresentazione numerica di questa sequenza di bit otteniamo un numero intero positivo

# Sintassi e Semantica

---

- ▶ Un linguaggio di programmazione è un linguaggio formale dotato di una sintassi e una semantica ben definita
- ▶ La **sintassi** è un insieme di regole che definiscono la “forma” di un linguaggio; esse definiscono come ottenere delle frasi sequenzializzando una serie di componenti fondamentali, dette parole
  - ▶ Applicando le regole della sintassi è possibile stabilire se una frase è corretta oppure no
- ▶ La semantica definisce il significato dei programmi sintatticamente corretti nel linguaggio di programmazione in questione

# Sintassi del linguaggio WHILE

---

Specifichiamo la sintassi del WHILE utilizzando le regole di produzione di una grammatica libera dal contesto

PROGRAM  $\rightarrow$  begin end | begin SEQCOMMANDS end

SEQCOMMANDS  $\rightarrow$  COMMAND | SEQCOMMANDS ; COMMAND

COMMAND  $\rightarrow$  ASSIGNEMENT | while TEST do COMMAND | PROGRAM

ASSIGNEMENT  $\rightarrow$  VAR:=0 | VAR:= s(VAR) | VAR :=pd (VAR)

TEST  $\rightarrow$  VAR  $\neq$  VAR

VAR  $\rightarrow$  LETTERA | VAR LETTERA | VAR CIFRA

LETTERA  $\rightarrow$  a | b | ... | z

CIFRE  $\rightarrow$  0 | 1 | ... | 9

# Esempio di utilizzo della sintassi

PROGRAM ->

begin SEQCOMMANDS end ->

begin SEQCOMMANDS ; COMMAND end ->

begin COMMAND ; COMMAND end ->

begin ASSIGNEMENT ; while TEST do COMMAND end ->

begin VAR:=0; while VAR ≠ VAR do PROGRAM end ->

begin VAR LETTERA :=0; while LETTERA ≠ LETTERA do begin SEQCOMMANDS end end ->

begin LETTERAc:=0; while f≠h do begin COMMAND end end ->

begin ac:=0; while f≠h do begin ASSIGNEMENT end end ->

begin ac:=0; while f≠h do begin VAR:= s(VAR) end end ->

begin ac:=0; while f≠h do begin VAR LETTERA := s(VAR CIFRA) end end ->

begin ac:=0; while f≠h do begin LETTERA LETTERA := s(LETTERA CIFRA) end end ->

begin ac:=0; while f≠h do begin LETTERA LETTERA := s(LETTERA CIFRA) end end ->

begin ac:=0; while f≠h do begin hf := s(t9) end end

```
begin
    ac:=0;
    while f≠h do
        begin
            hf := s(t9)
        end
    end
end
```

QUESTO CODICE HA VERAMENTE SENSO?

# Comandi

---

- ▶ Un programma **WHILE** è una sequenza, ordinata finita, eventualmente vuota) di comandi, separati da punti e virgole, preceduta da **begin** e seguita da **end**

I comandi sono definiti induttivamente

- ▶ Comandi di Assegnamento
- ▶ Comandi Composti
- ▶ Comandi While

# Comandi di Assegnamento

---

- ▶ Zero, s e pd costituiscono gli **operatori base del linguaggio**
- ▶ Nei programmi WHILE i valori assegnati alle variabili saranno sempre numeri naturali
  
- ▶ (Zero):  $X:=0$
  
- ▶ (s):  $X:=s(Y)$ 
  - ▶ E' la funzione di successore ( $s(n) = n + 1$  per ogni naturale n)
  
- ▶ (pd):  $X:=pd(Y)$ 
  - ▶ E' la funzione di predecessore ( $pd(n) = n - 1$  se  $n > 0$  e  $pd(0)=0$ )

# Comandi Composti

---

- ▶ Si tratta di istruzioni della forma

begin  $C_1$ ; ... ;  $C_N$  end

- ▶ Con  $N$  naturale e indica le istruzioni  $C_1, \dots, C_N$  devono essere eseguite in successione secondo l'ordine assegnato
- ▶ Ammettiamo anche il caso  $N = 0$  che si riduce sostanzialmente ad un'istruzione che lascia inalterato il valore di tutte le variabili

# Comandi While

---

While  $X \neq Y$  do Command

La semantica consiste nel ripetere l'esecuzione del comando descritto in Command per tutto il tempo in cui la variabile  $X$  assume un valore diverso dal valore della variabile  $Y$

**L'unico operatore di confronto tra variabili nel linguaggio While è almeno inizialmente  $\neq$**

# Macro Istruzioni

---

- ▶ Il potere espressivo dei programmi WHILE non è inferiore a quello di altri usuali linguaggi di programmazione almeno per quanto concerne l'aspetto computazionale
- ▶ E' quindi importante definire varie macro-istruzioni che possono agevolare la stesura di un programma qualora coerentemente con la sintassi non ci siano comandi primitivi utili
  - ▶ Assegnare il valore di una variabile Y ad un'altra variabile X
  - ▶ Assegnare il valore di una costante  $n > 0$  ad una variabile U,  
U:=n
  - ▶ ...

