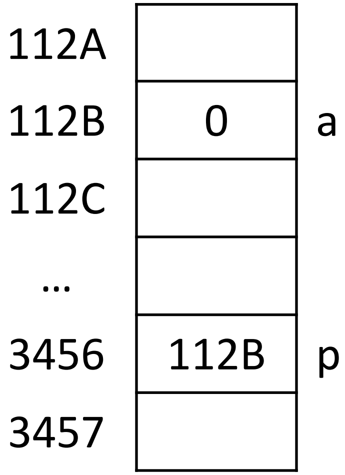
CMSC 21
Fundamentals of Programming

2nd Semester 2011-2012

POINTERS

Pointer

- A memory location or an address of another variable in the memory

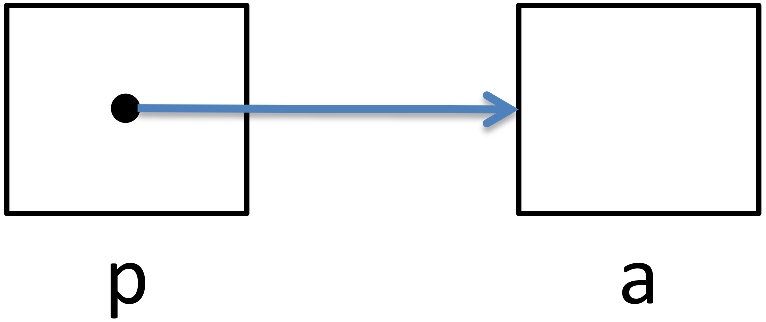


112A		
112B	0	a
112C		
…		
3456	112B	p
3457		

Here, p holds the address where a is located in the memory, thus, p is a pointer

Pointer

- A **pointer variable** holds the address in the memory of another variable
- A pointer can also be illustrated as:



Pointer Variables

- A pointer variable is declared as:

```
<data_type> * <variable_name>
```

```
int *p;   //p is a pointer to an int
float *q; //q is a pointer to a float
```

Pointer Variables

- `<data_type>`
  - Defines the type of variables that a pointer can point to
  - C assumes that the variable a pointer refers to is an object of `<data_type>`
  - Pointer arithmetic is done relative to the `<data_type>`

Pointer Operators

- Indirection Operator (`*`)
  - Returns the value of the variable located at the specified address
  - “The value/variable at memory address …”
- Address Operator (`&`)
  - Returns the memory address of its operand
  - “The memory address of …”

Example (1)

```
main () {
   int source;
   int target;
   int *p, *q;

   source = 8;
   p = &source;
   target = *p;
   q = p;
}
```

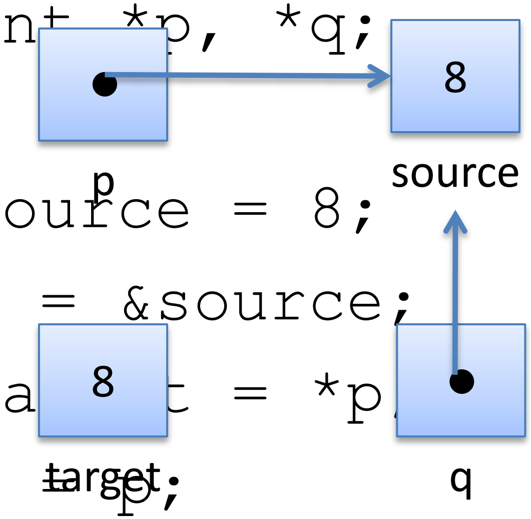
112A		
112B	8	source
112C		
…		
3452	8	target
3454		
3455		
3456	112B	p
3457	112B	q
3458		
3459		

Example (2)

- This example can also be viewed as:

```
main () {
    int source;
    int target;
    int *p, *q;

    source = 8;
    p = &source;
    target = *p;
    q = p;
}
```



Advantages of Using Pointers

- modify actual parameters
- Efficient accessing of array elements
- Improve the efficiency of certain routines
- Used to support dynamic data structures like linked list and binary trees

Notes

- Make sure that a pointer refers to a memory location before using it

```
/*This program is wrong!*/
int main () {
   int x, *p;
   x = 8;
   *p = x;   //invalid
}
```

Notes

- There should be a correspondence to the `<data_type>` when assigning values to pointers

```
int main () {
   int x = 8, *p;
   p = x;         //this is wrong!
   printf (“%d”, *p);
}
```

