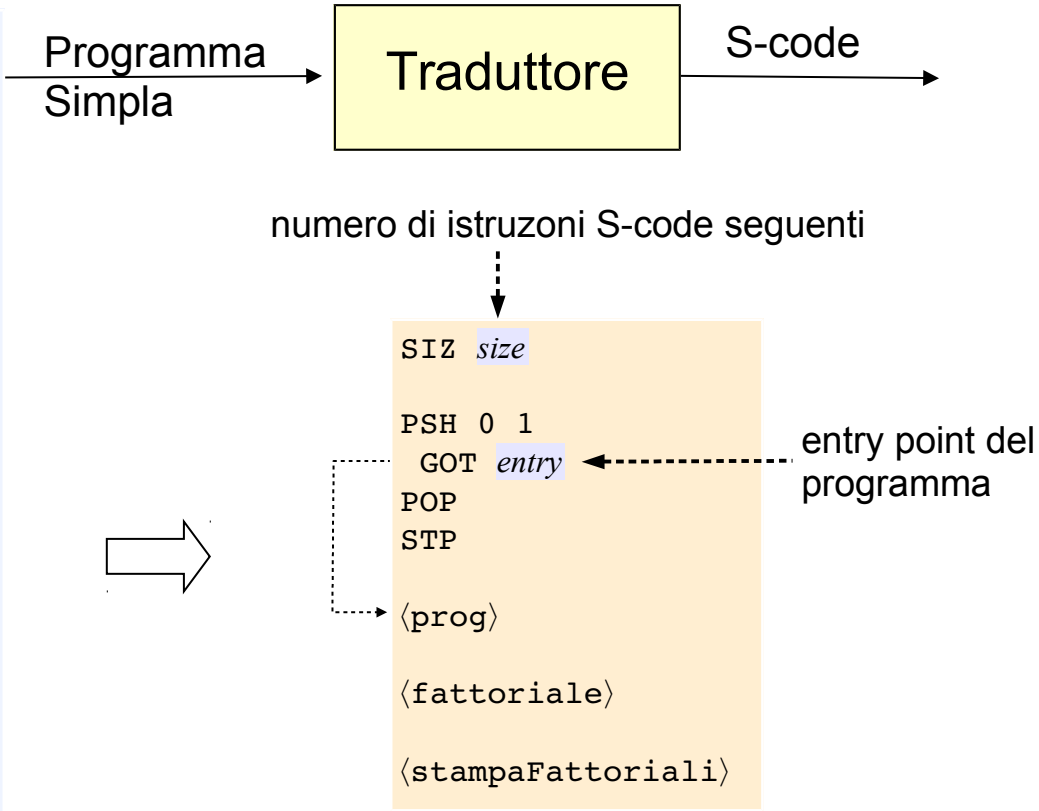
Generazione di Codice S-code

```
numero: integer;

func fattoriale(n: integer): integer
  fact: integer;
body
  if n == 0 then
    fact = 1;
  else
    fact = n * fattoriale(n-1);
  end;
  return fact;
end;

func stampaFattoriali(tot: integer): void
  i, f: integer;
body
  for i=0 to tot do
    f = fattoriale(i);
    writeln("Il fattoriale di ", i, "è ", f);
  end;
end;

body
  read(numero);
  if numero < 0 then
    writeln("Il numero ", numero, "non è valido");
  else
    stampaFattoriali(numero);
  end;
end.
```



`PSH` *num-formals num-variables*

num-formals = numero dei parametri formali
num-variables = numero delle varibili nell'ambiente

Generazione di Codice S-code (ii)

- **Scelte di progetto:**
  - Programma trattato come una funzione di tipo **`void`** senza parametri e senza **`return`**
  - Parametri formali della funzione trattati come variabili locali
  - Codice indirizzabile direttamente (senza uso di label)
  - Indirizzo di una istruzione S-code = posizione della istruzione nel codice (valore intero 0, 1, ...)
  - Descrittori degli oggetti (parametri e variabili) allocati nell'ordine in cui vengono dichiarati
  - Oggetti (parametri e variabili) trattati uniformemente dalla macchina astratta (S-machine)
  - Identificazione degli oggetti (variabili e parametri) mediante due interi: ***env oid,*** in cui ***env*** può essere 0 (variabile globale) o 1 (oggetto locale), mentre ***oid*** è l'object identifier dell'oggetto nel suo ambiente (locale o globale)

