Values

DENOTABLE

STORABLE

EXPRESSIBLE

COMPUTABLE

MUTABLE

IMMUTABLE

Code

```
CONST INT X = 3
INT X = 3
```

- 3 (in CONST INT X = 3) → AN IMMUTABLE INTEGER 3 for name X
- 3 (in INT X = 3) → A MUTABLE REFERENCE with AN INITIAL VALUE 3 for name X
- 3 IS ALSO A STORABLE VALUE here
- In both CASES, 3 IS USED AS AN EXPRESSIBLE VALUE

```
x = x + 2
x = 3 + 2
```

x + 2 and 3 + 2 → Two expressions which show that 5 is a computable value

* This concludes that integers in C are: DENOTABLE, STORABLE, EXPRESSIBLE, IMMUTABLE. Moreover, references of integers ARE AVAILABLE AS MUTABLE VALUE

* What about:
- \+ array e struct in C ?
- \+ Vector in JAVA ?
- \+ function in OCAML ?

\+ ARRAY IN C

```
CONST INT X[] = {2,3}
INT X[2,2] = {{2,3},{4,5}}
```

denotable, storable, IMMUTABLE (partially) expressible, MUTABLE

\+ STRUCT IN C, IS IN ADDITION, COMPUTABLE but not EXPRESSIBLE

```
typedef struct{int a; char b;} pair;
pair new(int a, char b){
        pair c;
        c.a = a; c.b = b
        return c;};
...
new(3,'b')
```

new(3,'b') ← is computing a struct value

```
/*
 Exercise Example 2 part (a) of slide 17 of Lecture3-6.
 The declaration is the beginning of a new block.
 Morover, a rephrasing in C shows what the code is computing.
 */


#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]){
    int z = 4;
    while(z > 0){
        --z;
        {
        int z;
        printf("value of the second occurrence of z=%2d\n", z);
        }
    }
    printf("final value of z in the outer block, z=%2d\n", z);
    return 0;
}
```

```
/*
 Exercise Example 2 part (b) of slide 17 of Lecture3-6.
 The declaration is moved to the beginning of the block.
 Morover, a rephrasing in C (as required by Example 2 on the right side)
 shows what the code is computing.
 */


#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]){
    int z = 4;
    while(z > 0){
        int z;
        --z;
        printf("value of the second occurrence of z=%2d\n", z);
    }
    printf("final value of z in the outer block, z=%2d\n", z);
    return 0;
}
```

```
/*
 Exercise Example of slide 28 of Lecture3-6-2014.
 When &A and &B are the same we have an aliasing and XSwap computes
 inappropriately.
*/

#include <stdio.h>
#include <stdlib.h>

void XSwap(int A[], int B[]){
    A[0] ^= B[0];
    B[0] ^= A[0];
    A[0] ^= B[0];
    };
int main(void){
    int A[] = {5};
    int B[] = {3};
    int C[] = {3};
    printf("A=%d,B=%d,C=%d\n",A[0],B[0],C[0]);
    XSwap(A,B);
    printf("A=%d,B=%d; ",A[0],B[0]);
    XSwap(A,C);
    printf("A=%d,C=%d\n",A[0],C[0]);
    XSwap(A,A);
    printf("A=%d\n",A[0]);
    return 0;
    }
```

```
(*
 Exercise Example of slide 31 of Lecture3-6-2014.
 Rephrase the code in OCaml and in addition, provide a mutable 'printer' in order
 to collect the sequence of the printed valued.
 *)

(* +++++++++++ Static Scope +++++++++++++++++ *)
let printer = ref [] in
let print = function n -> printer := n::!printer in
let x = ref 0 in
let pippo xr = function n -> xr := !xr + n in
let pippo1 = pippo(x) in
pippo1(3);
print(!x);
(let x = ref 0 in
 pippo1(3);
 print(!x));
print(!x);
List.rev(!printer);;

(* ========== Dynamic ============= *)

let printer = ref [] in
let print = function n -> printer := n::!printer in
let x = ref 0 in
let pippo xr = function n -> xr := !xr + n in
pippo(x)(3);
print(!x);
(let x = ref 0 in
 pippo(x)(3);
 print(!x));
print(!x);
List.rev(!printer);;

(* ++++++++++++++++++++++++++++++++++++ *)
```

```
(*
 Exercise Example of slide 32 of Lecture3-6-2014.
 Rephrase in OCam, the JavaScript code in the slide and insert an additional mu-
 table printer for collecting the sequence of the printed valued.
*)

let newloc init =
        let x = ref init in
        let mem(code,value) =
            if (code = "add") then (x := !x + value; x)
            else if (code = "reset") then (x := value; x) else x
            in mem;;
let aloc = newloc(5);;
aloc("inc",2);;
aloc("reset",12);;



(* alternatively,

type codes = Add | Reset;;
let newloc init =
        let x = ref init in
        let mem(code,value) = match code with
                Add -> (x := !x + value; x)
                |Reset -> (x := value; x)
        in mem;;
let aloc = newloc(5);;
aloc(Add,2);;
aloc(Reset,12);;

*)
```

