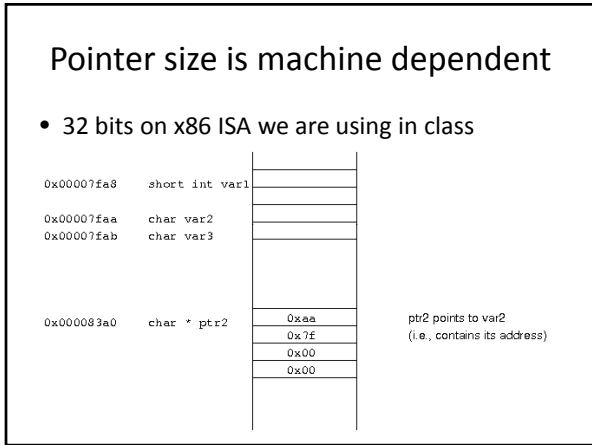


Pointers

CSE 361S

Pointers in C

- A pointer is a variable that contains the address of a variable
- Major uses of pointers:
 - Direct access to specific memory location (address)
 - Manipulation of array elements, especially multi-dimensional arrays
 - Program management of dynamically allocated memory
 - Passing arguments by reference
- Indirect addressing modes in assembly language represent exactly the same concept.



Pointer Declaration

[type] * [name] {= [init val]};

- e.g.,


```
char c = 'A';
char *p = &c;
```
- & is operator that returns address of a variable
- e.g.,
 - &c gives address of variable c
 - &p gives address of variable p

Pointer Use

* is the "indirection" or "dereference" operator, when applied to a pointer it accesses the memory at the address stored in the pointer variable.

* operator can exist on either side of assignment:

```
char c = 'A';
char *p = &c;
char fred;
fred = *p;
*p = 'B';
```

*char * p*

fred == 'A'

c == 'B'

Array Names and Pointers

- Note: name of array is equivalent to pointer to first element
- e.g.,


```
int z[10];
```

~~*z = &z*~~

**z = z*

z is the same thing as &z[0]

char f(3) = "AB";

*char *g;*

Example Use

```
int x = 1;
int y = 2;
int z[10];
int *ip;
int *iq;
```

→ time

addr.	var.	value								
2000	x	1	0							
2004	y	2	1							
2008	z		13	23	22	23				
200c						0				
2010										
2014										
2018										
201c							7			
2020										
2024										
2028										
202c										
2030	ip	2008	200f							
2034	iq		200f	200c	201c	203c				

```
ip = &x; /* ip points to x - i.e., contains address of x */
y = *ip; /* *ip accesses contents of x, now y == 1 */
*ip = 0; /* sets x to 0 */
ip = &z[0]; /* ip now points to z's first element,
            equivalent to ip = z */
*ip = 13; /* equiv. to z[0] = 13 */
*ip = *ip + 10; /* adds ten to z[0], now equal 23 */
*ip -= 1; /* subtract 1 from z[0], now equal 22,
           equiv. to (*ip)-- and --(*ip), but
           not equiv. to *ip-- */
```

```
iq = ip; /* iq now also points to z[0] */
(*iq)++; /* adds 1 to z[0], now equal 23 */
iq++; /* iq now points to z[1], independent of
       size of elements of z */
*iq = 0; /* sets z[1] to zero */
iq = iq + 4; /* iq now points to z[5] */
*iq = 7; /* equiv. to z[5] = 7 */
iq = iq + 8; /* iq now points to ? (z[13]?) */
```

Physical Address Example

```
unsigned int * status;

status = (unsigned int *)0x8000040c;
for (i=0; i<24; i++) {
    mask = 1 << i;
    status_bit[i] = *status & mask;
    status_bit[i] >>= i;
}
```

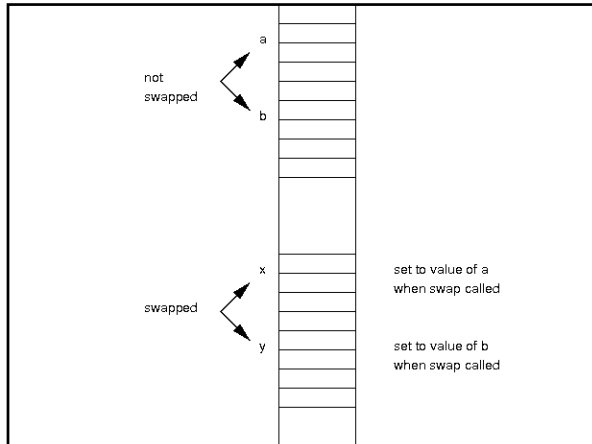
Pointers and Function Arguments

```
void swap (int x, int y) {
    int temp;

    temp = x;
    x = y;
    y = temp;
}

swap (a, b);
```

- Note: C passes parameters a and b by "value", so above call to swap does not change a and b, only copies of a and b.



Working Pass by Reference

```

void swap (int * px, int *py) {
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;
}

swap (&a, &b);
    
```

Handwritten notes in red:

- int x, y
- x = *px;
- y = *py;
- *px = x;
- *py = y;

Handwritten note in red: -Wall