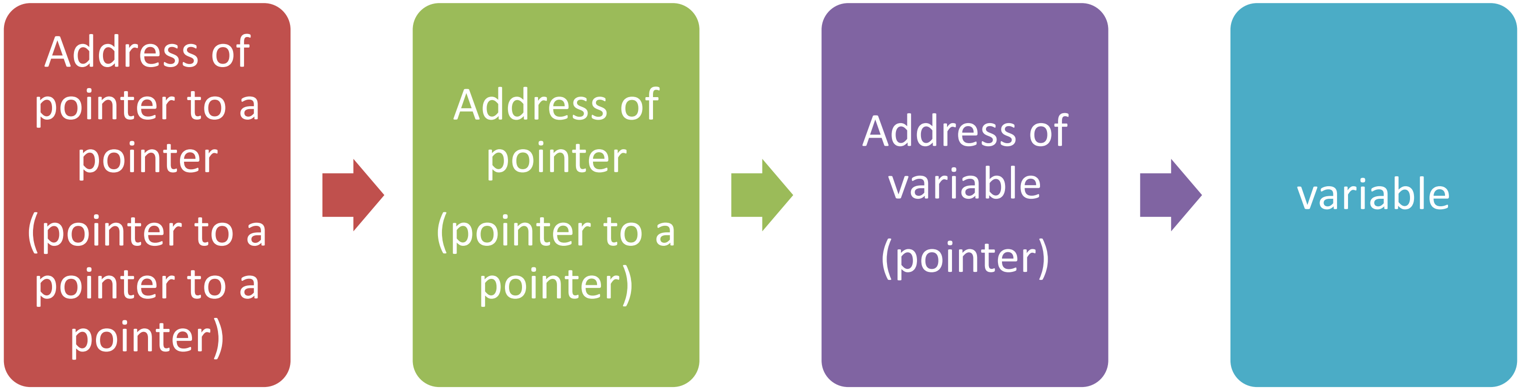
Notes

- There should be a correspondence to the `<data_type>` when assigning values to pointers

```
int main () {
   int x = 8, *p;
   float *q;
   p = &x;
   q = p;             //this is wrong!
}
```

Pointers to Pointers

- Chain of pointers
- Multiple indirection
- A pointer to a pointer holds an address of a pointer, a pointer to a pointer to a pointer holds an address of a pointer to a pointer and so on…



Pointers to Pointers

- To declare pointers to pointers:

```
<data_type> ** <var_name>;
```

```
int *a;   //pointer to an int
int **p;  //pointer to a pointer to
          //an int
int ***q; //pointer to a pointer to a
     //pointer to an int
```

Example (1)

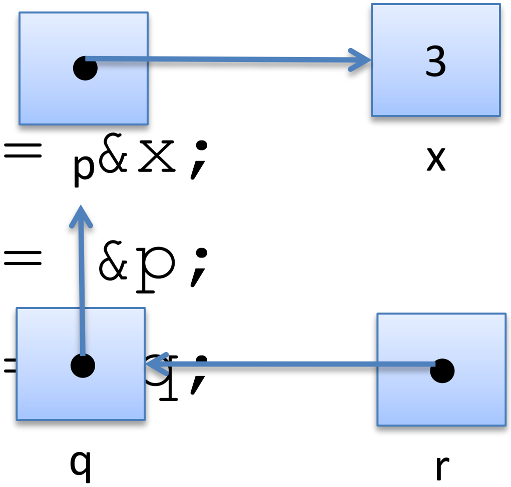
```
main () {
   int x = 3, *p, **q, ***r;
   p = &x;
   q = &p;
   r = &q;
}
```

112A		
112B	3	x
112C		
…		
3452	112B	p
3454		
3455		
3456	3452	q
3457	3456	r
3458		
3459		

Example (2)

- The example can also be illustrated as

```
main () {
    int x = 3, *p,
    **q, ***r;
    p = &x;
    q = &p;
    r = &q;
}
```



Example (3)

- x can be accessed indirectly using p, q and r

```
/*the following codes
print x on the
screen*/
printf (“%d”, *p);
printf (“%d”, **q);
printf (“%d”, ***r);
```

```
main () {
   int x = 3, *p,
   **q, ***r;

   p = &x;
   q = &p;
   r = &q;
}
```

Pointers as Parameters

- Pointers are used in pass by reference parameter passing
- If the address of a variable is passed as actual parameter, the formal parameter should be a pointer
- If the address of a pointer is passed as actual parameter, the formal parameter should be a pointer to a pointer and so on…

Example

```
/*foo prints the value of x on the screen*/
void foo (int * p) {
     printf (“%d”, *p);
}

/*main calls foo and passes the address of x
as actual parameter*/
main () {
     int x;
     foo (&x);
}
```

Example

```
/*foo prints the value of x on the screen then
calls foofoo and passes the address of p*/
void foo (int * p) {
     printf (“%d”, *p);
     foofoo (&p);
}

/*foofoo prints the value of x*/
void foofoo (int ** q) {
     printf (“%d”, **q);
}
```

QUIZ (1/4)

- Fill in the missing code

```
void one  (int **p) {
      scanf (“%d”, ___(1)___);
      two (___(2)___); //print value of x
}
void two (int *q) {
      printf (“%d”, ___(3)___);
}
main () {
      int x, *a;
      a = &x;
      one (___(4)___);
}
```

QUIZ (Answer)

- Fill in the missing code

```
void one  (int **p) {
      scanf (“%d”,   *p  );
      two (  *p  ); //print value of x
}
void two (int *q) {
      printf (“%d”,   *q  );
}
main () {
      int x, *a;
      a = &x;
      one (  &a  );
}
```