Programming Language Basics

Environments and State

Environment: is a mapping from names to locations (variables) in the store.
State: is a mapping from locations in store to their values.

$$\begin{tikzcd} \text{names} \arrow[r, bend left, "environment"] & \text{locations (variables)} \arrow[r, bend left, "state"] & \text{values} \end{tikzcd}$$

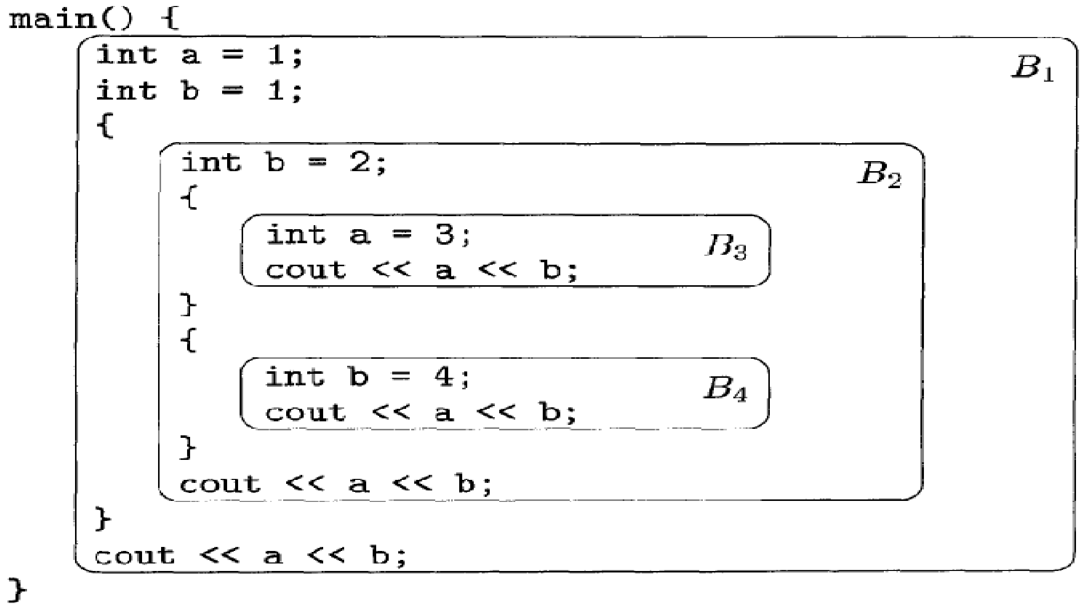
State of x may change dynamically by assignment statement. For example: x=x+5

Environment for x may change – if x is local to the class. Locations of x with respect to 'n' objects will be decided later

Static Scope and Block structure

Static scope

- can be enforced by **access specifiers** namely, private, public and protected
- can be enforced by block structure



```
main() {
    int a = 1;
    int b = 1;
    {
        int b = 2;
        {
            int a = 3;
            cout << a << b;
        }
        {
            int b = 4;
            cout << a << b;
        }
        cout << a << b;
    }
    cout << a << b;
}
```

Scope of declaration

Declaration	**Scope**
int a=1;	B1-B3
int b=1;	B1-B2
int b=2;	B2-B4
int a=3;	B3
int b=4;	B4

In static scoping, a variable used in a block is determined from the declaration within the block. Otherwise, it is determined from the outer block in which the block is nested. If the declaration is not found even in the outer blocks, then error is generated.

