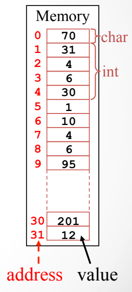
Pointers

Based on slides from K. N. King and Dianna Xu

Bryn Mawr College
CS246 Programming Paradigm

Variable and Address

- Variable = Storage in computer memory
  - Contains some value
  - Must reside at a specific location called *address*
  - Basic unit – byte
  - Imagine memory as a one-dimensional array with addresses as byte indices
  - A variable consists of one or more bytes, depending on its type (size)



Pointer – Reference

- A pointer (pointer variable) is a variable that stores an address (like Java reference)
  - value – address of some memory
  - type – size of that memory
- Recall in Java, when one declares variables of a *class* type, these are automatically references.
- In C, pointers have special syntax and much greater flexibility.

Address Operations in C

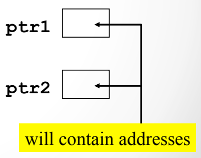
- Declaration of pointer variables
  - The *pointer declarator* '`*`'
- Use of pointers
  - The *address of* operator '`&`'
  - The *indirection* operator '`*`' – also known as de-referencing a pointer

Pointer Declaration

- Syntax
  - *destinationType* `*` *varName*`;`
- Must be declared with its associated type.
- Examples
  - `int *ptr1;`

    A pointer to an `int` variable
  - `char *ptr2;`

    A pointer to a `char` variable



Pointers are NOT integers

- Although memory addresses are essentially very large integers, pointers and integers are not interchangeable.
- Pointers are not of the same type
- A pointer's type depends on what it points to
  - `int *p1; // sizeof(*p1)=sizeof(int)`
  - `char *p2; //sizeof(*p2)=sizeof(char)`
- C allows free conversion btw different pointer types via casting (dangerous)

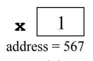
Address of Operator

- Syntax
  - **&** *expression*

    The expression must have an address. E.g., a constant such as "`1`" does not have an address.
- Example
  - `int x = 1;`

    `f(&x);`

    The address of **`x`** (i.e. where **`x`** is stored in memory), say, the memory location 567, (not 1) is passed to **`f`**.



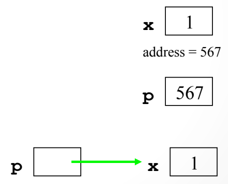
Pointer Assignment

- A pointer **`p`** *points* to **`x`** if **`x`**'s address is stored in **`p`**
- Example
  - `int x = 1;`

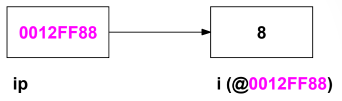
    `int *p;`

    `p = &x;`

    Interpreted as:



Pointer Diagram



```
int i = 8;
int *ip;

ip = &i;
```

Pointer Assignment

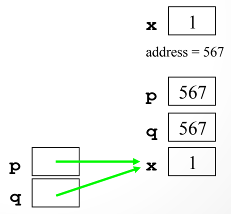
- A pointer **`p`** *points* to **`x`** if **`x`**'s address is stored in **`p`**
- Example
  - `int x = 1;`

    `int *p, *q;`

    `p = &x;`

    `q = p;`

    Interpreted as:

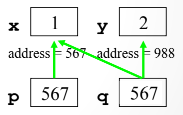


Pointer Assignment

- Example
  - `int x=1, y=2, *p, *q;`

    `p = &x; q = &y;`

    `q = p;`



Indirection Operator

- Syntax
  - ***** *pointerVar*
  - Allows access to value of memory being pointed to
  - Also called *dereferencing*

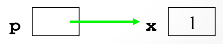
Note: '`*`' in a declaration and '`*`' in an expression are different.
`int *p; int * p; int* p;`

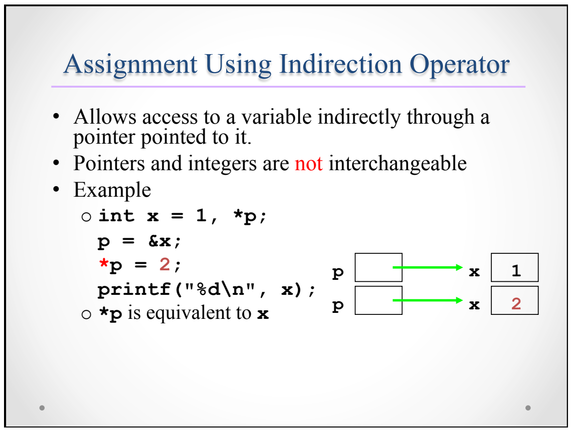
- Example
  - `int x = 1, *p;`

    `p = &x;`

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

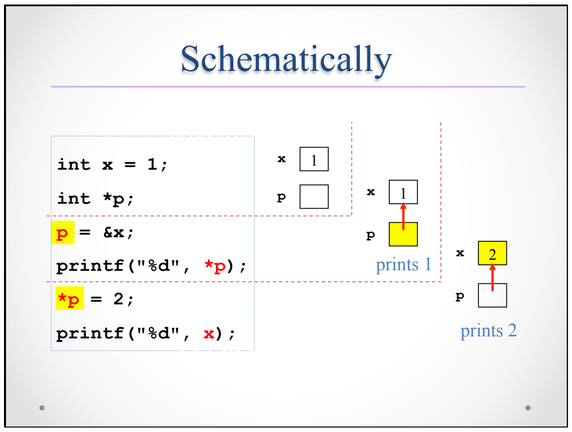
    `*p` refers to **`x`**; thus prints 1





Assignment Using Indirection Operator

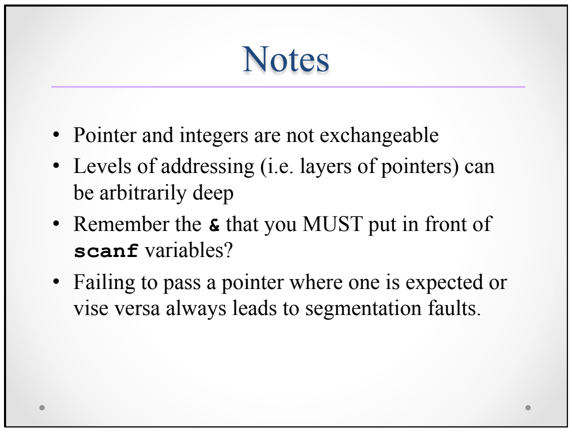
- Allows access to a variable indirectly through a pointer pointed to it.
- Pointers and integers are not interchangeable
- Example
  - `int x = 1, *p;`
    ```
    p = &x;
    *p = 2;
    printf("%d\n", x);
    ```
  - `*p` is equivalent to `x`



Schematically

```
int x = 1;
int *p;
p = &x;
printf("%d", *p);
*p = 2;
printf("%d", x);
```

x
1
p
x
1
p
prints 1
x
2
p
prints 2



Notes

- Pointer and integers are not exchangeable
- Levels of addressing (i.e. layers of pointers) can be arbitrarily deep
- Remember the **&** that you MUST put in front of **`scanf`** variables?
- Failing to pass a pointer where one is expected or vise versa always leads to segmentation faults.