Introduction to Pointers

Dr. Aminul Haque
Associate Professor, Department of CSE
Daffodil International University, Dhaka, Bangladesh

Topics

1. Advantages of using pointers in a program
2. Pointers to variables
3. Using pointers in functions
4. Arrays & strings as pointers
5. Pointer arithmetic

Advantages of using pointers in a program

1. Pointers used to access the address of the variable
2. Pointers increase the execution speed of program
3. Pointers are an important concept while implementing data structures such as link lists, trees & graphs
4. Pointers are used for dynamic memory allocation
5. Pointers makes possible to return more than one value in functions
6. Pointer enables us to access variables that are declared outside the functions
7. Strings and arrays are more efficient with pointers

Pointers to variables

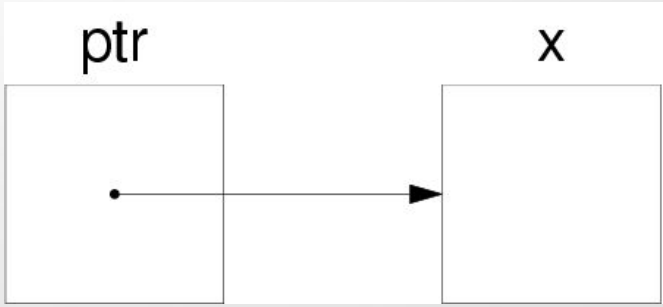
A pointer is nothing but a variable that contains an address of a location in memory. We can declare the variable ptr as a pointer to an integer as:

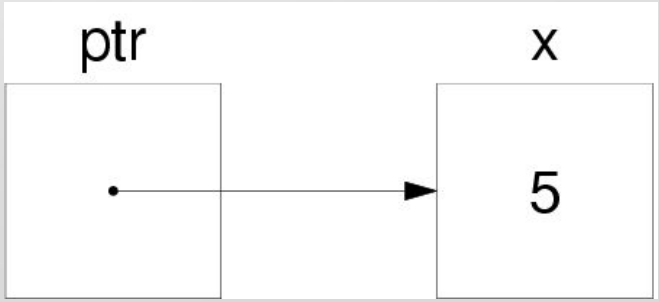
```
int *ptr;
```

Let us consider the following example:

```
int x;
int *ptr;
ptr = &x;
```

Now extending the code with x=5;



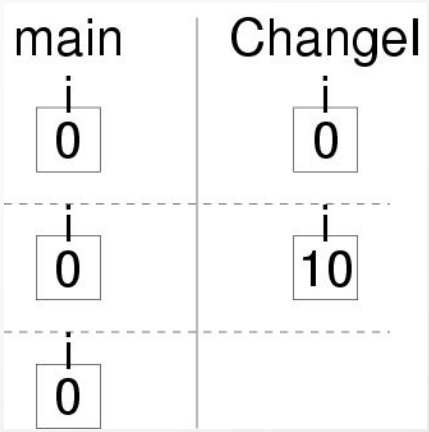


Pointers to variables

1. Dereferencing the pointer:
2. *ptr = 5;
3. The asterisk ‘*’ symbol is called dereferencing operator.

Using pointers in functions

- Passing arguments to functions:
  - void ChangeI(int i);
  -
  - Main( ) {
    int i=0;
    ChangeI(i);
    Print(i);
  - }
  - void ChangeI(int i) {
    i=10;
  - }

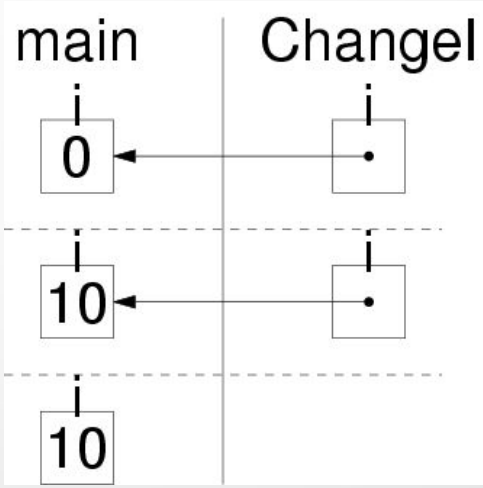


- The value of *i* is not changed because the argument to the function is ***passed by value***.

Using pointers in functions

Passing the arguments to the function by reference (***passed by reference***):

```
void Changel(int *i);
main() {
      int i=0;
      Changel(&i);
      Print(i);
}
void Changel(int *i) {
      *i=10;
}
```



Using pointers in functions

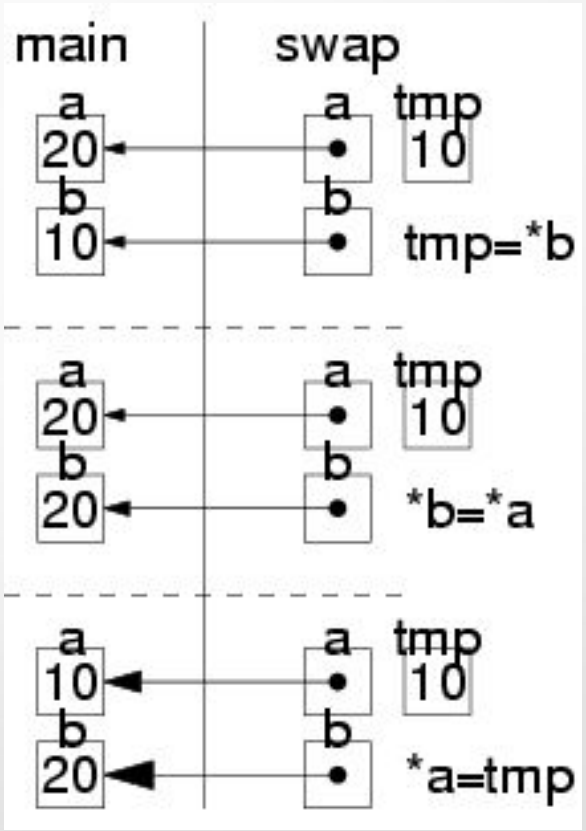
Significance of passing variables by reference:

```
- void swap(int a, int b) {
    int tmp = b;
    b=a;
    a=tmp;
- }
    main() {
        int a=10, b=20;
        Swap(a, b);
        Print(a, b);
    }
```

Now, calling the function *swap(a, b)* will not actually swap the values; however, changing it to *swap(&a, &b)* will do the work.