Dichiarazione di Variabili

```
i, j: integer;
x: real;
nome, cognome: string;
ok: boolean;
```

```
VAR |integer|
VAR |integer|
VAR |real|
VAR |string|
VAR |string|
VAR |integer|
```

- **Note:**
  - Dichiarazione di una variabile avente dimensione *type-size*: **VAR** *type-size*
  - *type-size:* dimensione del tipo della variabile nella macchina virtuale (interprete)
  - Stringa allocata nello heap dell'interprete e memorizzata come puntatore a carattere, quindi **|string|** = dimensione del puntatore a carattere
  - Per ragioni di efficienza relative alle copiature ed alle uguaglianze di stringhe, se una stringa è già allocata nello heap del'interprete, non può essere riallocata (la copiatura di stringhe si riduce alla copiatura di puntatori e l'uguaglianza di stringhe alla uguaglianza di puntatori)
  - Variabile di tipo **boolean**: trattata come **integer**

Referenza a Costante

1. Costante intera `y = 25;` ⇨ `LCI 25`
2. Costante reale `z = 3.14;` ⇨ `LCR 3.14`
3. Costante stringa `s = "alpha";` ⇨ `LCS "alpha"`
4. Costante booleana `b = true;` ⇨ `LCI 1`

- **Nota:**
  - Valori booleani: **`true`**, **`false`** → surrogati da interi: 1, 0

Referenza a Identificatore

1. Oggetto locale `x = y + 1;` ⇨ `LOD 1 ^y`

2. Oggetto globale `x = z + 1;` ⇨ `LOD 0 ^z`

- **Nota:**
  - Argomenti di **`LOD`** = *env* (ambiente: 0 = globale, 1 = locale) e *oid* (object identifier)
  - `^name` indica l'object identifier dell'oggetto di nome `name` (quindi, un intero)

Assegnamento

- Computazione della espressione di assegnamento + store (**`STO`**)

1. Assegnamento di un oggetto locale (variabile o parametro):

`x = expr;` ⇨

```
⟨expr⟩
STO 1 ^x
```

2. Assegnamento di una variabile globale:

`y = expr;` ⇨

```
⟨expr⟩
STO 0 ^y
```

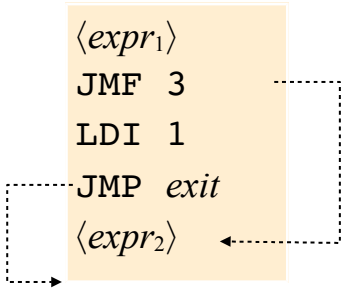
- **Nota:**
  - Argomento di **`STO`** = *env* (ambiente: 0 = globale, 1 = locale) e *oid* (object identifier)

Operazioni Logiche (`and`, `or`)

$logic\text{-}expr \rightarrow expr_1 \; expr_2$

and

or



locali

```
a, b, c: boolean;
...
a = (b and c) or g;
```

globale

```
LOD 1 ^b
JMF 3
LOD 1 ^c
JMP 2
LDI 0
JMF 3
LDI 1
JMP 2
LOD 0 ^g
STO 1 ^a
```

- **Note:**
  - Valutazione in corto circuito
  - **`JMP`** = salto incondizionato
  - **`JMF`** = salto condizionato (a false)
  - Argomento di **`JMP`**, **`JMF`** = lunghezza del salto (offset) $\begin{cases} exit = |\langle expr_2 \rangle| + 1 \\ offset = |\langle expr_2 \rangle| + 2 \end{cases}$

Operazioni Relazionali: ==, !=

rel-expr → $expr_1$ $expr_2$

==	!=
⟨$expr_1$⟩ ⟨$expr_2$⟩ EQU	⟨$expr_1$⟩ ⟨$expr_2$⟩ NEQ

locali

```
i, j: integer;
b: boolean;

b = (i == j or j != k);
```

globale

```
LOD 1 ^i
LOD 1 ^j
EQU
JMF 3
LDI 1
JMP 4
LOD 1 ^j
LOD 0 ^k
NEQ
STO 1 ^b
```