# Macro - assegnazione

---

- ▶ Assegnare il valore di una variabile  $Y$  ad un'altra variabile  $X$

begin

$X := s(Y) ;$

$X := pd(X)$



$X := Y$

end

- ▶ Assegnare il valore di una costante  $n > 0$  a una variabile  $U$  di fatto il comando  $U = s(U)$  deve essere ripetuto  $n$  volte

begin

$U := 0 ;$

$U := s(U) ;$

$\dots ;$

$U := s(U)$



$U := n$

end

# Macro - Somma

---

- ▶ Sommare il valore della variabile Y alla variabile X

Begin

U:=0 ;

while U ≠ Y do

begin

X:= s(X) ;

U:=s(U)

end

end



X:=X+Y

# Macro – Sommare due Variabili

---

- ▶ Sommare il valore della variabile Y e della variabile X assegnandolo alla variabile Z (se usiamo le macro già definite)

Begin

Z:= Y;

Z:= Z+X;

end



Z:=X+Y

Moltiplicare il valore  
della variabile X per  
la variabile Y,  $X:=X*Y$

$X:=X*Y$



```
begin
  u:=0;
  z:=0;
  y:=pd(y);
  while u ≠ x do
    begin
      u:=s(u);
    end
  while z ≠ y do
    begin
      h:=0;
      z:=s(z);
      while h ≠ u do
        begin
          x:=s(x);
          h:=s(h);
        end
      end
    end
  end
end
```

# Differenza Aritmetica

- ▶ Sottrarre il valore della variabile  $Y$  alla variabile  $X$

```
begin
  U:=0;
  while U ≠ Y do
    begin
      X:=pd(X);
      U:=s(U)
    end
  end
end
```



$X := X - Y$

Possiamo considerare l'operazione di *differenza aritmetica* indicata col simbolo  $\#$ , tra due numeri interi positivi  $X$  ed  $Y$ .

Avremo che  $X \# Y = X - Y$  se  $X$  è maggiore od uguale ad  $Y$

In caso contrario, ovvero quando  $Y$  è maggiore di  $X$  allora  $X \# Y = 0$ , ovvero poniamo il risultato della differenza propria uguale a zero. In questo modo **l'operazione è ben definita per ogni coppia di numeri interi positivi**

Dividere il valore della variabile X per  
la variabile Y,  $X:=X/Y$

---





# Macro: Quoziente e Resto della divisione tra A e B

```
Begin
resto:= 0;
Check:=A;
Q:=0;
While (A ≠ 0)
    begin
        A:= A – B;
        Q:= Q+1;
    end
Quoziente:=Q;
Prova = B * Quoziente;
While (Prova ≠ Check)
    begin
        Quoziente = Q – 1;
        Prova = Check;
    end
Prova2 := B * Quoziente
While (Prova2 ≠ Check)
    begin
        Resto = Check – Prova2;
        Prova2 = Check;
    end
End
```

# Operatori di confronto

---

- ▶ L'unico operatore di confronto disponibile è il  $\neq$  ma è spesso necessario considerare anche altri tipi di test
- ▶ Come possiamo rendere il **minore**, **il maggiore**, **l'uguale**?

# Minore

---

$$X < Y$$

$$(Y - X) - \text{pd}(Y - X)$$

$X < Y$  è vera se e soltanto se  $(Y - X) - \text{pd}(Y - X) = 1$

$X < Y$  è falsa se e soltanto se  $(Y - X) - \text{pd}(Y - X) = 0$



# Espressioni Booleane

---

- ▶ Possiamo costruire espressioni booleane “E” concatenando **and** ( $\wedge$ ), **not** ( $\neg$ ) e **or** ( $\vee$ )?
- ▶ L'intuizione è associare a ciascuna espressione una condizione di test che ha come unici valori ammissibile 0 e 1, che fanno riferimento rispettivamente al falso e al vero

# Espressioni

---

- ▶ Attraverso le macro possiamo quindi costruire gli usuali connettivi della logica elementare not, and e or
- ▶ (NOT)  $T_{\neg E} = 1 - T_E$  (dunque 1 se  $T_E = 0$  e viceversa)
- ▶ (AND)  $T_{E \wedge E'} = \text{pd}(T_E + T_{E'})$  (dunque 1 se  $T_E = T_{E'} = 1$ , 0 altrimenti)
- ▶ (OR)  $T_{E \vee E'} = T_E + T_{E'} - \text{pd}(T_E + T_{E'})$  (dunque 1 se  $T_E = 1$  o  $T_{E'} = 1$ , 0 altrimenti)

