CS 201
Introduction to Pointers

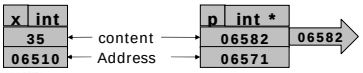
Debzani Deb

Incoming Exam

- Next Monday (5th March), we will have Exam #1.
- Closed book.
- Sit with an empty space in either side of you.
- Calculators that have text-allowing memory is not permitted.
- Covers Chapter 1-6 (Lecture Materials, Readings, Quiz, Linux basics, Makefile, Eclipse, Pointer & Labs).
- You should be able to answer correctly 80-85% of the exam, if you get hold of the above materials. The rest 15-20% will be challenging.
- Email me/TA any question you have or see us at our office times. Don't just wait for the last minute.

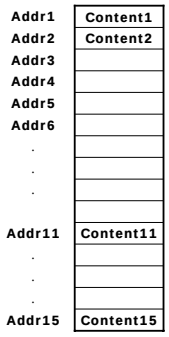
What is a Pointer?

- A pointer is a variable which points to a place in computer's memory.
  - It contains memory address as its value (POINTER = ADDRESS)
  - Pointer variable: a variable dedicated to hold memory addresses.
- A variable name directly references a value (i.e. content of a memory cell).
- A pointer indirectly references a value. Referencing a value through a pointer is called indirection.



Pointers and Addresses

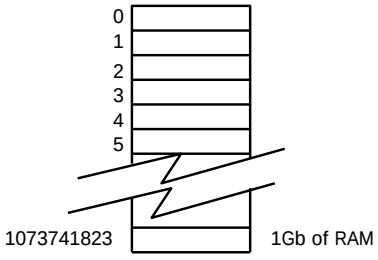
- Memory can be conceptualized as a linear sequence of addressable locations.
- Variable reference the content of a location.
- Pointers have a value of the address of a location.



Size of Variable Types

- Everything in a computer program must fit into high speed memory.
- High speed memory is simply a string of bytes. Lots and Lots and Lots of bytes, but just that…bytes.
- Some number of bytes are allocated every time you define a variable.

High Speed Memory (RAM)



So What Is The Size Of Different Types?

If you ran a short program with a bunch of these statements.

```
printf("sizeof(int) = %d\n", sizeof(int));
```

- sizeof(int) = 4 sizeof(long) = 4
- sizeof(long int) = 4 sizeof(long long int) = 8
- sizeof(char) = 1 sizeof(float) = 4
- sizeof(double) = 8 sizeof(long double) = 12

So What Is The Size Of Different Types?

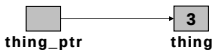
Depends on hardware, brand of OS and compiler.

Here is Visual Studio .Net on Windows Xp Professional. (Differences are marked in red.)

- sizeof(int) = 4 sizeof(long) = 4
- sizeof(long int) = 4 (long long illegal on WinXp C++.Net)
- sizeof(char) = 1 sizeof(float) = 4
- sizeof(double) = 8 sizeof(long double) = 8
- Now, on to pointers!

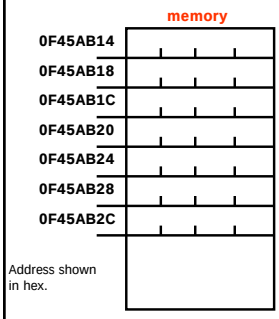
Again What is Pointer?

- A variable containing the address of another variable. (Hence it "points" to the other location).
- There are "things" and "pointers to things"



Reference	Explanation	Value
thing	Direct value of the variable thing	3
thing_ptr	Direct value of variable thing_ptr	Pointer to location containing 3
*thing_ptr	Indirect value of thing_ptr	3

Structure Of A Pointer

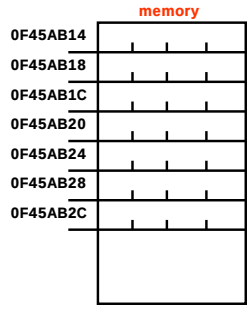


int x=2,y=4,z=6;

name	address

Symbol Table

Structure Of A Pointer

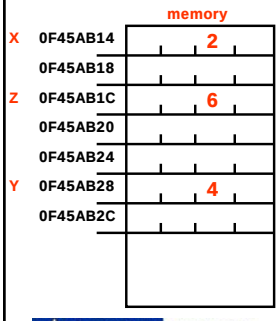


int x=2,y=4,z=6;

name	address
X	
Y	
Z	

Symbol Table

Structure Of A Pointer



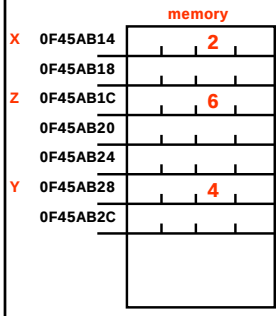
int x=2,y=4,z=6;

Note, I have shown the value of the variable in memory, not the actual representation!

