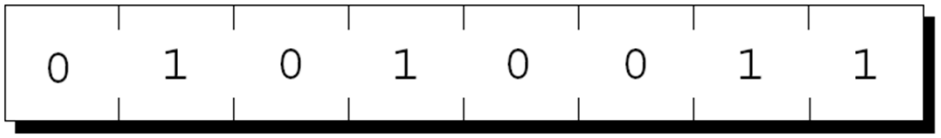
CSE102 – Programming Fundamentals

Pointers (Part 1)

Pointer Variables

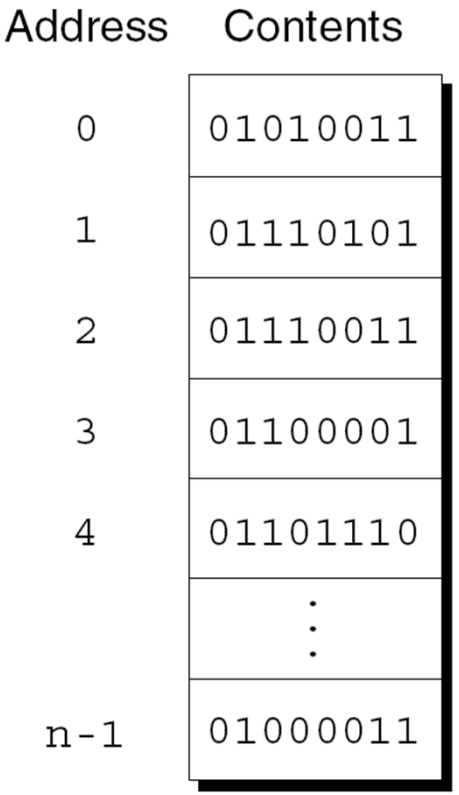
- The first step in understanding pointers is visualizing what they represent at the machine level.
- In most modern computers, main memory is divided into ***bytes,*** with each byte capable of storing eight bits of information:



- Each byte has a unique ***address.***

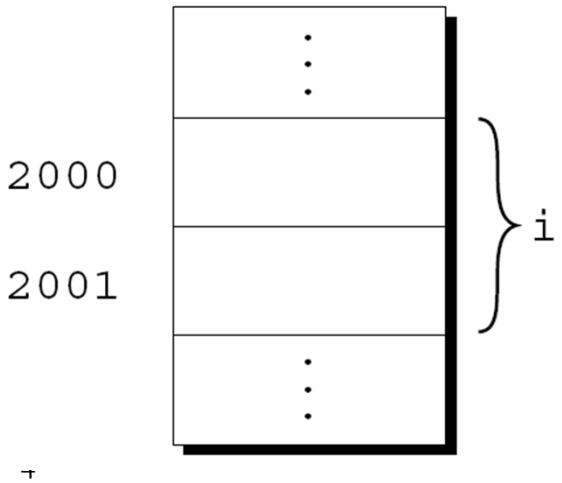
Pointer Variables

- If there are *n* bytes in memory, we can think of addresses as numbers that range from 0 to *n* – 1:



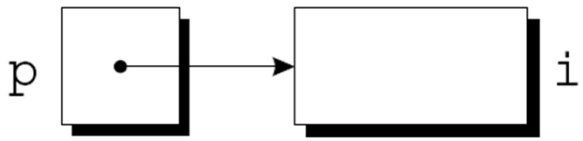
Pointer Variables

- Each variable in a program occupies one or more bytes of memory.
- The address of the first byte is said to be the address of the variable.
- In the following figure, the address of the variable `i` is 2000:



Pointer Variables

- Addresses can be stored in special ***pointer variables.***
- When we store the address of a variable `i` in the pointer variable p, we say that p "points to" `i`.
- A graphical representation:



Declaring Pointer Variables

- When a pointer variable is declared, its name must be preceded by an asterisk:

```
int *p;
```

- p is a pointer variable capable of pointing to ***objects*** of type `int`.

Declaring Pointer Variables

- Pointer variables can appear in declarations along with other variables:

```
int i, j, a[10], b[20], *p, *q;
```

- C requires that every pointer variable point only to objects of a particular type (the ***referenced type***):

```
int *p;     /* points only to integers   */
double *q;  /* points only to doubles    */
char *r;    /* points only to characters */
```

- There are no restrictions on what the referenced type may be.

