Result Expression

P **result** e

Result Expression

P **result** e — execute P then evaluate e

Result Expression

P **result** e — execute P then evaluate e but no state change

Result Expression

P **result** *e* execute *P* then evaluate *e* but no state change

var *term*, *sum*: *rat* := 1·

for *i*:= 1;..15 **do** *term*:= *term*/*i*. *sum*:= *sum*+*term* **od**

result *sum*

Result Expression

P **result** e execute P then evaluate e but no state change

var $term, sum: rat := 1\cdot$

for $i:= 1;..15$ **do** $term:= term/i.\ \ sum:= sum+term$ **od**

result sum

axiom $P.\ \ (P$ **result** $e)=e$

Result Expression

P **result** *e* execute *P* then evaluate *e* but no state change

var *term*, *sum*: *rat* := 1·
for *i*:= 1;..15 **do** *term*:= *term*/*i*. *sum*:= *sum*+*term* **od**
result *sum*

axiom *P*. (*P* **result** *e*)=*e* but don't double-prime (*P* **result** *e*)
and don't substitute in (*P* **result** *e*)

Result Expression

P **result** e — execute P then evaluate e but no state change

var *term*, *sum*: *rat* := 1·
for i:= 1;..15 **do** *term*:= *term*/i. *sum*:= *sum*+*term* **od**
result *sum*

axiom — P. (P **result** e)=e — but don't double-prime (P **result** e)
and don't substitute in (P **result** e)

$\top$ — **result** axiom

= x:= x+1. (x:= x+1 **result** x)=x

Result Expression

P **result** e — execute P then evaluate e but no state change

var $term, sum$: rat := 1·

for i:= 1;..15 **do** $term$:= $term/i$. sum:= $sum+term$ **od**

result sum

axiom — P. (P **result** e)=e — but don't double-prime (P **result** e)

and don't substitute in (P **result** e)

	$\top$	**result** axiom
$=$	x:= x+1. (x:= x+1 **result** x)=x	Substitution Law but ...
$=$	(x:= x+1 **result** x) = x+1	

Result Expression

P **result** *e* execute *P* then evaluate *e* but no state change

var *term*, *sum*: *rat* := 1·

for *i*:= 1;..15 **do** *term*:= *term*/*i*. *sum*:= *sum*+*term* **od**

result *sum*

axiom *P*. (*P* **result** *e*)=*e* but don't double-prime (*P* **result** *e*)

and don't substitute in (*P* **result** *e*)

Result Expression

P **result** *e* execute *P* then evaluate *e* but no state change

var *term*, *sum*: *rat* := 1·
for *i*:= 1;..15 **do** *term*:= *term*/*i*. *sum*:= *sum*+*term* **od**
result *sum*

axiom *P*. (*P* **result** *e*)=*e* but don't double-prime (*P* **result** *e*)
don't substitute in (*P* **result** *e*)

	y:= (x:= x+1 **result** x)	assignment
=	$y' = (x{:=}\ x{+}1$ **result** $x) \land x'{=}x$	previous calculation
=	$y' = x{+}1 \land x'{=}x$	assignment
=	y:= x+1	

Result Expression

P **result** e execute P then evaluate e but no state change

implementation

Result Expression

P **result** e execute P then evaluate e but no state change

implementation

Replace each nonlocal variable within P and e that is assigned within P by a fresh local variable initialized to the value of the nonlocal variable. Then execute P and evaluate e.

Result Expression

P **result** e execute P then evaluate e but no state change

implementation

Replace each nonlocal variable within P and e that is assigned within P

by a fresh local variable initialized to the value of the nonlocal variable.

Then execute P and evaluate e .

but some language implementations don't introduce local variables

so expression evaluation can cause state change

Side Effects

Side Effects

$x = x$?

Side Effects

$x = x$ **?** not if there are side-effects **!**

Side Effects

$x = x$ **?** not if there are side-effects **!**

$x + x = 2 \times x$ **?**

Side Effects

$x = x$? — not if there are side-effects !