- **Nota:**
  - **EQU**, **NEQ**: polimorfi per tutti i tipi di oggetti

Operazioni Relazionali: >, >=, <, <=

- Operatori differenziati per i diversi tipi: 4 operatori * 3 tipi = 12 operatori

$rel\text{-}expr \rightarrow expr_1 \; expr_2$

	>	>=	<	<=
integer	$\langle expr_1 \rangle$ $\langle expr_2 \rangle$ GTI	$\langle expr_1 \rangle$ $\langle expr_2 \rangle$ GEI	$\langle expr_1 \rangle$ $\langle expr_2 \rangle$ LTI	$\langle expr_1 \rangle$ $\langle expr_2 \rangle$ LEI
real	$\langle expr_1 \rangle$ $\langle expr_2 \rangle$ GTR	$\langle expr_1 \rangle$ $\langle expr_2 \rangle$ GER	$\langle expr_1 \rangle$ $\langle expr_2 \rangle$ LTR	$\langle expr_1 \rangle$ $\langle expr_2 \rangle$ LER
string	$\langle expr_1 \rangle$ $\langle expr_2 \rangle$ GTS	$\langle expr_1 \rangle$ $\langle expr_2 \rangle$ GES	$\langle expr_1 \rangle$ $\langle expr_2 \rangle$ LTS	$\langle expr_1 \rangle$ $\langle expr_2 \rangle$ LES

```
i, j: integer;
x, y: real
b: boolean;

b = (i > j or x < y);
```

```
LOD 0 ^i
LOD 0 ^j
GTI
JMP 3
LDI 1
JMP 4
LOD 0 ^x
LOD 0 ^y
LTR
STO 0 ^b
```

Operazioni Aritmetiche: +, –, *, /

math -expr → $expr_1$ $expr_2$

	+	–	*	/
integer	⟨$expr_1$⟩ ⟨$expr_2$⟩ ADI	⟨$expr_1$⟩ ⟨$expr_2$⟩ SBI	⟨$expr_1$⟩ ⟨$expr_2$⟩ MUI	⟨$expr_1$⟩ ⟨$expr_2$⟩ DVI
real	⟨$expr_1$⟩ ⟨$expr_2$⟩ ADR	⟨$expr_1$⟩ ⟨$expr_2$⟩ SBR	⟨$expr_1$⟩ ⟨$expr_2$⟩ MUR	⟨$expr_1$⟩ ⟨$expr_2$⟩ DVR

```
i, j, k: integer;
x, y: real;

i = (i + 5) * (j – k);
x = (y - 3.14) / (x + y);
```

```
LOD 0 ^i
LCI 5
ADI
LOD 0 ^j
LOD 0 ^k
SBI
MUI
STO 0 ^i
LOD 0 ^y
LCR 3.14
SBR
LOD 0 ^x
LOD 0 ^y
ADR
DVR
STO 0 ^x
```

Operazioni di Negazione: `-`, `not`

	`- (integer)`	`- (real)`	`NOT`
neg-expr → *expr*	⟨*expr*⟩ `UMI`	⟨*expr*⟩ `UMR`	⟨*expr*⟩ `NEG`

```
i, j, k: integer;
a, b: boolean;

b = i > j * k and not (a or j == -k);
```

```
LOD 0 ^i
LOD 0 ^j
LOD 0 ^k
MUI
GTI
JMF 11
LOD 0 ^a
JMF 3
LDI 1
JMP 5
LOD 0 ^j
LOD 0 ^k
UMI
EQU
NEG
JMP 2
LDI 0
STO 0 ^b
```

Chiamata di Funzione

func-call → **id** $expr_1$ ... $expr_n$

⇨

$\langle expr_1 \rangle$

...

