Pass by Value vs. Reference vs. Name

Pass by Value

- Pass By Value: the value of a variable is copied into the parameter
- Example:

```
void check (int x) {
        //body of function
}

int main ( ) {
        int b = 10;
        check (b);
}
```

Pass by Value

- The variable ‘b’ is passed by value as the parameter ‘x’ to the square() routine. The value of ‘b’ is copied in ‘x’. The square() function operates on the copy only.
- Example:

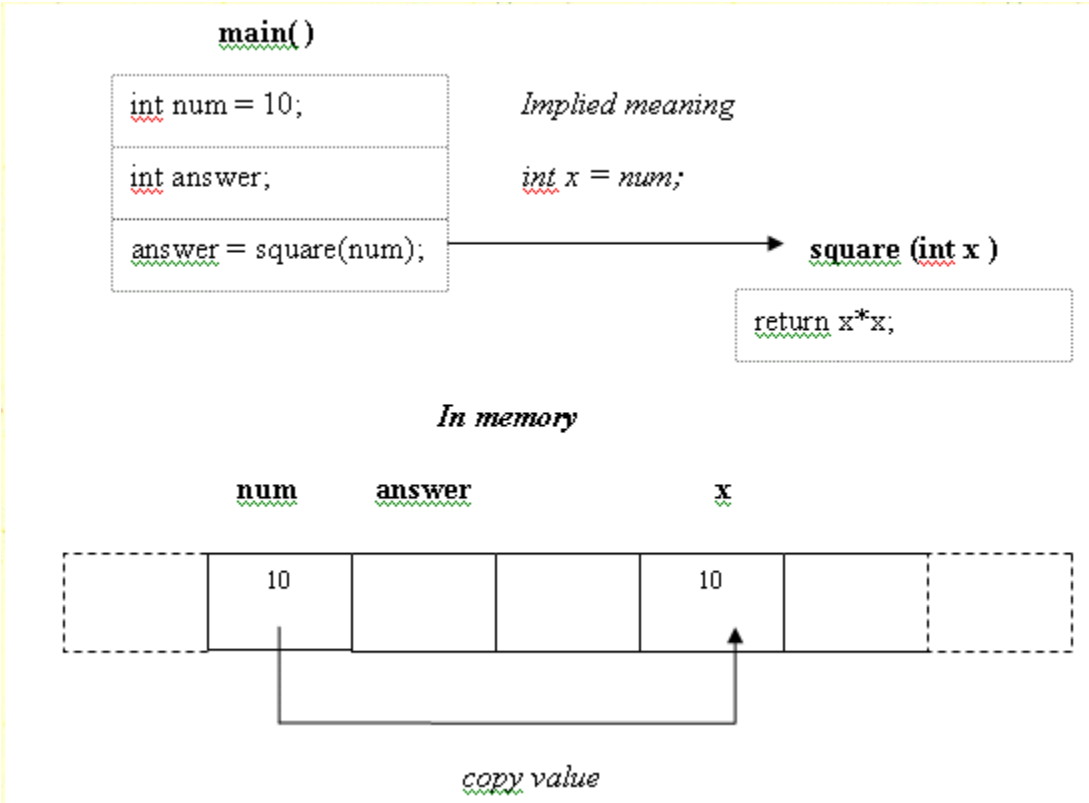
```
#include <iostream.h>

int square (int x) {
        return x*x;
}

int main ( ) {
        int num = 10;
        int answer;
        answer = square(num);
        cout<<"Answer is "<<answer; // answer is 100
        cout<<" Value of a is "<<num; // num will be 10
        return 0;
}
```

Path by Value: Analysis

- The value of the original is unchanged.



Pass by Reference (in C)

- Function operates on the original copy.
- A reference to the original variable is passed.
- Example:

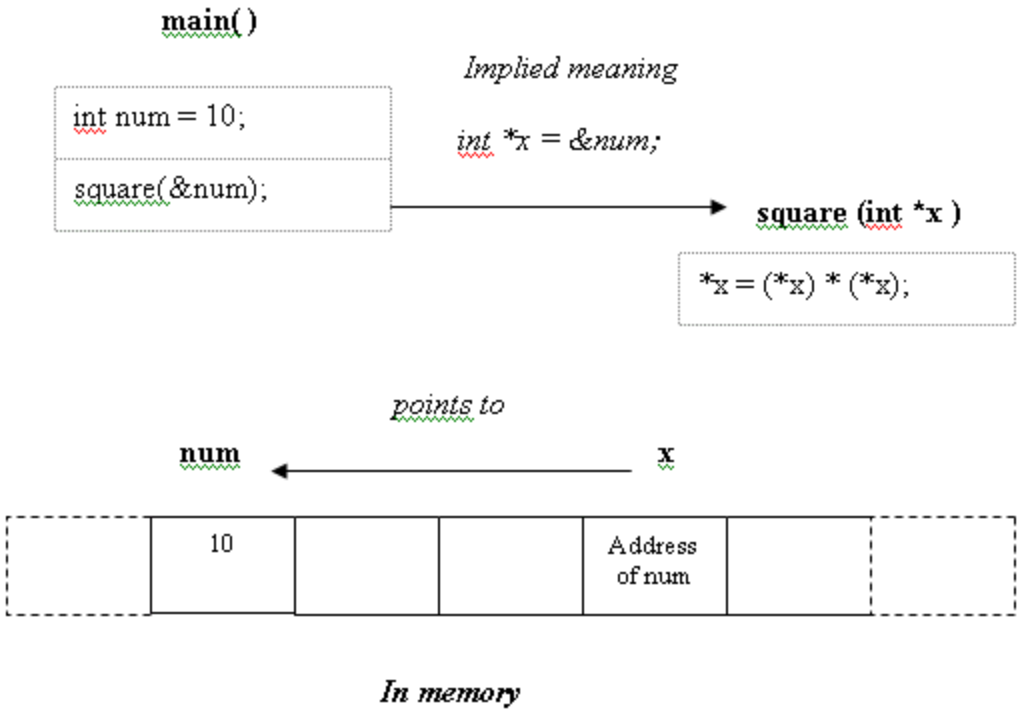
```
#include <iostream.h>

void square (int *x) {
        *x = (*x) * (*x);
}

int main ( ) {
        int num = 10;
        square(&num);
        cout<<" Value of num is "<<num; // Value of num is 100
        return 0;
}
```

Pass by Reference Analysis

- The variable ‘x’ is a pointer of type int that references (is pointing to) the variable ‘num’.
- The value of ‘num’ is manipulated.



Pass by Reference (in C++)

- C++ uses a reference parameter, but the behavior is the same.
- Example:

```
void square ( int &x ) // x is a reference parameter {
        x = x * x;
}


int main ( ) {
                int num =10;
                int answer;
                square(num);
                cout<<" Value of num is "<<num; // Value of a is 100
                return 0;
}
```

Other uses for Reference Vars

- The variable ‘ref’ is a reference variable to the variable ‘x’. Any changes to ‘x’ will also be made to ‘ref’
- Example:

```
#include <iostream.h>

int main( ) {
        int x;
        int &ref = x; //ref is a reference variable
        x=5;
        cout<<endl<<x<<" "<<ref;
        x++;
        cout<<endl<<x<<" "<<ref;
        ref++;
        cout<<endl<<x<<" "<<ref;
        cout<<endl<<&x<<" "<<&ref;
        return 0;

}
```

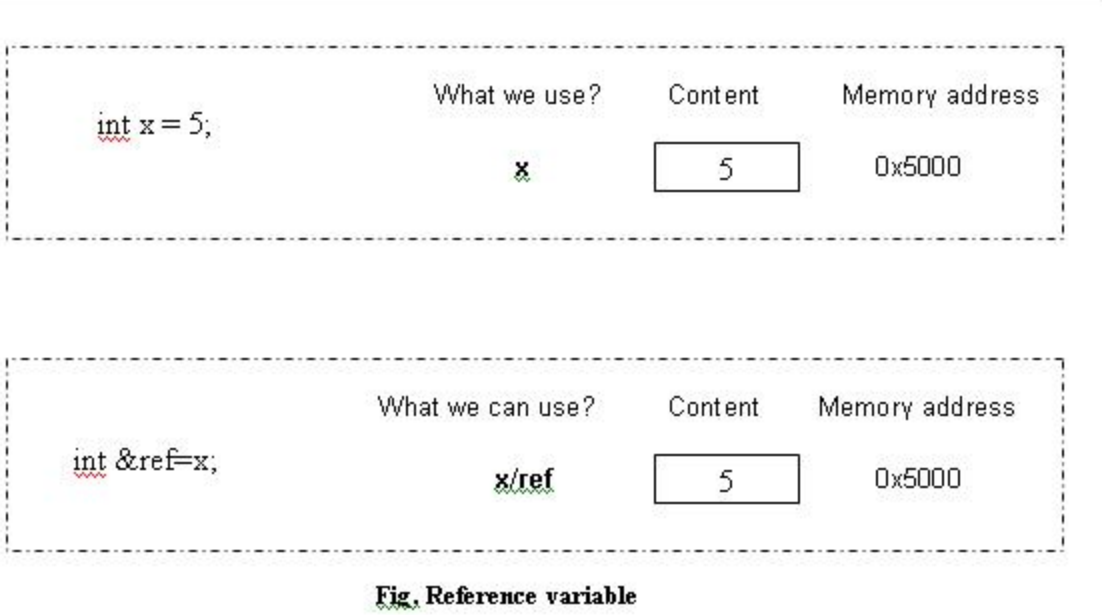
The output is:

```
5 5
6 6
7 7
0x0065FDF4  0x0065FDF4
```

