

# Generazione di Codice

- **Trasferimento di controllo nel linguaggio Intermedio:**
  - Generare codice con side effects: *emit*
  - Contatori di statements generati: *quad*
- **Backpatching**
  - Lista statements incompleti
  - Completamento: *backpatch*
- **Espressioni booleane:** short circuit

## Generiamo codice per effetto laterale

*emit*: emette il codice in un file di quadruple per codice

I comandi

Effetto laterale

```
[9] A::=ide:=E {emit(ide.loc ':='E.loc)}
```

Uso dell'attributo code

```
[9] A::=ide:=E {A.code:=E.code||gen(ide.loc ':='E.loc)}
```

## Le Espressioni

[11]  $E ::= F \{E'.in := F.loc\} E' \{E.loc := E'.loc\}$

[12]  $E'_1 ::= op-l F \{t := newtemp; emit(t := E'_1.loc \text{ 'op-l' } F.loc); E'_2.in := t\} E'_2 \{E'_1.loc := E'_2.loc\}$

...

[11]  $E ::= F \{E'.in := F.loc\} E' \{E.code := F.code \parallel E'.code; E.loc := E'.loc\}$

[12]  $E'_1 ::= op-l F \{t := newtemp; E'_2.in := t\}$

$E'_2 \{E'_1.code := F.code \parallel$

$gen(t := E'_1.loc \text{ 'op-l' } F.loc) \parallel$

$E'_2.code;$

$E'_1.loc := E'_2.loc\}$

...

## La sequenzializzazione di comandi

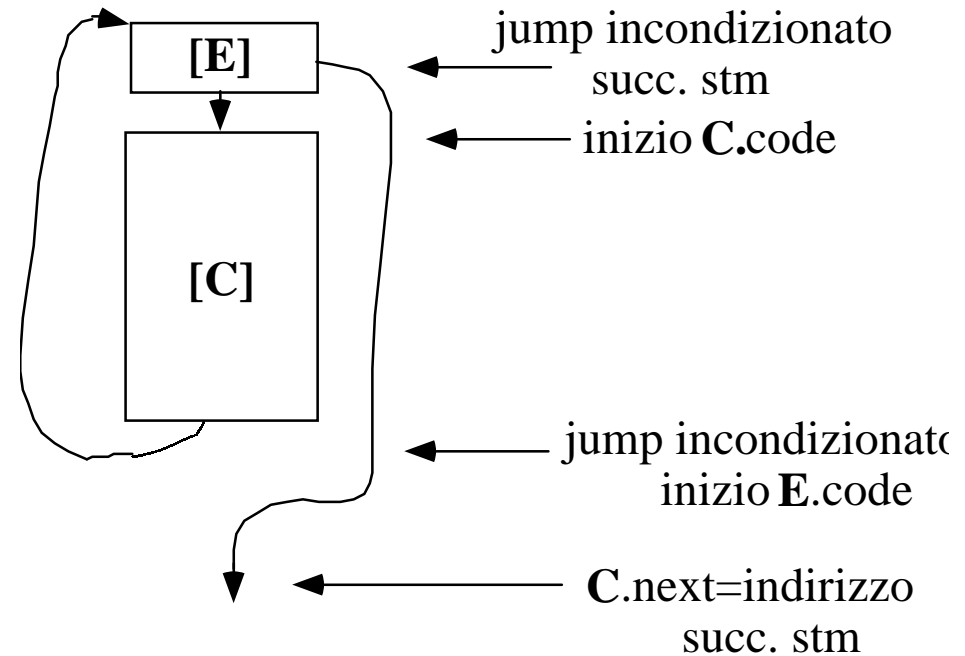
```
[5] Cs1 ::= ; C Cs2  
[6] Cs ::= ε  
...
```

```
[5] Cs1 ::= ; C Cs2 {Cs1.code := C.code || Cs2.code}  
[6] Cs ::= ε {Cs.code := empty }  
...
```

## Comandi che trasferiscono controllo

[10]W ::= while E do C endw

```
[E]
If E.loc=false goto ?
[C]
Goto ??
```



[10]W ::= while E do {emit('if' E.loc '=' 'false' goto --)} C {emit(goto --)} endw

## Usiamo un contatore quad di quadruple

- ricorda la posizione della quadrupla
- lo incrementiamo (automaticamente) ad ogni allocazione di quadrupla (ovvero: ad ogni **emit**)

```
[10]W ::= while {init:= quad} E do  
          {emit('if' E.loc '=' 'false' goto --)} C  
          {emit(goto init)} endw
```

## USIAMO BACKPATCHING

- Per ogni riferimento R ad una quadrupla la cui posizione non e' ancora nota (ovvero R e' non noto):
- Raccogliamo in una lista L(R) gli indirizzi di tutte le quadruple che richiedono tale R come operando ad esempio: goto R.

```
[10]W ::= while {init:= quad} E do  
           {w.next:=mk-L(quad);  
            emit('if' E.loc '=' 'false' goto --)} C  
           {emit(goto init)} endw
```

## Ogni comando e' un potenziale trasferimento di controllo

Ogni comando ha un'attributo: **next**

La traduzione precedente e' quindi errata allorché C  
generi un trasferimento di controllo:

```
[10]W ::= while {init:= quad} E do  
        {w.next:= mk-L(quad);  
         emit('if' E.loc '=' 'false' goto --)} C {BK(C.next, init);  
         emit(goto init)} endw
```



## I comandi (rivisti per l'attributo next)

```
[9] A := ide := E {emit(ide.loc := 'E.loc;  
A.next := emptylist}
```

## La sequenzializzazione di comandi (rivisitata per next)

```
[5] Cs1 ::= ; C {BK(C.next, quad)}  
      Cs2 {Cs1.next:= Cs2.next}  
[6] Cs ::= ε {Cs.next:= emptylist}  
...
```

# Espressioni booleane:

## Short Circuit

Generiamo codice per le espressioni booleane interpretate come trasferimento di controllo

**(x<y) or z**

produce

```
if x.loc<y.loc goto _{true}  
if z.loc goto _{true}  
goto _{false}
```

{da completare con  
indirizzo cui trasferire  
controllo per *true*}

## Usiamo effetti laterali: emit, nextquad

**.true** = lista incompleti che trasferiscono controllo a istruzione per *true*

**.true** = lista incompleti che trasferiscono controllo a istruzione per *false*

**Eb ::= Eb1 or M Eb2**

```
Eb.true ::= app(Eb1.true, Eb2.true),  
Eb.false ::= Eb2.false,  
BK(Eb1.false, M.quad)
```

**Eb ::= Eb1 and M Eb2**

```
Eb.true ::= Eb2.true,  
Eb.false ::= app(Eb1.false, Eb2.false),  
BK(Eb1.true, M.quad)
```

**Eb ::= true**

```
Eb.true ::= MK(quad, emptylist),  
Eb.false ::= emptylist,  
emit('goto' _)
```

**Eb ::= false**

```
Eb.true ::= emptylist,  
Eb.false ::= MK(quad, emptylist),  
emit('goto' _)
```

**Eb ::= ide**

```
Eb.true ::= MK(quad, emptylist),  
E.false ::= MK(quad+1, emptylist),  
emit('if' ide.loc '=' 'true' 'goto' _),  
emit('goto' _)
```

# Osservazioni:

Abbiamo usato grammatica per bottom-up

Produciamo una traduzione per top-down

$$Eb ::= Ea Eb'$$
$$Eb' ::= Op\_low Ea Eb'$$

Non abbiamo usato operatore *not*

Estendiamo la grammatica