$x + x = 2 \times x$? — not if there are side-effects !

Side Effects

$x = x$ **?** not if there are side-effects **!**

$x + x = 2 \times x$ **?** not if there are side-effects **!**

for reasoning

$x := (P$ **result** $e)$ becomes $P.\ x := e$

Side Effects

$x = x$ **?** not if there are side-effects **!**

$x + x = 2{\times}x$ **?** not if there are side-effects **!**

for reasoning

$x:= (P$ **result** $e)$	becomes	$P.\ x:= e$
$x:= (P$ **result** $e) + y$	becomes	(**var** $z:= y\cdot\ P.\ x:= e{+}z$)

Side Effects

$x = x$ **?** not if there are side-effects **!**

$x + x = 2{\times}x$ **?** not if there are side-effects **!**

for reasoning

$x{:=}\ (P$ **result** $e)$ becomes $P.\ \ x{:=}\ e$

$x{:=}\ (P$ **result** $e) + y$ becomes (**var** $z{:=}\ y\cdot\ \ P.\ \ x{:=}\ e{+}z)$

↑

Side Effects

$x = x$ **?** not if there are side-effects **!**

$x + x = 2 \times x$ **?** not if there are side-effects **!**

for reasoning

$x := (P$ **result** $e)$	becomes	$P.\ x := e$
$x := (P$ **result** $e) + y$	becomes	$($**var** $z := y \cdot\ P.\ x := e + z)$
↑		↑

Side Effects

$x = x$ **?** not if there are side-effects **!**

$x + x = 2{\times}x$ **?** not if there are side-effects **!**

for reasoning

$x{:=}\ (P$ **result** $e)$ becomes $P.\ \ x{:=}\ e$

$x{:=}\ (P$ **result** $e) + y$ becomes $($**var** $z{:=}\ y\cdot\ \ P.\ \ x{:=}\ e{+}z)$

↑ ↑ ↑

Side Effects

$x = x$ **?** not if there are side-effects **!**

$x + x = 2{\times}x$ **?** not if there are side-effects **!**

for reasoning

$x{:=}\ (P\ \mathbf{result}\ e)$	becomes	$P.\ \ x{:=}\ e$
$x{:=}\ (P\ \mathbf{result}\ e) + y$	becomes	$(\mathbf{var}\ z{:=}\ y\cdot\ \ P.\ \ x{:=}\ e{+}z)$

Don't neglect the time for expression evaluation.

Function

```
int bexp (int n)
{  int r = 1;
   int i;
   for (i=0; i<n; i++) r = r*2;
   return r; }
```

Function

↓

```
int bexp (int n)
{  int r = 1;
   int i;
   for (i=0; i<n; i++) r = r*2;
   return r; }
```

C function = assertion about the result

Function

↓

```
int bexp (int n)
{  int r = 1;
   int i;
   for (i=0; i<n; i++) r = r*2;
   return r; }
```

C function = assertion about the result
+ name

Function

↓

```
int bexp (int n)
{  int r = 1;
   int i;
   for (i=0; i<n; i++) r = r*2;
   return r; }
```

C function = assertion about the result
+ name
+ parameters

Function

```
int bexp (int n)
{  int r = 1;
   int i;
   for (i=0; i<n; i++) r = r*2;
   return r; }
```

C function = assertion about the result
+ name
+ parameters
+ scope control

Function

```
int bexp (int n)
{  int r = 1;
   int i;
   for (i=0; i<n; i++) r = r*2;
→  return r; }
```

C function = assertion about the result

\+ name

\+ parameters

\+ scope control

\+ result expression

Function

```
int bexp (int n)
{  int r = 1;
   int i;
   for (i=0; i<n; i++) r = r*2;
   return r; }
```

$bexp \; = \; \langle \; n{:}\; int \cdot$

$\quad$ **var** $r{:}\; int := 1 \cdot$

$\quad$ **for** $i{:}= 0;..n$ **do** $r{:}= r{\times}2$ **od**.