4.6 Exercises

Exercises 6–13, while really being centred on issues relating to scope, presuppose knowledge of parameter passing which we will discuss in Chap. 7.

1. Consider the following program fragment written in a pseudo-language which uses static scope and where the primitive `read(Y)` allows the reading of the variable `Y` from standard input.

```
  ...
  int X = 0;
  int Y;
void fie(){
    X++;
}
void foo(){
    X++;
    fie();
}
read(Y);
if Y > 0{int X = 5;
         foo();}
else foo();
write(X);
```

   State what the printed values are.

2. Consider the following program fragment written in a pseudo-language that uses dynamic scope.

```
 ...
int X;
X = 1;
int Y;
void fie() {
   foo();
   X = 0;
}
void foo(){
   int X;
   X = 5;
}
read(Y);
if Y > 0{int X;
         X = 4;
         fie();}
else     fie();
write(X);
```

   State which is (or are) the printed values.

3. Consider the following code fragment in which there are gaps indicated by (*) and (**). Provide code to insert in these positions in such a way that:

a. If the language being used employs static scope, the two calls to the procedure `foo` assign the same value to `x`.
b. If the language being used employs dynamic scope, the two calls to the procedure `foo` assign different values to `x`.

The function `foo` must be appropriately declared at (*).

```
{int i;
(*)
for (i=0; i<=1; i++){
   int x;
   (**)
   x= foo();
   }
}
```

4. Provide an example of a denotable ~~object~~ VALUE whose life is longer than that of the references (names, pointers, etc.) to it.
5. Provide an example of a ~~connection between a name and a denotable object~~ BINDING whose life is longer than that of the object itself → DENOTABLE VALUE
6. Say what will be printed by the following code fragment written in a pseudo-language which uses static scope; the parameters are passed by a value.

```
{int x = 2;
 int fie(int y){
     x = x + y;
  }

  {int x = 5;
   fie(x);
   write(x);
  }
 write(x);
}
```