Si può provare anche la moltiplicazione per ottenere espressioni più semplici per  $\wedge$  e  $\vee$

# Estensione comando While

---

```
while EXPRBOOL do COMMAND
```

```
begin  
    U:=TEXPRBOOL  
    V:=0  
    while U ≠ V do begin COMMAND; U:=TEXPRBOOL end  
end
```

**EXPRBOOL** -- Generica Espressione Booleana

**T<sub>EXPRBOOL</sub>** -- Condizione di Test associata all'espressione booleana

# If ... then ... else

---

**if EXPRBOOL then COMMAND<sub>1</sub> else COMMAND<sub>2</sub>**

**begin**

**U:=T<sub>EXPRBOOL</sub>**

**V:=0**

**while (U = 1)  $\wedge$  (V = 0) do begin COMMAND<sub>1</sub>; V:=1 end**

**while (U = 0)  $\wedge$  (V = 0) do begin COMMAND<sub>2</sub>; V:=1 end**

**end**

# Concludendo!

---

- ▶ Per quanto riguarda la sintassi con tutte queste macro l'espressività del WHILE è sostanziale all'espressività del PASCAL

# La semantica del linguaggio WHILE

---

- ▶ Consideriamo programmi WHILE  $P$  a  $k$  variabili, cioè programmi abilitati a utilizzare  $k$  variabili  $X_1, \dots, X_k$ , o eventualmente un loro sottoinsieme
- ▶ Conveniamo poi che le variabili  $X_1, \dots, X_k$  siano ordinate in base ai loro indici  $i \in \{1, \dots, k\}$
- ▶ Chiamiamo *stato di computazione* di  $P$  una qualunque  $k$ -upla  $a = (a_1, \dots, a_k)$  di naturali: si intende allora che in  $P$   $X_1$  assume il valore  $a_1, \dots, X_k$  assume il valore  $a_k$ .
- ▶ Uno stato di computazione è anche detto *vettore di stato*

# Computazione

---

- ▶ Consideriamo di rappresentare la *computazione* di un programma WHILE  $P$  a  $k$  variabili
- ▶ Useremo le lettere  $C, C$  e così via per denotare termini della categoria sintattica COMMAND (cioè singoli comandi per  $P$ ) e  $C, C', \dots$  per indicare termini della categoria sintattica SEQCOMMAND (cioè sequenze di comandi per  $P$ )
- ▶ Si ricordi che  $P$  stesso può essere visto come una sequenza di comandi

# Configurazione

---

- ▶ Una coppia ordinata  $(C, a)$  dove  $C$  è una sequenza di comandi di  $P$  e  $a$  è un vettore di stato di  $P$  è chiamata *configurazione* di  $P$
- ▶ La sequenza  $C$  descrive, intuitivamente, quali comandi sono da svolgere a un certo punto della sua esecuzione mentre il vettore di stato  $a$  precisa il valore delle variabili a cui  $C$  si applica
- ▶ La *relazione di transizione*  $\rightarrow$  tra queste coppie  $(C, a)$  (sequenza di comandi, vettore di stato) mostra come vengono eseguiti i singoli comandi di un programma e come vengono modificate le variabili del programma stesso
- ▶ La terminazione di una sequenza di comandi viene denotata con `end`

# Semantica Operazionale – part I

---

$$SC_{zero} \frac{a = (a_1, \dots, a_k) \text{ e } a' = (a_1, \dots, a_{u-1}, 0, a_{u+1}, \dots, a_k)}{(Xu := 0, a) \rightarrow (\text{end}, a')}$$

$$SC_s \frac{a = (a_1, \dots, a_k) \text{ e } a' = (a_1, \dots, a_{u-1}, s(a_v), a_{u+1}, \dots, a_k)}{(Xu := s(Xv), a) \rightarrow (\text{end}, a')}$$

$$SC_{pd} \frac{a = (a_1, \dots, a_k) \text{ e } a' = (a_1, \dots, a_{u-1}, pd(a_v), a_{u+1}, \dots, a_k)}{(Xu := pd(Xv), a) \rightarrow (\text{end}, a')}$$

Le regole descrivono come vengono modificati i valori del vettore di stato quando un comando di assegnamento viene eseguito

