

## Strings and Pointers in C

---

1

### String Declarations

---

C

- Unlike Java, there is no String data type in C
- Strings in C are simply arrays of characters terminated with 0 (character '\0')

```
char some[10]; // need to specify max size
```

```
// one way:
```

```
char msg[6] = {'H','e','y',' ','!','\0'};
```

```
// another way (last 2 places unused):
```

```
char msg[8] = "Hey !"; // no '\0'
```

```
// or
```

```
char msg[.] = "Hey !"; // no '\0'
```

memory for 6 characters ( 5 plus the null char '\0' )  
automatically allocated

2

## Memory Representation

- char some[10];

```
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
|_|_|_|_|_|_|_|_|_|_|
```

- char msg[6] = {'H','e','y',' ','!','\0'};

```
[0] [1] [2] [3] [4] [5]
|'H'|'e'|'y'|' '|'!'|0|
```

- char msg[8] = "Hey !";

```
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
|'H'|'e'|'y'|' '|'!'|0|_|_|_|_|
```

- char msg[] = "Hey !";

```
[0] [1] [2] [3] [4] [5]
|'H'|'e'|'y'|' '|'!'|0|
```

3

## Reading Into a String (1)

C

```
#define MAX_BUFFER 20
char buffer[MAX_BUFFER];

scanf("%s", buffer); // or
gets(buffer, MAX_BUFFER);
```

- What if the array is not large enough to hold the input?
  - characters will be stored into memory locations past the end of the array
  - will result in run-time memory access violation error!

4

## Reading Into a String (2)

C

- Better:

```
#define MAX_BUFFER 20
char buffer[MAX_BUFFER];

fgets(buffer, MAX_BUFFER, stdin);
```

- `fgets` is similar to `gets`, but:

- it takes a third argument, in our case standard input
- if stores into buffer no more than `MAX_BUFFER` chars (extra characters are ignored), so memory violation error won't occur

5

## Functions for Manipulating Strings

C

- C provides a large number of functions for manipulating strings. Four important ones:

```
strlen(s)
// returns the length of s

strcpy(toS, fromS)
// copy fromS to toS (toS must be large enough)

strcmp(s1, s2)
// returns 0 if s1 == s2
// returns an integer < 0 if s1 < s2
// returns an integer > 0 if s1 > s2

strtok - read the Sun manual pages to find out what
        this function does
```

6

## Pointers in C

---

7

## What are Pointers?

---

C

- A pointer is a variable that holds the address of another variable (object)
- Suppose that we have an integer variable  
`int i;`  
and wish to have a pointer point to this variable. Thus we need to know the memory address of `i`.
- How do we know the address of `i`? ...  
`&i`  
is the address of `i`. The operator `&` is called the ADDRESS-OF operator.

8

## Pointers Made Easy (1)

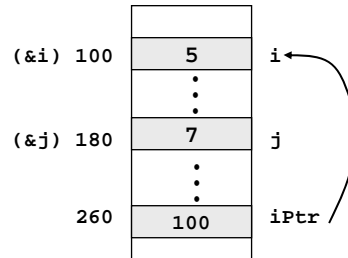
C

- We can declare that a pointer `iPtr` points to an `int` by saying

```
int * iPtr;
```

- Suppose that we have:

```
int i = 5;  
int j = 7;
```



- We can make `iPtr` point to `i` by assigning to `iPtr` the memory location where `i` is stored. Thus

```
iPtr = &i;
```

sets `iPtr` to point to `i`.



9

## Pointers Made Easy (2)

C

- We can also initialize `iPtr` at the point of declaration:

```
int i;  
int * iPtr = &i;
```



- Here is a common error:

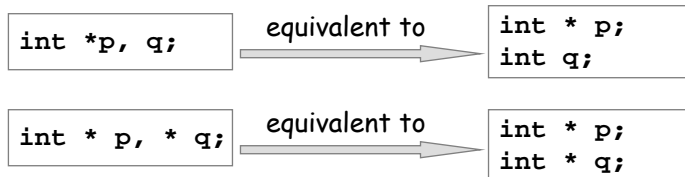
```
int i;  
int * iPtr = i; // ERROR: i is not an address
```

10

## Declaring Pointers

C

- When declaring several pointer variables in one statement - the asterisk does not distribute throughout the statement:



11

## Dereference \*

C

- The value of the data being pointed at is obtained by using the operator \*
- If `p` is a pointer value, then

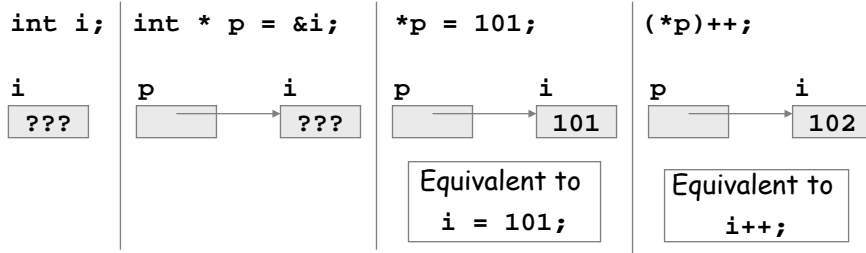
`*p`

refers to *the variable pointed to by p*. Since reference is another name for address, the operator \* is called *dereference* operator.