$\quad$ **assert** $r{:}\; int$

$\quad$ **result** $r \; \rangle$

C function = assertion about the result
+ name
+ parameters
+ scope control
+ result expression

Function

```
int bexp (int n)
{  int r = 1;
   int i;
   for (i=0; i<n; i++) r = r*2;
   return r; }
```

⟨ *n*: *int*·
var *r*: *int* := 1·
for *i*:= 0;..*n* **do** *r*:= *r*×2 **od**.
assert *r*: *int*
result *r* ⟩

C function	=	assertion about the result
	+	name
	+	parameters
	+	scope control
	+	result expression

Function

```
int bexp (int n)
{  int r = 1;
   int i;
   for (i=0; i<n; i++) r = r*2;
   return r; }
```

$bexp = \langle\ n{:}\ int\cdot$

$\quad$ **var** $r{:}\ int := 1\cdot$

$\quad$ **for** $i{:}= 0;..n$ **do** $r{:}= r{\times}2$ **od**.

$\quad$ **assert** $r{:}\ int$

$\quad$ **result** $r\ \rangle$

C function = assertion about the result

\+ name

\+ parameters

\+ scope control

\+ result expression

Function

```
int bexp (int n)
{  int r = 1;
   int i;
   for (i=0; i<n; i++) r = r*2;
   return r; }
```

$bexp = \langle\, n{:}\, int \cdot$
$\quad$ **var** $r{:}\, int := 1 \cdot$
$\quad$ **for** $i{:}= 0;..n$ **do** $r{:}= r\times 2$ **od**.
$\quad$ **result** $r \,\rangle$

C function = assertion about the result
+ name
+ parameters
+ scope control
+ result expression

Function

```
int bexp (int n)
{  int r = 1;
   int i;
   for (i=0; i<n; i++) r = r*2;
   return r; }
```

$bexp \;=\; \langle\; n\!: int\cdot$
var $r\!: int := 1\cdot$
for $i\!:= 0;..n$ **do** $r\!:= r\times 2$ **od**.
assert $r\!: int$
result $r\;\rangle$

C function = assertion about the result
+ name
+ parameters
+ scope control
+ result expression

Procedure

procedure	=		name of procedure
		+	parameters
		+	scope control

Procedure

procedure	=		name of procedure
		+	parameters
		+	scope control

$$P \;=\; \langle x\colon int\cdot\ a' < x < b' \rangle$$

Procedure

procedure	=	name of procedure
	+	parameters
	+	scope control

$$P \;=\; \langle x\colon int\cdot\ a' < x < b' \rangle$$

$$P\,3 \;=\; a' < 3 < b'$$

Procedure

procedure	=	name of procedure
	+	parameters
	+	scope control

$$P \;=\; \langle x\text{: } int\cdot\ a' < x < b' \rangle$$

$$P\ 3 \;=\; a' < 3 < b'$$

$$P\ (a{+}1) \;=\; a' < a{+}1 < b'$$

Procedure

procedure = name of procedure
+ parameters
+ scope control

$$P \;=\; \langle x\colon int\cdot\ a' < x < b' \rangle$$

$$P\,3 \;=\; a' < 3 < b'$$

$$P\,(a{+}1) \;=\; a' < a{+}1 < b'$$

$$a' < x < b' \;\Leftarrow\; a:= x{-}1.\;\; b:= x{+}1$$

Procedure

procedure = name of procedure
+ parameters
+ scope control

$P \;=\; \langle x\text{: } int\cdot\ a' < x < b' \rangle$

$P\,3 \;=\; a' < 3 < b'$

$P\,(a+1) \;=\; a' < a+1 < b'$

$a' < x < b' \;\Leftarrow\; a{:=}\, x-1\,.\;\; b{:=}\, x+1$

$\langle p\text{: } D\cdot\ B\rangle\, a \;=\; (\textbf{var}\ p\text{: } D := a\cdot\ \ B)$ if B doesn't use p' or $p{:=}$