# Semantica Operazionale – part II

Le regole gestiscono i comandi di interazione while

$$SC_{while1} \frac{a = (a_1, \dots, a_k) \text{ e } a_u \neq a_v}{(\text{while } Xu \neq Xv \text{ do } C, a) \rightarrow (C; \text{while } Xu \neq Xv \text{ do } C, a)}$$

Mostra come permettere l'esecuzione del corpo del while nel caso in cui il test è soddisfatto

$$SC_{while1} \frac{a = (a_1, \dots, a_k) \text{ e } a_u = a_v}{(\text{while } Xu \neq Xv \text{ do } C, a) \rightarrow (end, a)}$$

Mostra come permettere l'esecuzione del corpo del while nel caso in cui il test **non** è soddisfatto

# Semantica Operazionale – part III

---

$$SC_{program1} \frac{}{(\text{begin } end, a) \rightarrow (end, a)}$$

$$SC_{program2} \frac{}{(\text{begin } C \text{ end}, a) \rightarrow (C, a)}$$

Le regole gestiscono i comandi composti, e quindi si procede alla valutazione del loro corpo

# Semantica Operazionale – part IV

---

$$SC_{seq1} \frac{(C_1, a) \rightarrow (C'_1, a') \text{ e } C'_1 \neq end}{(C_1 ; C_2, a) \rightarrow (C'_1 ; C_2, a')}$$

$$SC_{seq2} \frac{(C_1, a) \rightarrow (end, a')}{(C_1 ; C_2, a) \rightarrow (C_2, a')}$$

Trattano sequenze di comandi e in particolar modo implementano la sequenzializzazione dei comandi specificati

# Computazione di un programma WHILE

---

Siano:

- P un programma WHILE a K variabili
- $a = (a_1, \dots, a_k)$  un vettore di stato

Una computazione di P è una sequenza, possibilmente infinita, della forma

$$\langle P, a \rangle \rightarrow \langle C_1, a' \rangle \rightarrow \langle C_2, a'' \rangle \rightarrow \langle C_3, a''' \rangle \rightarrow \dots$$

Dove per ogni  $i \geq 1$   $a^i$  è un vettore stato a k dimensioni e  $C_i$  una sequenza di comandi (che corrisponde alla situazione del programma P al passo i-esimo della computazione)

Se la sequenza di computazione è finita allora ha la forma

$$\langle P, a \rangle \rightarrow \langle C_1, a' \rangle \rightarrow \langle C_2, a'' \rangle \rightarrow \langle C_3, a''' \rangle \rightarrow \dots \rightarrow \langle \text{end}, a^n \rangle$$

La proposizione che segue è diretta conseguenza delle regole operazionali della relazione di transizione  $\rightarrow$  definita per tramite delle regole.

# Un programma divergente o convergente

---

Siano:

- P un programma WHILE a K variabili
- $a = (a_1, \dots, a_k)$  un vettore di stato

Allora esiste esattamente una computazione da P che ammette  $a$  come vettore di stato iniziale

In questo caso  $a$  prende il nome **di vettore di input** e si conviene quanto segue:

- Se la computazione P su  $a$  è **infinita**, si dice che P **diverge** sull'input  $a$  e che il suo output è indefinito
- Se invece la computazione è **finita** e della forma

$$\langle P, a \rangle \rightarrow \langle C_1, a' \rangle \rightarrow \langle C_2, a'' \rangle \rightarrow \langle C_3, a''' \rangle \rightarrow \dots \rightarrow \langle \text{end}, a^n \rangle$$

si dice che P **converge o termina** sull'input  $a$  e che il suo output è  $a^n$

# Funzione WHILE calcolabile

---

- ▶ Considerato che la computazione di un programma WHILE  $P$  su un dato input è unica possiamo parlare di **funzione calcolata** da  $P$
- ▶ Sia dunque  $P$  un programma a  $K$  variabili  $X_1, \dots, X_k$  per qualche  $K \geq 1$ ; si dice **funzione calcolata da  $P$**  la funzione parziale  $f_P$  da  $\mathbb{N}^k$  a  $\mathbb{N}$  tale che, per ogni  $a \in \mathbb{N}^k$ ,
  - ▶  $f_P(a)$  è il valore di  $X_1$  nell'output della computazione di  $P$  su  $a$  se questa computazione ha fine
  - ▶  $f_P(a)$  non è definita altrimenti
- ▶ **DEFINIZIONE:** Una funzione parziale  $f_P$  da  $\mathbb{N}^k$  a  $\mathbb{N}$  si dice **WHILE-calcolabile** se  $f_P$  è la funzione calcolata da qualche programma WHILE  $P$

# Programmi e MdT

---

- ▶ **DEFINIZIONE:** Una funzione parziale  $f_p$  da  $N^k$  a  $N$  si dice WHILE-calcolabile se e solo se è calcolabile (secondo Turing)