CSE 3302
Programming Languages

Control II
Procedures and Environments (cont.)

Chengkai Li
Fall 2007

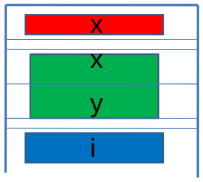
Activation Record for Procedures

```
int x; //global
void B(void) {
  int i;
  i = x/2;
}
void A(void) {
  int x,y;
  x = y*10;
  B();
}
main() {
  A();
  return 0;
}
```

i: local variable in called environment

x: global variable in defining environment

x,y: local variable in calling environment

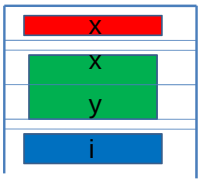


Activation Record for Procedures

```
int x; //global
void B(void) {
  int i;
  i = x/2;
}
void A(void) {
  int x,y;
  x = y*10;
  B();
}
main() {
  A();
  return 0;
}
```

Can only access global variables in defining environment

No direct access to the local variables in the calling environment
(Need to communicate through parameters)



Procedure Call

- Caller:

```
…
f(i);
…
```

Callee:

```
int f(int a){
  ...;
  ...a...;
  ...;
}
```

actual parameter / argument

formal parameter / argument

Parameter Passing Mechanisms:

- When and how to evaluate parameters
- How actual parameter values are passed to formal parameters
- How formal parameter values are passed back to actual parameters

Parameter Passing Mechanisms

- Pass/Call by Value
- Pass/Call by Reference
- Pass/Call by Value-Result
- Pass/Call by Name

Example

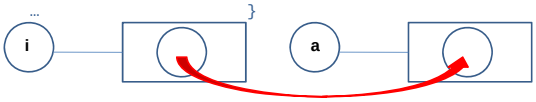
- What is the result?

```
void swap(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}
main(){
    int i=1, j=2;
    swap(i,j);
    printf("i=%d, j=%d\n", i, j);
}
```

- It depends…

Pass by Value

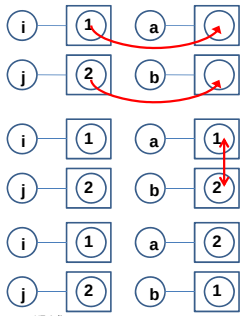
- Caller:
  ```
  …
  f(i);
  …
  ```
- Callee:
  ```
  int f(int a){
     ...a...;
  }
  ```



- Most common one
- Replace formal parameters by the values of actual parameters
- Actual parameters: No change
- Formal parameters:
  - Constant (Ada, "in")
  - Local variables (C, C++, Java, Pascal)

Example: Pass By Value

```
void swap(int a, int b) {
   int temp;
   temp = a;
   a = b;
   b = temp;
}
main(){
   int i=1, j=2;
   swap(i,j);
   printf("i=%d, j=%d\n", i, j);
}
```



Are these Pass-by-Value?

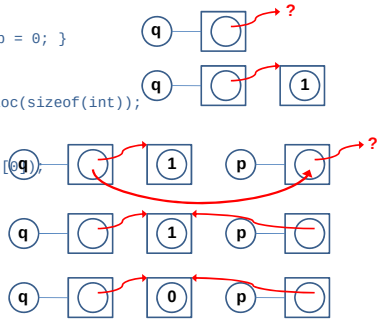
- C:
  ```
  void f(int *p) { *p = 0; }

  void f(int a[]) { a[0]=0; }
  ```
- Java:
  ```
  void f(Vector v) { v.removeAll(); }
  ```

Yes!

Pass-by-Value: Pointers

- C:

```
void f(int *p) { *p = 0; }
main() {
   int *q;
   q = (int *) malloc(sizeof(int));
   *q = 1;
   f(q);
   printf("%d\n", q[0]);
}
```



Pass-by-Value: Pointers

- C:

```
void f(int *p) { p = (int *) malloc(sizeof(int)); *p = 0; }
main() {
   int *q;
   q = (int *) malloc(sizeof(int));
   *q = 1;
   f(q);
   printf("%d\n", q[0]);
}
```

- What happens here?

Pass-by-Value: Arrays

- C:

```
void f(int p[]) { p[0] = 0;}
main() {
   int q[10];
   q[0]=1;
   f(q);
   printf("%d\n", q[0]);
}
```

- What happens here?

Pass-by-Value: Arrays

- C:

```
void f(int p[]) { p=(int *) malloc(sizeof(int)); p[0] = 0; }
main() {
   int q[10];
   q[0]=1;
   f(q);
   printf("%d\n", q[0]);
}
```

- What happens here?

Pass-by-Value: Java Objects

- Java:

```
void f(Vector v) { v.removeAll(); }

main() {
   Vector vec;
   vec.addElement(new Integer(1));
   f(vec);
   System.out.println(vec.size());
}
```

- What happens here?

Pass-by-Value: Java Objects

- Java:

```
void f(Vector v) { v = new Vector(); v.removeAll(); }

main() {
   Vector vec;
   vec.addElement(new Integer(1));
   f(vec);
   System.out.println(vec.size());
}
```

- What happens here?

Pass by Reference

- Caller:

```
…
f(i);
…
```

Callee:

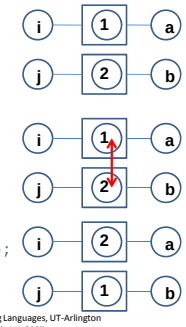
```
int f(int a){
   ...a...;
}
```

- Formal parameters become **alias** of actual parameters
- Actual parameters: changed by changes to formal parameters
- Examples:
  - Fortran: the only parameter passing mechanism
  - C++ (reference type, &) /Pascal (`var`)

Example: Pass By Reference

C++ syntax. Not valid in C

```
void swap(int &a, int &b) {
   int temp;
   temp = a;
   a = b;
   b = temp;
}
main(){
   int i=1, j=2;
   swap(i,j);
   printf("i=%d, j=%d\n", i, j);
}
```



Pass-by-Reference: How to minic it in C?