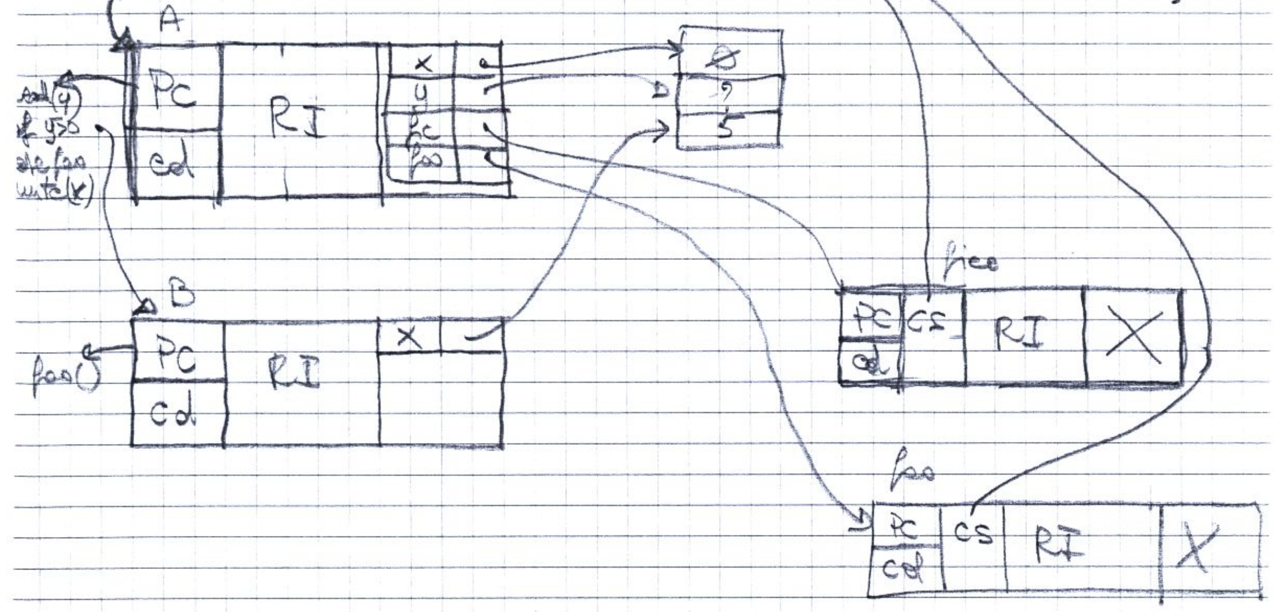
7. Say what is printed by the code in the previous exercise if it uses dynamic scope and call by reference.
8. State what is printed by the following fragment of code written in a pseudo-language which uses static scope and passes parameters by reference.

```
{int x = 2;
 void fie(reference int y){
     x = x + y;
     y = y + 1;
 }
 {int x = 5;
  int y = 5;
  fie(x);
  write(x);
 }
 write(x);
}
```

7bis. Check Your Answers to 6 and to 7, by using a rephrasing in C, of the 'pseudo' code and lambda lifting when needed.

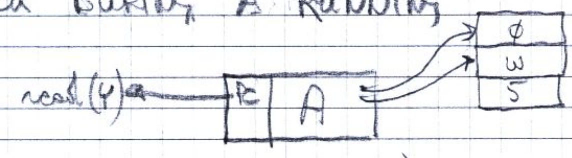
Ex 4.6.1

① ACTIVATION RECORDS (ALL of THEM ARE STATICS)

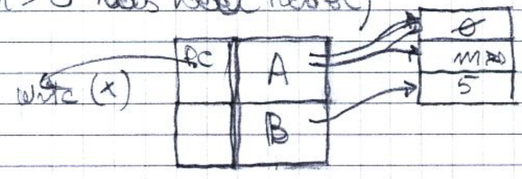


②- ARs STACK DURING A RUNNING

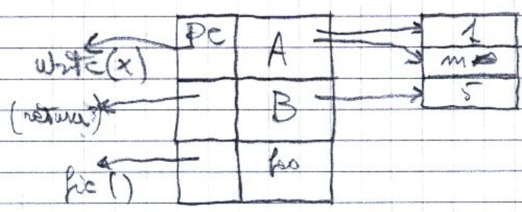
step 1



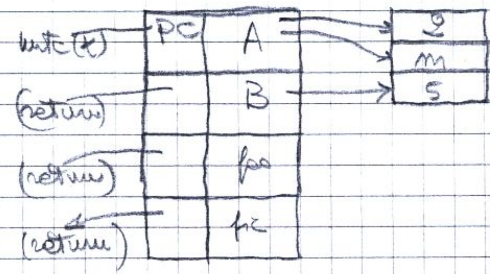
step 2.1.1 (when m > 0 has been read)



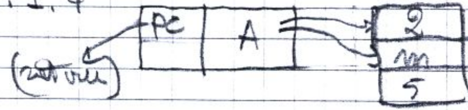
step 2.1.2



step 2.1.3



step 2.1.4



output

2

(3) - what value is printed when $m \le 0$?