Procedure

reference parameter **var** parameter

Procedure

reference parameter **var** parameter

$\langle \mathbf{var}\ x: int\cdot\ \ a{:=}\ 3.\ \ b{:=}\ 4.\ \ x{:=}\ 5\rangle$

Procedure

reference parameter **var** parameter

$\langle \mathbf{var}\ x{:}\ int\cdot\ \ a{:}{=}\ 3.\ \ b{:}{=}\ 4.\ \ x{:}{=}\ 5\rangle\ a$

Procedure

reference parameter **var** parameter

$$\langle \textbf{var}\ x{:}\ int\cdot\ \ a{:}= 3.\ \ b{:}= 4.\ \ x{:}= 5\rangle\ a$$

$$=\quad a{:}= 3.\ \ b{:}= 4.\ \ a{:}= 5$$

Procedure

reference parameter **var** parameter

$\langle \mathbf{var}\ x\colon int \cdot\ a:= 3.\ b:= 4.\ x:= 5\rangle\ a$

$=\quad a:= 3.\ b:= 4.\ a:= 5$

$=\quad a'=5 \wedge b'=4$

Procedure

reference parameter **var** parameter

$$\langle \textbf{var}\ x: int\cdot\ \ a:=3.\ \ b:=4.\ \ x:=5\rangle\ a$$

$$=\quad a:=3.\ \ b:=4.\ \ a:=5$$

$$=\quad a'=5 \land b'=4$$

$$\langle \textbf{var}\ x: int\cdot\ a'=3 \land b'=4 \land x'=5\rangle\ a \quad = \quad ?$$

Procedure

reference parameter **var** parameter

$\langle \mathbf{var}\ x{:}\ int\cdot\ \ a{:=}\ 3.\ \ b{:=}\ 4.\ \ x{:=}\ 5\rangle\ a$

$= \quad a{:=}\ 3.\ \ b{:=}\ 4.\ \ a{:=}\ 5$

$= \quad a'{=}5 \wedge b'{=}4$

$\langle \mathbf{var}\ x{:}\ int\cdot\ \ x{:=}\ 5.\ \ b{:=}\ 4.\ \ a{:=}\ 3\rangle\ a$

$= \quad a{:=}\ 5.\ \ b{:=}\ 4.\ \ a{:=}\ 3$

$= \quad a'{=}3 \wedge b'{=}4$

$\langle \mathbf{var}\ x{:}\ int\cdot\ a'{=}3 \wedge b'{=}4 \wedge x'{=}5\rangle\ a \quad = \quad$ ***?***

Procedure

reference parameter **var** parameter

$$\langle \mathbf{var}\ x: int\cdot\ \ a:=3.\ \ b:=4.\ \ x:=5\rangle\ a$$
$$=\ \ a:=3.\ \ b:=4.\ \ a:=5$$
$$=\ \ a'=5 \land b'=4$$

$$\langle \mathbf{var}\ x: int\cdot\ \ x:=5.\ \ b:=4.\ \ a:=3\rangle\ a$$
$$=\ \ a:=5.\ \ b:=4.\ \ a:=3$$
$$=\ \ a'=3 \land b'=4$$

$$\langle \mathbf{var}\ x: int\cdot\ a'=3 \land b'=4 \land x'=5\rangle\ a \quad = \quad ?$$

warning Use only for programs, not for arbitrary specifications.

Do not manipulate the procedure body.