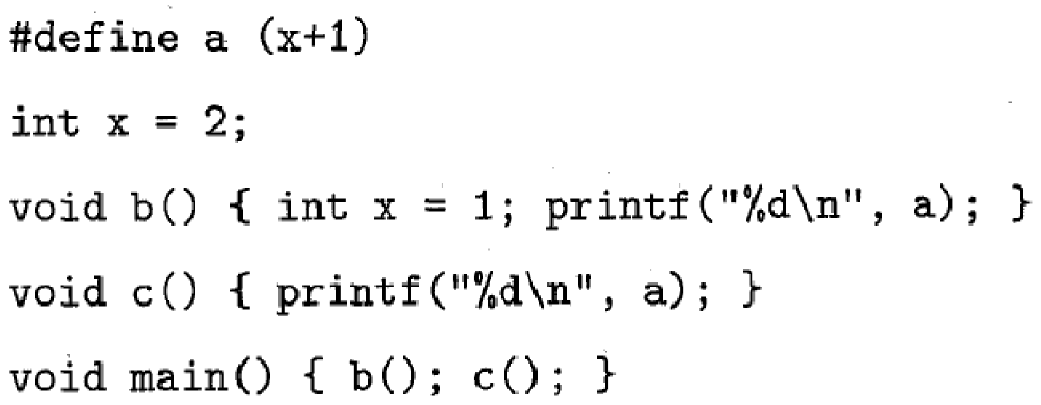
The **output generated** by the above code segment is given as follows.
cout << a << b; in B3 : 3 2 (a from B3 and b from B2)
cout << a << b; in B4 : 1 4 (a from B1 and b from B4)
cout << a << b; in B2 : 1 2 (a from B1 and b from B2)
cout << a << b; in B1 : 1 1 (a from B1 and b from B1)

Dynamic scoping policy

If it is based on factor(s) that can be known only when the program executes

Identifies ‘a’ in the code is a macro that stands for an expression x+1. What is the value of x?

It cannot be resolved statically.



```
#define a (x+1)

int x = 2;

void b() { int x = 1; printf("%d\n", a); }

void c() { printf("%d\n", a); }

void main() { b(); c(); }
```

Parameter Passing Mechanisms

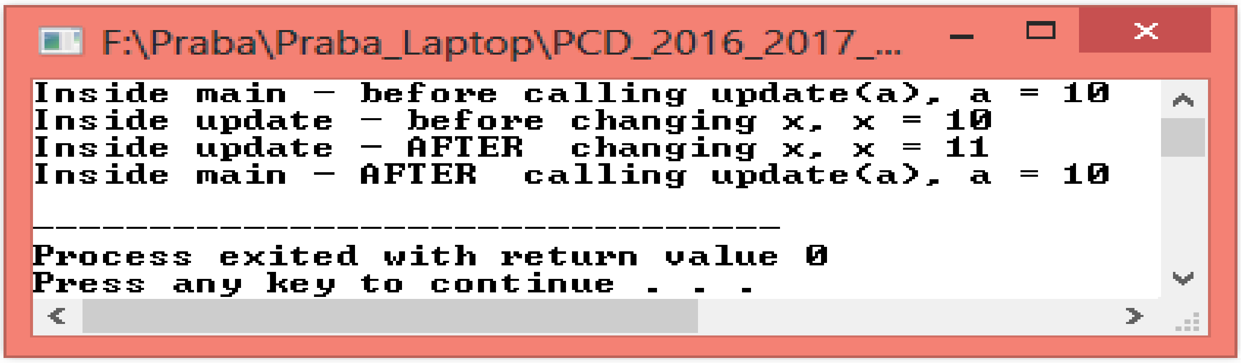
Call by value - The value of the actual parameter has been substituted in the formal parameter.