2 (again)

(4) Apply le Blank-Cook and show the resulting code

```
...
int x = 0;
int y;
void fie(){
    [1,0]++;
}
void foo(){
    [1,0]++;
    [1,2]();
}
read([0,1]);
if [0,1]>0 { int x=5;
             [1,3](); }
else [0,3]();
write([0,0]);
```

EX 4.6.2

① SHOW THE PRINTED VALUES, IF ANY

- when m > 0 has been read
  the value is 1
- when m ≤ 0 has been read
  the value is ∅.

② SHOW HOW THE FRAGMENT MUST BE REPHRASED BY THE USE OF LAMBDA LIFTING WHEN THE LANGUAGE USES STATIC SCOPE

```
int x
x = 1
void fie (by reference x) {
    foo ();
    x = 0;
}
void foo () {
    int x;
    x = 5;
}
if Y > 0 { int x;
           x = 4
           fie (x);
         }
else fie (x);
write (X)
```

Ex 4.6.3

(a) & b) * ≡ i = 5;
int foo() { return i; }

** ≡ empty code or any code that is unaffecting i and k.

Ex 4.6.4

```
...
int x;
void fie (by reference z){
    z = 7
}
fie(x);
...
```

the denotable value of z survives to the binding of z.

```
...
int w = 3;
int *x [] = {&w};
void fie (int*z[]){
    int y = 5;
    z[0] = &y;
};
fie(x);
print (*x[0]);
...
```

the storable value of *X[0] is the denotable value of y

Ex 4.6.5

what does it happen when fie before is replaced by

```
void fie (int *z[]){
    int y = 5
    z[0] = &y;
    free(&y);
}
```

```
/*
 Exercise 4 page 88 from the book by Gabrielli-Martini.
 The denotable value of z is a int-pointer &x which survives to the
 binding (z,&x).
 Noting that &x will be *5 (i.e. a mutable with 5) and free(z) will
 deallocate such a mutable, and then it will create a dangling-refen
 rence for x.
 */

#include <stdio.h>
#include <stdlib.h>

void fie(int *z){
    *z = 7;
/*  free(z); */
    };
int main(void){
    int x;
    fie(&x);
    printf("x=%d\n",x);
    return 0;
    }
```

```
/*
 Exercise 5 page 88 from the book by Gabrielli-Martini.
 The binding (w,*3) is created in the body of main and the denotable
 value *3 is used for setting x[0]. What does it happen when *3 is
 deallocated before such an assignment.
 */

#include <stdio.h>
#include <stdlib.h>

void fie(int *z[]){
    int y = 5;
    z[0] = &y;
/*  free(&y); */
    };

int main(void){
    int w=3;
    int *x[1]={&w};
    fie(x);
    printf("x=%d\n",*x[0]);
    return 0;
}

/* replace the main with:
int main(void){
    int w=3;
    int *x[1];
//  free(&w);
    x[0]=&w;
    printf("x=%d\n",*x[0]);
    return 0;
}

*/
```

```c
/*
 Exercise 6 page 88 from the book by Gabrielli-Martini.
 Use of code rephrasing
*/

#include <stdio.h>
#include <stdlib.h>

int x;

int fie(int y){
    x = x+y;
}

int main (int argc, char *argv[]){
    x = 2;
     {
       int x = 5;
       fie(x);
       printf("value of the second occurrence of x=%2d\n", x);
     };
    printf("value of the first occurrence of x=%2d\n", x);
    return 0;
}
```

```c
/*
 Exercise 7 page 88 from the book by Gabrielli-Martini.
 Use of code rephrasing and of Lambda-lifting (due to dynamic scope)
*/

#include <stdio.h>
#include <stdlib.h>

int x;

int fie(int *x, int *y){
    *x = *x + *y;
    *y = *y + 1;
}

int main (int argc, char *argv[]){
    x = 2;
     {
       int x = 5;
       int y = 5;
       fie(&x, &x);
       printf("value of the second occurrence of x=%2d\n", x);
     };
    printf("value of the first occurrence of x=%2d\n", x);
    return 0;
}
```