$\langle expr_n \rangle$

```
PSH num-formals num-variables
  GOT entry
POP
```

```
i, j, k: integer;

k = j * f(i+j, x, j-i);
```

globale

⇨

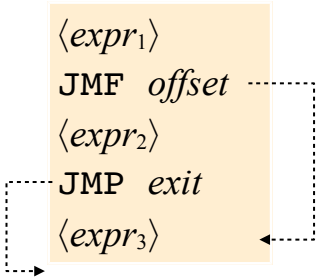
```
LOD 1 ^j
LOD 1 ^i
LOD 1 ^j
ADI
LOD 0 ^x
LOD 1 ^j
LOD 1 ^i
SBI
PSH 3 5
  GOT &f
POP
MUI
STO 1 ^k
```

entry point della funzione `f`

- **Note:**
  - *num-formals* = numero di parametri formali
  - *num-variables* = numero di variabili locali (non parametri)
  - *entry* = indirizzo della funzione chiamata (entry point nel corpo della funzione)

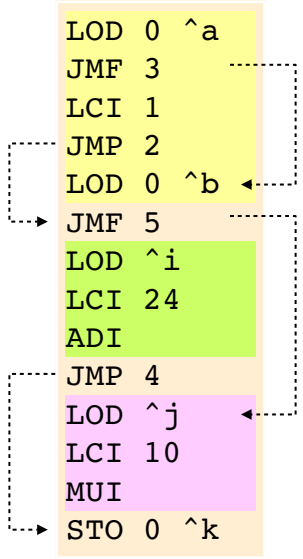
Espressione Condizionale

$cond\text{-}expr \rightarrow expr_1 \; expr_2 \; expr_3$




```
a, b: boolean;
i, j, k: integer;

k = if a or b then i + 24 else j * 10 end;
```



- **Note:**
  - $offset = |\langle expr_2 \rangle| + 2$
  - $exit = |\langle expr_3 \rangle| + 1$

Cast

casting → *expr*

⇨

integer

⟨*expr*⟩
TOI

real

⟨*expr*⟩
TOR

```
i, j: integer;
x: real;

x = real(i+j);
```

⇨

```
LOD 0 ^i
LOD 0 ^j
ADI
TOR
STO 0 ^x
```

Read

read-stat → **id**$_1$ ... **id**$_n$ ⇨

```
INP format  env oid1
 ...
INP format  env oidn
```

```
num: integer;
avg: real;
code: string;
flag: boolean;

read(num, avg, code, flag);
```

⇨

```
INP i 0 ^num
INP r 0 ^avg
INP s 0 ^code
INP b 0 ^flag
```

- **Note:**
  - *env* = ambiente (0 = globale, 1 = locale)
  - *oid* = object identifier dell'oggetto da istanziare
  - *format* = carattere (`i`, `r`, `s`, `b`) che indica il tipo di oggetto da istanziare (**`integer`**, **`real`**, **`string`**, **`boolean`**)

Write

$write\text{-}stat \rightarrow expr_1 \ldots \ expr_n$

⇨

$\langle expr_1 \rangle$
...
$\langle expr_n \rangle$
OUT n *format*

```
n, m: integer;
x, y: real;
code: string;

write(n+m, x-y, code);
```

⇨

```
LOD 0 ^n
LOD 0 ^m
ADI
LOD 0 ^x
LOD 0 ^y
SBR
LOD 0 ^code
OUT 3 "irs"
```

- **Note:**
  - n = numero di valori (sulla pila) da stampare
  - *format* = stringa dei caratteri che identificano i tipi dei valori delle espressioni da stampare

Istruzione Condizionale

if-stat → *expr stat-list*

⟨*expr*⟩
JMF *exit*
⟨*stat-list*⟩

if-stat → *expr* *stat-list*$_1$ *stat-list*$_2$

⟨*expr*⟩
JMF *offset*
⟨*stat-list*$_1$⟩
JMP *exit*
⟨*stat-list*$_2$⟩

```
a, b, c: integer;
if a == b then
    c = a + 1;
else
  if d > a then
    b = b - a;
  end;
end;
```

```
LOD 0 ^a
LOD 0 ^b
EQU
JMF 6
LOD 0 ^a
LCI 1
ADI
STO 0 ^c
JMP 9
LOD 0 ^d
LOD 0 ^a
GTI
JMF 5
LOD 0 ^b
LOD 0 ^a
SBI
STO 0 ^b
```