Example

```
#include<stdio.h>
void update( int x )    // a is passed by value
{
  printf( "Inside update - before changing x, x = %d\n", x);
  x = x + 1;
  printf( "Inside update - AFTER  changing x, x = %d\n", x);
}

int main()
{
  int a = 10;

  printf( "Inside main - before calling update(a), a = %d\n", a);
  update ( a );
  printf( "Inside main - AFTER  calling update(a), a = %d\n", a);
}
```



```
Inside main - before calling update(a), a = 10
Inside update - before changing x, x = 10
Inside update - AFTER  changing x, x = 11
Inside main - AFTER  calling update(a), a = 10

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

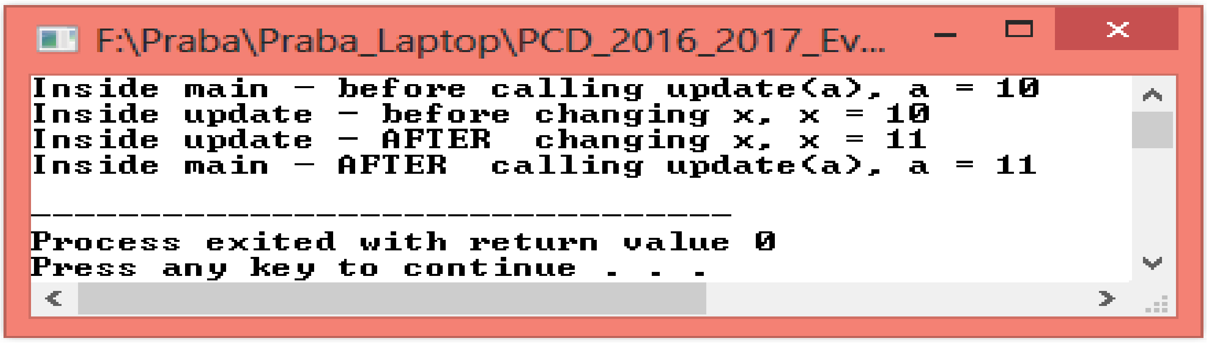
Call by reference – Address of the actual parameter has been substituted in the pointers which are formal parameters.

Example

```
#include<stdio.h>
void update( int *x )    // a is passed by value
{
  printf( "Inside update - before changing x, x = %d\n", *x);
  *x = *x + 1;
  printf( "Inside update - AFTER  changing x, x = %d\n", *x);
}

int main()
{
  int a = 10;
```

```
    printf( "Inside main - before calling update(a), a = %d\n", a);
    update ( &a );
    printf( "Inside main - AFTER  calling update(a), a = %d\n", a);
 }
```



```
F:\Praba\Praba_Laptop\PCD_2016_2017_Ev...
Inside main - before calling update(a), a = 10
Inside update - before changing x, x = 10
Inside update - AFTER  changing x, x = 11
Inside main - AFTER  calling update(a), a = 11

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

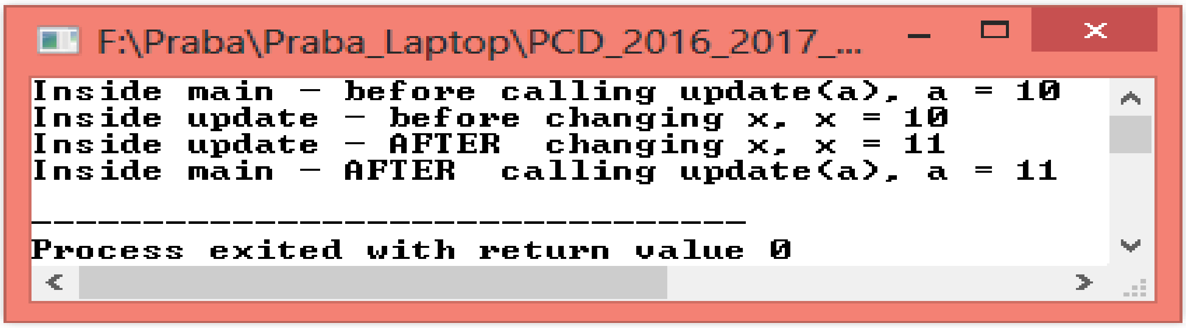
Aliasing – Aliases are created corresponding to the actual parameters

Example

```
 #include<stdio.h>
 void update( int &x )    // a is passed by value
 {
    printf( "Inside update - before changing x, x = %d\n", x);
    x = x + 1;
    printf( "Inside update - AFTER  changing x, x = %d\n", x);
 }

 int main()
 {
    int a = 10;

    printf( "Inside main - before calling update(a), a = %d\n", a);
    update ( a );
    printf( "Inside main - AFTER  calling update(a), a = %d\n", a);
 }
```



```
F:\Praba\Praba_Laptop\PCD_2016_2017_...
Inside main - before calling update(a), a = 10
Inside update - before changing x, x = 10
Inside update - AFTER  changing x, x = 11
Inside main - AFTER  calling update(a), a = 11

--------------------------------
Process exited with return value 0
```