Other uses for Reference Vars

- Reference variable are also aliases.
- Example:

```
int x;
int &ref=x; // 'ref' is another name for 'x'
```



Fig, Reference variable

Pass by Reference vs. Pointers

- Passing by reference is similar to pointers except for pointers have to deal with a lot of dereferencing.

Passing pointers (C style)	Passing by reference

void square (int *x)
{
*x = (*x) * (*x);
}
	void square (int &x)
	{
	x = x * x;
	
	}

Using ‘const’

- Using ‘const’ prevents modifying the argument when using reference parameters.
- Example:

```
void print(const int &x) {
        cout<<endl<<"The value is:"<<x;
}

void square (int &x) {
        x = x * x;
}

int main ( ) {
        int y=5;
        print(y);
        square(y);
        print(y);
        return 0;
}
```

Using ‘const’ (cont’d)

- Trying to change a reference parameter when using ‘const’ will cause a compilation error.
- Example:

```
void square (const int &x) {
    x = x * x; // this change is illegal – compilation error
}
```

Returning a Reference

- References can be returned from a function, but can get unexpected results because the returned variable will be destroyed when reach the end of the function that returns it. The variable ‘x’ is an alias, and so when ‘x’ is destroyed, it should not be used.
- Example:

```
int& dummy( ) {
        int x=5;
        return x;
}

int main ( ) {
        cout<<dummy( );
        return 0;
}
```

Returning a Pointer

- Pointers can also be returned. Returning pointers is basically returning an address from the function, and need to be careful of the syntax.
- Example:

```
#include <iostream.h>

int* create( ) {
        int marks[3];
        int *pt=marks;
        for (int i=0;i<3;i++) {
                marks[i]=80;
        }
        return pt;
}
int main( ) {
        int *p;
        p=create( );
        cout<<endl<<"The marks are : "<<*(p)<<" "<<*(p+1) <<" "<<*(p+2);
        return 0;
}
```

Output: **The marks are : 80 80 80**

Pass by Name (Algol)

- Passing by name allows both for it to be accessed and updated. You can pass a variable by its name rather than its value.
- Every call statement is replaced by the body of the call method
- Each occurrence of the parameter in the called method is replaced with the corresponding argument (the actual text of the argument and not the value)
- Example:

```
void Init(int x, int y) {
        for (int k = 0; k < 10; k++) {
        y = 0;
            x++;
        }
}

main() {
        int j;
        int A[10];

        j = 0;
        Init(j, A[j]);
}
```

Pass by Name (Algol) (cont'd)

- Below is an example of how the substitution occurs (substituted code in red).
- Instead of passing values or addresses, an address to a method is passed for each argument (called 'thunk'), and each thunk knows how to determine the address of the corresponding argument. Each time the parameter is used, the thunk is called and the address returned by the thunk is used. In the example below, for call by reference, A[0] = 0 would be executed ten times, but in call by name, the entire array is initialized.
- Below, the thunks are:
  - 'thunk' for j - return address of j
  - 'thunk' for A[j] - return the address of the jth element of A, using the current value of j
- Example:

```
main() {
        int j;
        int A[10];

        j = 0;
        for (int k = 0; k < 10; k++) {
                A[j] = 0; <== actual A[j] for formal y
            j++;     <== actual j for formal x
        }
}
```