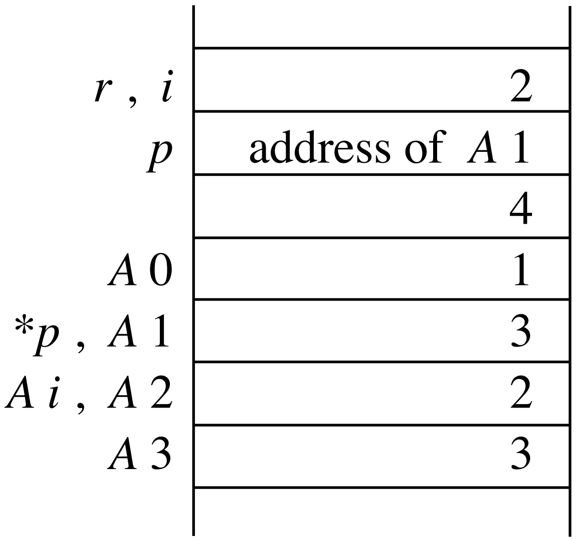
Substitute arguments for parameters before any other manipulations.

Apply programming theory separately for each call.

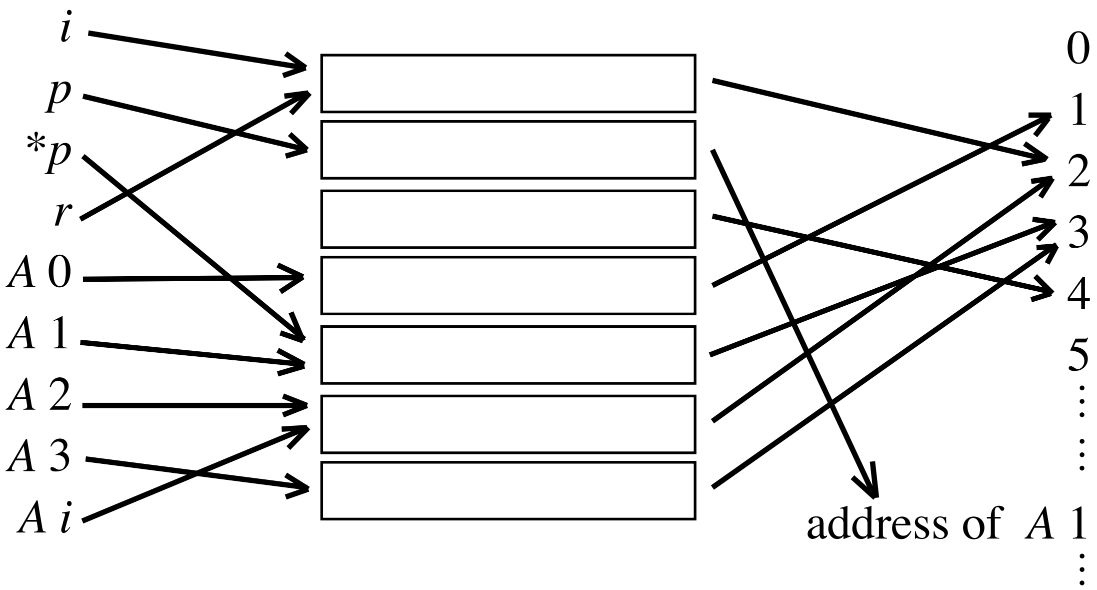
Alias

r , i	2
p	address of A 1
	4
A 0	1
$*p$, A 1	3
A i , A 2	2
A 3	3

Alias

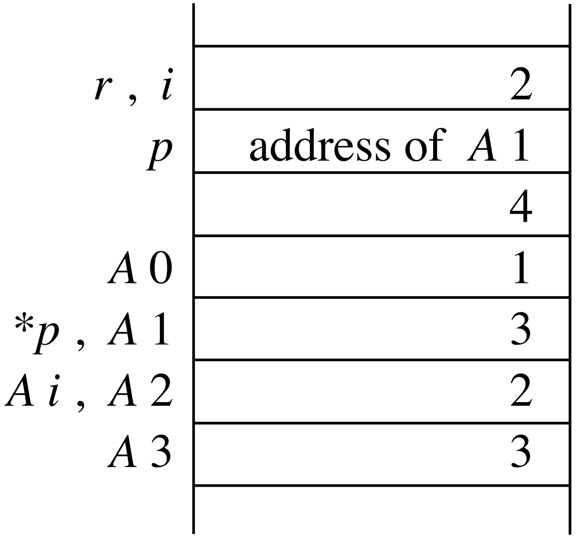