- C:

```
void f(int *p) { *p = 0; }
main() {
   int q;
   q = 1;
   f(&q);
   printf("%d\n", q);
}
```

- It is really pass-by-value. Why?

It is really pass-by-value

- C:

```
void f(int *p) { p = (int *) malloc(sizeof(int)); *p = 0; }
main() {
   int q;
   q = 1;
   f(&q);
   printf("%d\n", q);
}
```


Pass-by-Reference: C++ Constant Reference

- C++:

```
void f(const int & p) {
   int a = p;
   p = 0;
}
main(){
    int q;
    q = 1;
    f(q);
    printf("%d\n", q);
}
```

- What happens here?

Pass-by-Reference: C++ Reference-to-Pointer

- C++:

```
void f(int * &p) { *p = 0; }
main(){
    int *q;
    int a[10];
    a[0]=1;
    q=a;
    f(q);
    printf("%d, %d\n", q[0], a[0]);
}
```

- What happens here?

Pass-by-Reference: C++ Reference-to-Pointer

- C++:

```
void f(int * &p) { p = new int; *p = 0; }
main(){
    int *q;
    int a[10];
    a[0]=1;
    q=a;
    f(q);
    printf("%d, %d\n", q[0], a[0]);
}
```

- What happens here?

Pass-by-Reference: C++ Reference-to-Array

- C++:

```
void f(int (&p)[10]) {
     p[0]=0;
}
main(){
    int *q;
    int a[10];
    a[0]=1;
    q = a;
    f(a);
    printf("%d, %d\n", q[0], q[0]);
}
```

- What happens here?

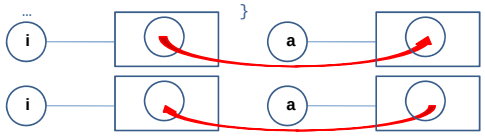
Pass by Value-Result

- Caller:

```
…
f(i);
…
```

Callee:

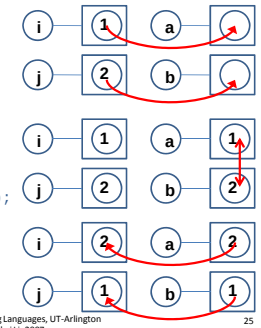
```
int f(int a){
   ...a...;
}
```



- Combination of Pass-by-Value and Pass-by-Reference (Pass-by-Reference without aliasing)
- Replace formal parameters by the values of actual parameters
- Value of formal parameters are copied back to actual parameters
- Algol W, Ada (`in out`)

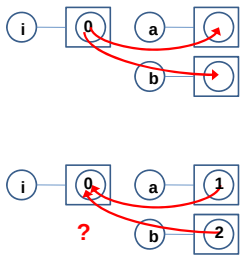
Example: Pass By Value-Result

```
void swap(int a, int b) {
   int temp;
   temp = a;
   a = b;
   b = temp;
}
main(){
   int i=1, j=2;
   swap(i,j);
   printf("i=%d, j=%d\n", i, j);
}
```



Unspecified Issues

```
void f(int a, int b) {
   a = 1;
   b = 2;
}
main(){
   int i=0;
   f(i,i);
   printf("i=%d\n", i);
}
```



Pass by Name

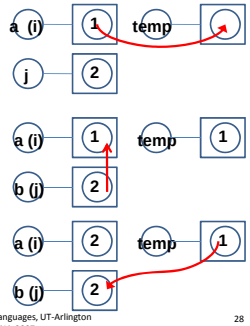
- Caller:
  …
  `f(i);`
  …

  Callee:
  ```
  int f(int a){
     ...a...;
  }
  ```

- Actual parameters only evaluated when they are needed
- The same parameter can be evaluated multiple times
- Evaluated in calling environment
- Essentially equivalent to normal order evaluation
- Example:
  - Algol 60
  - Not adopted by any major languages due to implementation difficulty

Example: Pass By Name

```
void swap(int a, int b) {
   int temp;
   temp = a;
   a = b;
   b = temp;
}
main(){
   int i=1, j=2;
   swap(i,j);
   printf("i=%d, j=%d\n", i, j);
}
```



Pass-by-Name: Side Effects

```
int p[3]={1,2,3};
int i;

void swap(int a, int b) {
   int temp;
   temp = a;
   a = b;
   b = temp;
}
main(){
   i = 1;
   swap(i, a[i]);
   printf("%d, %d\n", i, a[i]);
}
```

- What happens here?

Some Variants

- Pass by Name
  – Evaluated at every use, in the calling environment
- Pass by Need
  – Evaluated once, memorized for future use
- Pass by Text (Macro)
  – Evaluated using the called environment.

- All belong to Non-strict evaluation (lazy evaluation)

Comparions

- Call by Value
  - Efficient. No additional level of indirection.
  - Less flexible and less efficient without pointer.
    - (array, struct, union as parameters)
- Call by Reference
  - Require one additional level of indirection (explicit dereferencing)
  - If a parameter is not variable (e.g., constant), a memory space must be allocated for it, in order to get a reference.
  - Easiest to implement.
- Call by Value-Result
  - You may not want to change actual parameter values when facing exceptions.
- Call by Name
  - Lazy evaluation
  - Difficult to implement