Pointer Variable Declaration and Initialization

- Pointer declaration
  - Multiple pointers require using a * before each variable definition

```
int *myPtr1, *myPtr2;
```

  - Can define pointers to any data type
  - It's crucial to initialize p before we use it.
  - Initialize pointers to 0, NULL, or **an address**
    - 0 or NULL – points to nothing (NULL preferred)

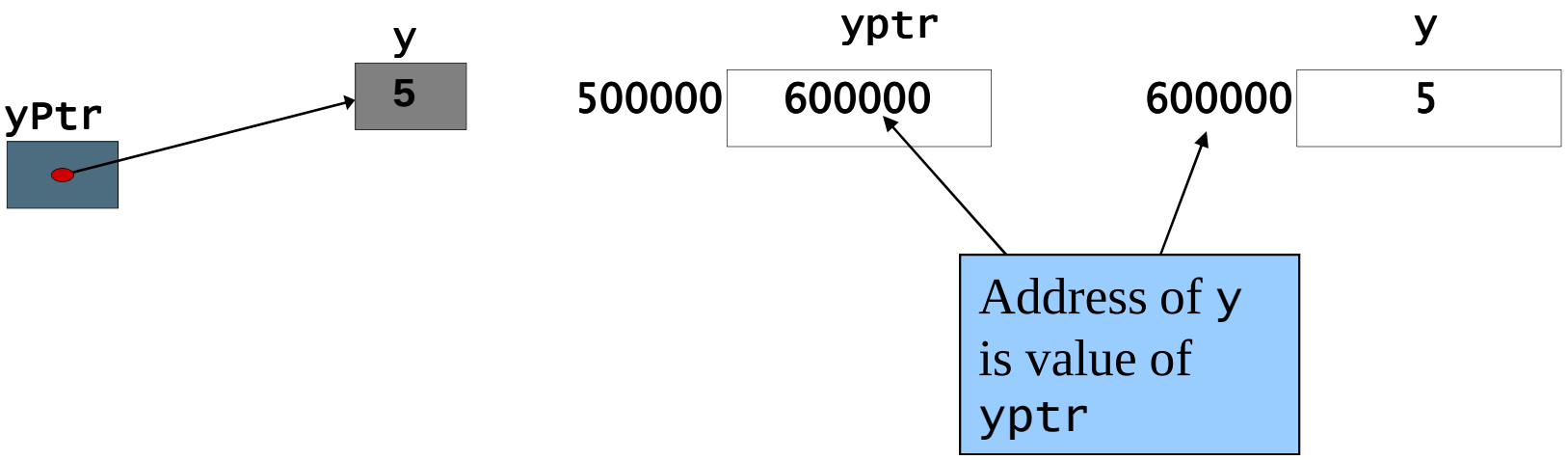
The Address and Indirection Operators

- C provides a **pair** of operators designed specifically for use with pointers.
  - &
    - To find the address of a variable, we use the & (address) operator.
  - *
    - To gain access to the object that a pointer points to, we use the * (***indirection***) operator.

Pointer Operators

- & (address operator)
  - Returns address of operand

```
int y = 5;
int *yPtr;
yPtr = &y;      /* yPtr gets address of y */
yPtr "points to" y
```



The Address Operator

- It's also possible to initialize a pointer variable at the time it's declared:

```
int i;
int *p = &i;
```

The Indirection Operator

- Once a pointer variable points to an object, we can use the `*` (indirection) operator to access what's stored in the object.
- If `p` points to `i`, we can print the value of `i` as follows:

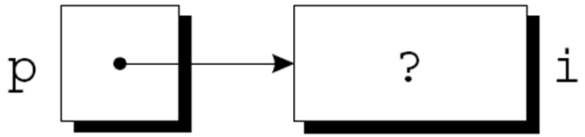
```
printf("%d\n", *p);
```

The Indirection Operator

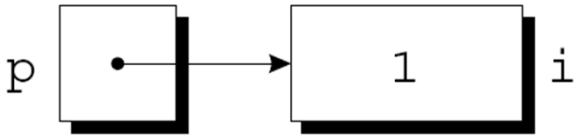
- As long as p points to `i`, `*p` is an ***alias*** for `i`.
  - `*p` has the same value as `i`.
  - Changing the value of `*p` changes the value of `i`.
- The example on the next slide illustrates the equivalence of `*p` and `i`.