Using pointers in functions

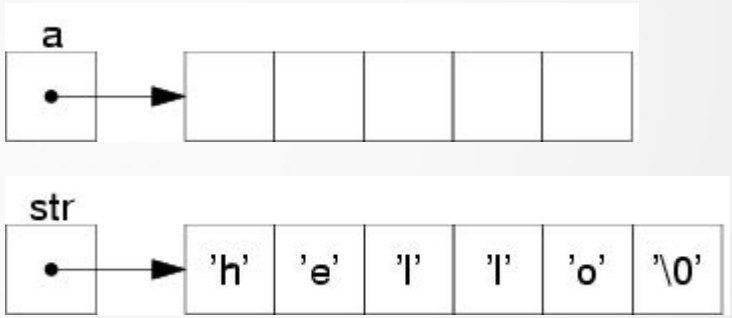
```
void swap(int *a, int *b) {
      int tmp = *b;
      *b=*a;
      *a=tmp;
}

From main( ),

int a=20,b=10;
swap(&a,&b);
Print(a, b);
```



Arrays & strings as pointers

On the stack (static):

- float a[5];
-
- char str[ ] = "hello";

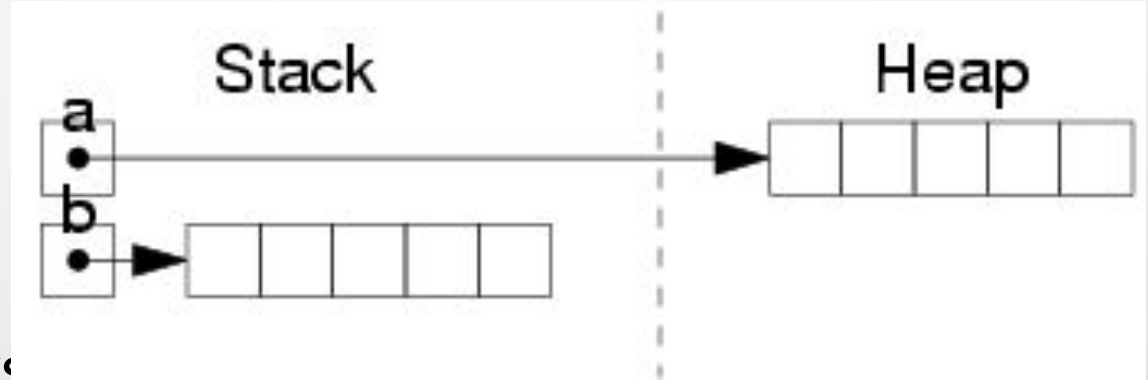


Arrays & strings as pointers

Dynamically allocated in the heap:

```
float *a = malloc(5*sizeof(float));
```

The sizeof(float) will return 4 because floating point number requires 4 bytes of memory.



The benefit of using arrays that are dynamically allocated in the heap is that we can free up the memory that was allocated for them with:

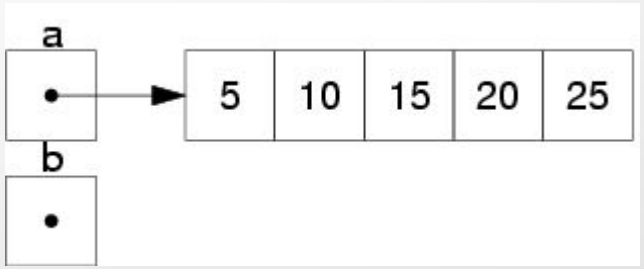
```
free(a);
```

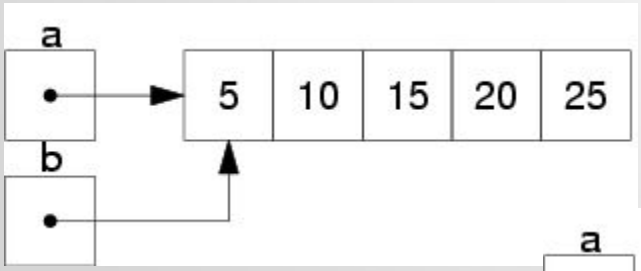
Pointer arithmetic

while every pointer only needs 4 bytes to store a memory address, the compiler also needs to know the type of variable that it points to in order to perform pointer arithmetic. Let us consider,

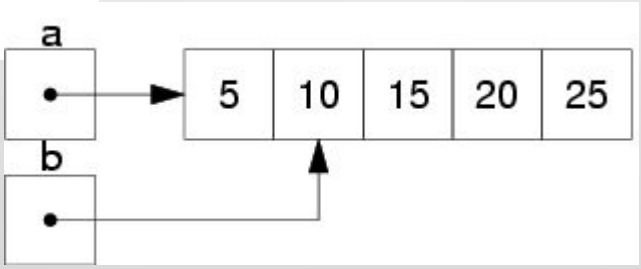
```
int *b, a[ ]={5,10,15,20,25};
```

Now, b=a





Again, b=a+1



Thanks!