Ciclo While

while-stat → *expr stat-list*

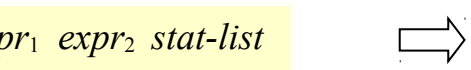
```
⟨expr⟩
JMF exit
⟨stat-list⟩
JMP up
```

```
a, b, c: integer;
c = 0;
while a >= b do
    c = c + 1;
    a = a – b;
    if a == 15 then
      break;
    end;
end;
```

```
LCI 0
STO 0 ^c
LOD 0 ^a
LOD 0 ^b
GEI
JMF 15
LOD 0 ^c
LDI 1
ADI
STO 0 ^c
LOD 0 ^a
LOD 0 ^b
SBI
STO 0 ^a
LOD 0 ^a
LCI 15
EQU
JMF 2
JMP 2
JMP -17
```

Ciclo For

for-stat → **id** $expr_1$ $expr_2$ *stat-list*



```
⟨expr1⟩
STO env ^id
⟨expr2⟩
STO env ^temp
LOD env ^id
LOD env ^temp
LEI
JMF exit
⟨stat-list⟩
INC env ^id
JMP up
```

```
i, tot, sum: integer;
...
for i=1 to tot do
  sum = sum + i;
end;
```

```
LCI 1
STO 0 ^i
LOD ^tot
STO 0 ^temp
LOD 0 ^i
LOD 0 ^temp
LEI
JMF 7
LOD 0 ^sum
LOD 0 ^i
ADI
STO 0 ^sum
INC 0 ^i
JMP -9
```

- **Note:**
  - `^temp`: object identifier di una variabile intera anonima (da aggiugere implicitamente nell'ambiente) che mantiene inalterato il valore di $expr_2$ nell'iterazione del ciclo
  - `INC` = incrementa di una unità il suo argomento intero (nello specifico, la variabile di conteggio)

Dichiarazione di Funzione con Valore di Ritorno

func-decl → **id** *opt-param-list* **type**
var-decl-list
stat-list

⇨

```
ENT ^id
⟨var-decl-list⟩
⟨stat-list⟩
RET 1
```

```
func fattoriale(n: integer): integer
  fact: integer;
body
  if n < 0 then
    return 0;
  end;
  if n == 0 then
    fact = 1;
  else
    fact = n * fattoriale(n-1);
  end;
  return fact;
end;
```

⇨

```
ENT ^fattoriale
VAR |integer|
LOD 1 ^n
LCI 0
LTI
JMF 3
LCI 0
JMP 18
LOD 1 ^n
LCI 0
EQU
JMF 4
LCI 1
STO 1 ^fact
JMP 10
LOD 1 ^n
LOD 1 ^n
LDC 1
SBI
PSH 1 1
  GOT &fattoriale
POP
MUI
STO 1 ^fact
LOD 1 ^fact
RET 1
```

- **Note:**
  - **^id** = object identifier della funzione
  - Se **`return`** non è l'istruzione finale della funzione → `JMP` a `RET` come traduzione della **`return`**
  - In generale, istruzione S-code: `RET` *n*, in cui *n* = numero di valori di ritorno (temporanei) sulla pila
  - Se alla esecuzione di `RET 1` non c'è un valore di ritorno (temporaneo) sulla pila: errore runtime

Dichiarazione di Funzione <u>senza</u> Valore di Ritorno

qualifica = `void`

func-decl → **id** *opt-param-list* **type**
var-decl-list
stat-list

⇨

```
ENT ^id
⟨var-decl-list⟩
⟨stat-list⟩
RET 0
```

```
func alfa(n: integer, m: integer): void
  r: integer;
body
  if n == 0 then
    return;
  end;
  m = m + 10;
  r = (n - m) / n;
end;
```

⇨

```
ENT ^alfa
VAR |integer|
LOD 1 ^n
LCI 0
EQU
JMF 2
JMP 11
LOD 1 ^m
LCI 10
ADI
STO 1 ^m
LOD 1 ^n
LOD 1 ^m
SBI
LOD 1 ^n
DVI
STO 1 ^r
RET 0
```

- **Note:**
  - Ultima istruzione: `RET 0`
  - Alla esecuzione di `RET 0`, impossibile avere un valore di ritorno (temporaneo) sulla pila (controllo semantico precedente: **`return`** senza argomento)