The Indirection Operator

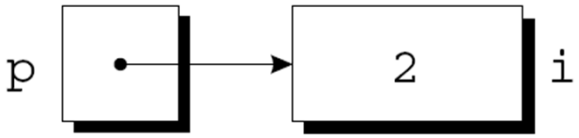
```
p = &i;
```




```
i = 1;
```




```
printf("%d\n", i);    /* prints 1 */
printf("%d\n", *p);   /* prints 1 */
*p = 2;
```




```
printf("%d\n", i);    /* prints 2 */
printf("%d\n", *p);   /* prints 2 */
```

The Indirection Operator

- Applying the indirection operator to an uninitialized pointer variable causes undefined behavior:

```
int *p;
printf("%d", *p);   /*** WRONG ***/
```

- Assigning a value to `*p` is particularly dangerous:

```
int *p;
*p = 1;   /*** WRONG ***/
```

Pointer Assignment

- C allows the use of the assignment operator to copy pointers of the same type.
- Assume that the following declaration is in effect:

```
int i, j, *p, *q;
```

- Example of pointer assignment:

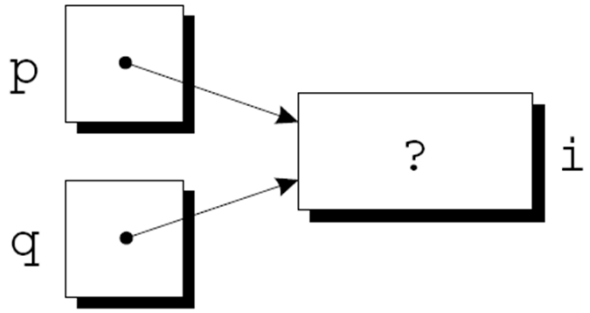
```
p = &i;
```

Pointer Assignment

- Another example of pointer assignment:

  ```
  q = p;
  ```

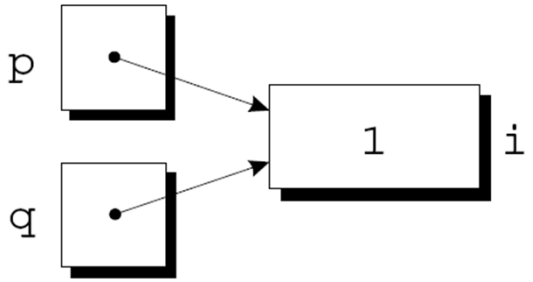
  q now points to the same place as p:



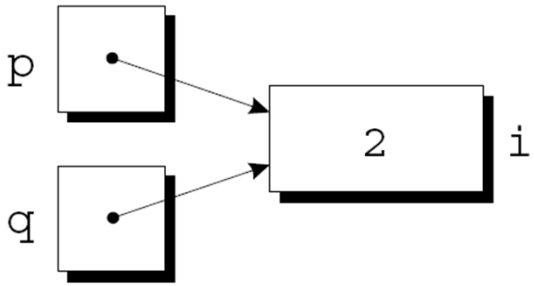
Pointer Assignment

- If p and q both point to `i`, we can change `i` by assigning a new value to either `*p` or `*q`:

```
*p = 1;
```




```
*q = 2;
```



- Any number of pointer variables may point to the same object.

Pointer Assignment

- Be careful not to confuse

  ```
  q = p;
  ```

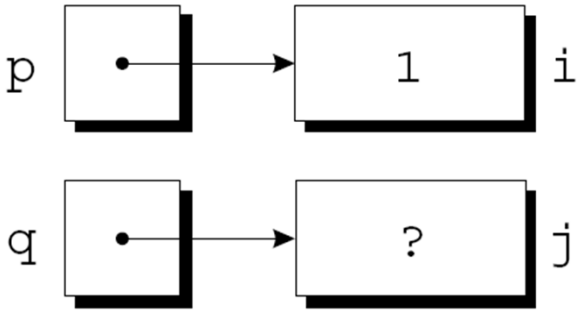
  with

  ```
  *q = *p;
  ```

- The first statement is a pointer assignment, but the second is not.
- The example on the next slide shows the effect of the second statement.

Pointer Assignment

```
p = &i;
q = &j;
i = 1;
```




```
*q = *p;
```