12

## Dereference Example

C

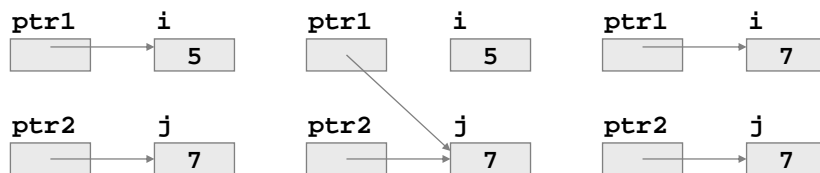


- A dereferenced pointer behaves exactly like the variable it points to.

13

## Note the Difference ...

C



Initial state

After

After

`ptr1 = ptr2;`

starting from initial  
state

`*ptr1 = *ptr2;`

starting from initial  
state

14

## Uninitialized Pointers

C

- Suppose that we have the following declarations:

```
int i;  
int * iPtr;  
*iPtr = 100;
```



- What is the value of `iPtr`? **Undefined**. What could happen?
  - `iPtr` could hold an address that does not make sense at all, causing your program to crash if dereferenced.
  - `iPtr` could point to an address which is accessible. Then the assignment `*iPtr = 100;` would accidentally change some other data, which could result in a crash at a later point. This is a tough error to detect since the cause and symptom may be widely separated in time.

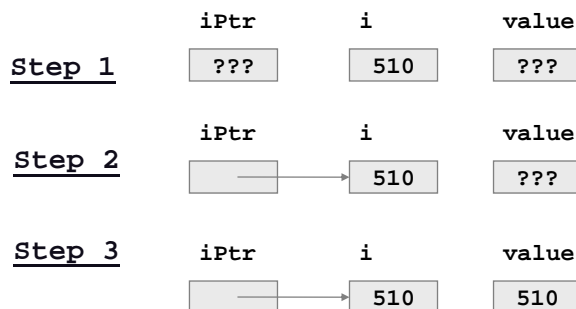
15

## Putting it all Together...

C

- ... with the help of a simple example:

```
int i, value;  
int * iPtr; // declares iPtr to be a pointer to an integer  
  
i = 510;           // Step 1  
iPtr = &i;        // Step 2  
value = *iPtr;    // Step 3
```



16

## The null Pointer

C

- The value of a pointer can be:
  - some garbage (pointer unassigned)
  - the address of some variable (eg., `int * p = &i;`)
  - the *constant 0* (the *null pointer*, points to absolutely nothing)

```
somePointer = 0;
```

This statement does not cause `somePointer` to point to memory location zero; it guarantees that `somePointer` does not point to anything

- The null pointer is a special pointer value that a program can test for:

```
if (somePointer == 0) ...
```

17

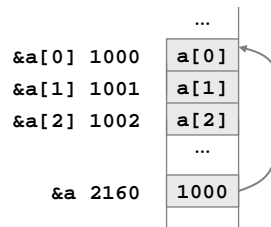
## Strings and Pointers

C

- A string name is basically a *constant pointer*

- Consider the declaration:

```
char a[3];
```



- The compiler allocates three characters for the array object. These are referenced as `a[0]`, `a[1]`, `a[2]` and occupy a *contiguous* block of memory.
- The value of `a` is exactly `&a[0]`, the address of the first integer in the array

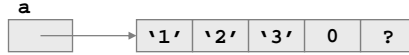
18

# Arrays and Pointers - Examples

C

- Consider the following declarations:

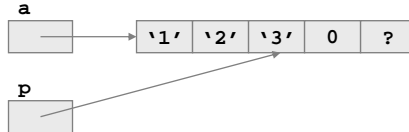
```
char a[5] = "123";
```



```
int * p;
```



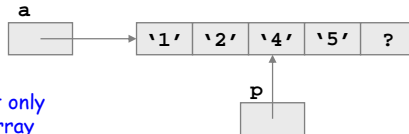
```
p = &a[2];
```



- You can use the index [ ] operator with a pointer:

```
p[0] = '4';
```

```
p[1] = '5';
```



Indexing can be used with *any* pointer, but it only makes sense when the pointer points to an array

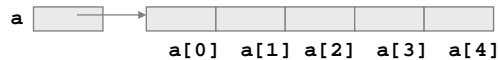
19

# Arrays are NOT Pointers

C

- Declaring an array sets aside space for its elements

```
char a[5];
```



- Declaring a pointer variable sets aside only space to hold the variable

```
char * p;
```

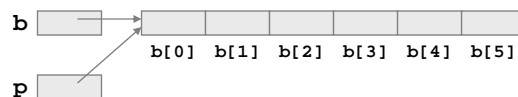


- You can change a pointer variable, but *not the address* of an array

```
char b[6];
```

```
p = b; // OK
```

```
a = b; // ERROR !
```



20

## Other resources

---

- A very good tutorial on pointers and arrays in C:

<http://pw1.netcom.com/%7Etjensen/ptr/pointers.htm>