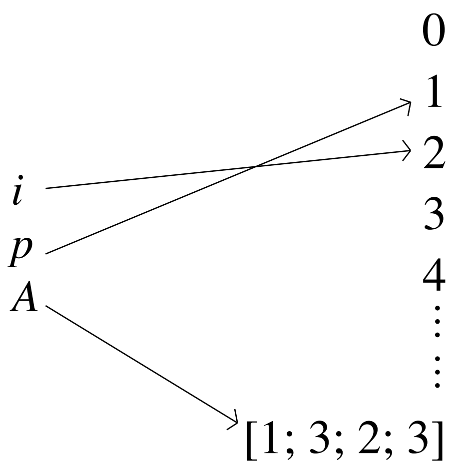
r , i
2
p
address of A 1
4
A 0
1
*p , A 1
3
A i , A 2
2
A 3
3

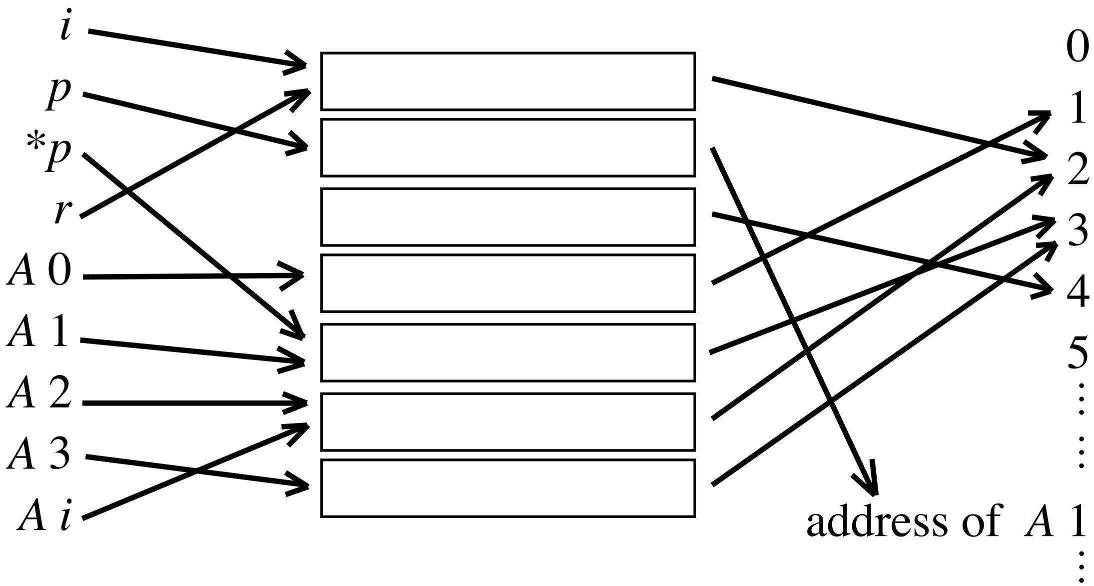


i
p
*p
r
A 0
A 1
A 2
A 3
A i
0
1
2
3
4
5
address of A 1

Alias


r , i
2
p
address of A 1
4
A 0
1
*p , A 1
3
A i , A 2
2
A 3
3


0
1
2
i
3
p
4
A
[1; 3; 2; 3]


i
p
*p
r
A 0
A 1
A 2
A 3
A i
0
1
2
3
4
5
address of A 1