

# Pointers

Note Title

10/8/2007

A pointer is a variable that contains the addr. of some data object.

Major Uses:

- direct access to specific mem location
- manipulation of array elements
- prog. management of dynamically alloc. mem.
- passing arguments by reference

In assembly lang., indirect addressing mode represents exactly same concept

Size of pointer is machine dependent  
(32 bits on x86 ISA used in class)

short int var1

char var2

char var3

0x7fa8

0x7faa

0x7fab

⋮

char \* ptr2

0xaa

0x7f

0x00

0x00

← ptr2 points  
to var2

declaring pointers

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

e.g.,

`char c = 'A';`

`char *p = &c;`

`&` is operator that returns address of variable

`&c`    addr of `c`

`&p`    addr of `p`

\* is the "indirection" or "dereference" operator. When applied to a pointer it accesses the "pointed-to" memory

\* can go on either side of assignment

Note: name of array is equivalent to pointer to first element

`int z[10];`      `z ≡ &z[0]`

```
int x = 1;
```

```
int y = 2;
```

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

```
int *ip;
```

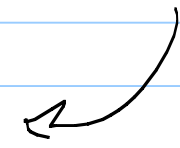
```
int *iq;
```

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	2000	2008							
2034	ip					2008	200c	201c	203c	

Z[0]  
[1]  
[2]

[3]

Z[9]



$ip = \&x;$	$\/* ip points to x \*/$
$y = *ip;$	$\/* *ip accesses contents of x \*/$
$*ip = 0;$	$\/* sets x to 0 \*/$
$ip = \&z[0];$	$\/* ip now points to firstelement of z \*/$
$*ip = 13;$	$\/* z[0] = 13 \*/$
$*ip = *ip + 10$	$\/* adds 10 to z[0] \*/$
$*ip -= 1;$	$\/* equiv. to (*ip)--and --(*ip), butnot *ip-- \*/$
$iq = ip;$	

$(\ast iq) ++ ;$   $\/* z[0] \text{ now } z_3 \*/$

$iq ++ ;$   $\/* iq \text{ now points to } z[1] \*/$

$\ast iq = 0 ;$   $\/* z[1] \leftarrow 0 \*/$

$iq = iq + 4 ;$   $\/* iq \text{ now points to } z[5] \*/$

$\ast iq = 7 ;$

$iq = iq + 8 ;$   $\/* iq \text{ points to } z[13]? \*/$

$\ast iq = \text{anything}$   $\/* \text{BAD, BAD, BAD!} \*/$

physical addr example

```
unsigned int * status;
```

```
status = (unsigned int *) 0x8000040c;
```

```
for (i=0; i<24; i++) {
```

```
    mask = 1 << i;  
    test_bit[i] = (*status) & mask;
```

```
}
```

## function arguments

```
void swap (int x, int y)
    int temp;
    temp = x;
    x = y;
    y = temp;
}
swap (a, b);
```

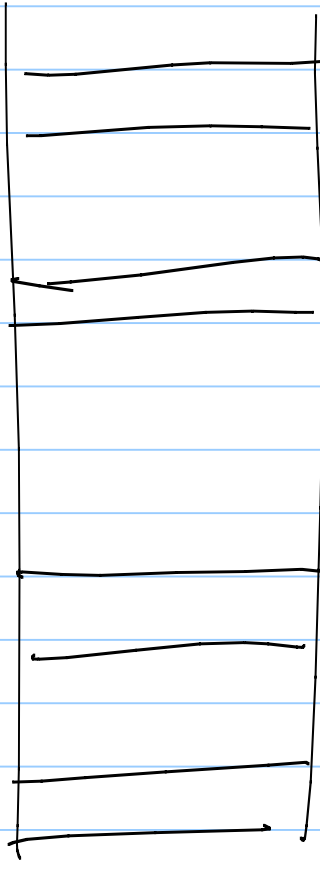
C passes parameters a, b by value

not swapped



a

b



swapped



x

y

set to value of a  
when swap called

set to value of b  
when swap called

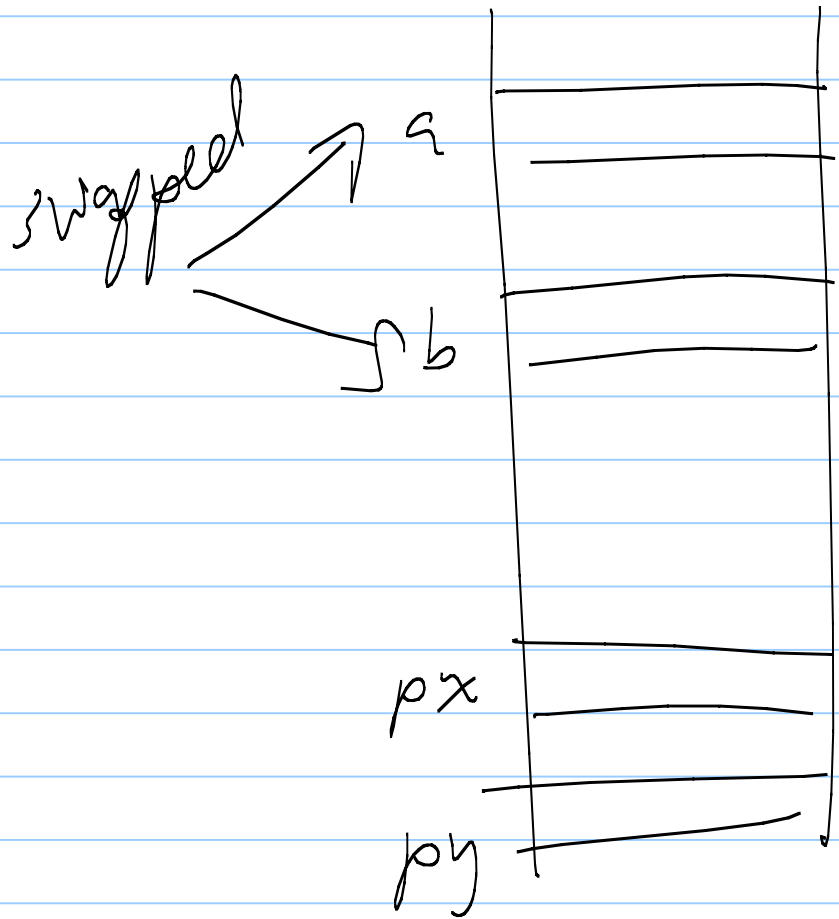
```
void swap (int *px, int *py)  
    int temp;
```

```
    temp = *px;
```

```
    *px = *py;
```

```
    *py = temp;  
}
```

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



set to addr of a when swap called  
 " " " b " " "

The call is by reference

command line arguments in C

```
int main (int argc, char * argv[])
```

argc is Number of arguments (incl. prog. name)

argv is array of strings

argv[0] : program name

argv[1] : 1st argument

argv[argc-1] : last arg.

```
int main (int argc, char * argv[]) {
    int i;
    printf ("%d args\n", argc);
    for (i=0; i < argc; i++) {
        printf ("%d: %s\n", i, argv[i]);
    }
    return 0;
}
```

echoargs.c

> ./echoargs Hi there

3 args

0: ./echoargs

1: Hi

2: there

>

> gcc echoargs.c -o echoargs