name	address
X	0F45AB14
Y	0F45AB28
Z	0F45AB1C

Symbol Table

Structure Of A Pointer

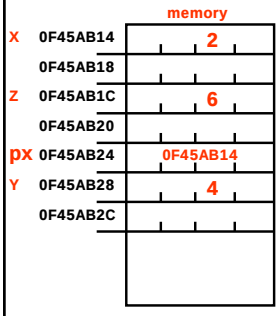



```
int x=2,y=4,z=6;
int * px;
px = &x;
```

name	address
X	0F45AB14
Y	0F45AB28
Z	0F45AB1C

Symbol Table

Structure Of A Pointer




```
int x=2,y=4,z=6;
int * px;
px = &x;
```

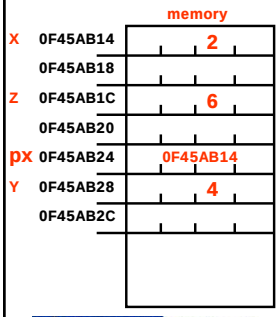
name	address
X	0F45AB14
Y	0F45AB28
Z	0F45AB1C
px	0F45AB24

Symbol Table

So, what is the size of a pointer?

- sizeof(int *) = 4
- sizeof(long long int *) = 4
- sizeof(float *) = 4
- sizeof(double *) = 4
- sizeof (char *) = 4
- Pointers to different types are all the same size.
- Once again, their size may be dependent on your hardware, OS, or even compiler.
- Now, back to my diagram.

Structure Of A Pointer

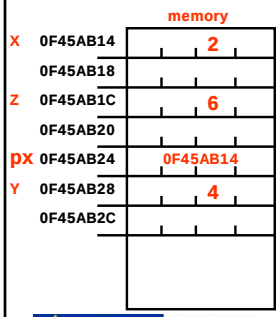



```
int x=2,y=4,z=6;
int * px;
px = &x;
```

name	address
X	0F45AB14
Y	0F45AB28
Z	0F45AB1C
px	0F45AB24

Symbol Table

Structure Of A Pointer

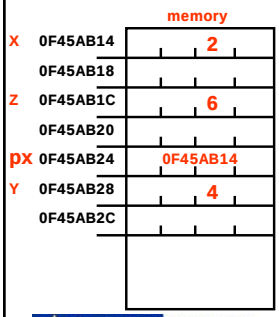



```
int x=2,y=4,z=6;
int * px;
px = &x;
printf("%d\n",x);
```

name	address
X	0F45AB14
Y	0F45AB28
Z	0F45AB1C
px	0F45AB24

Symbol Table

Structure Of A Pointer

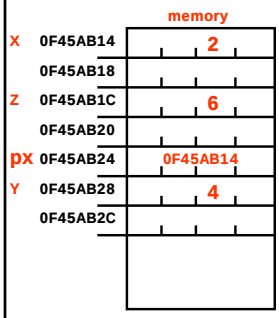



```
int x=2,y=4,z=6;
int * px;
px = &x;
printf("%d\n",x);  2
```

name	address
X	0F45AB14
Y	0F45AB28
Z	0F45AB1C
px	0F45AB24

Symbol Table

Structure Of A Pointer

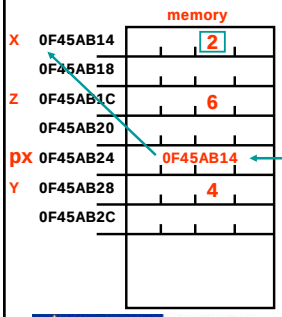



```
int x=2,y=4,z=6;
int * px;
px = &x;
printf("%d\n",x);   2
printf("%d\n",*px);
```

X	0F45AB14
Y	0F45AB28
Z	0F45AB1C
px	0F45AB24

Symbol Table

Structure Of A Pointer




```
int x,y,z;
int * px = &x;
printf("%d\n",x);   2
printf("%d\n",*px); 2
```

X	0F45AB14
Y	0F45AB28
Z	0F45AB1C
px	0F45AB24

Symbol Table

Pointer Declaration

int * p; // pointer is declared

p is a pointer pointing at an variable of type int.

- * tells the compiler that the variable named p is to be a pointer.
- The name of the pointer variable is p (not *p).
- The size of the memory addressed by a pointer depends on the type of the object mentioned in declaration.

```
int * p;      // p points at 4 consecutive bytes
double * p;   // p points at 8 consecutive bytes
```


Operators Related To Pointers

- & address-of
  - printf("%p\n",&variable);
- * dereference
  - printf("%d\n", * pointer_variable)

Address of Operator (&)

```
int x = 3;   // Variable defined
int * p;     // Pointer declared
p = &x;      // Pointer p is set up to point at
             // the variable x.
```

Address of variable x is assigned to pointer p.

- & is an unary address operator which returns the memory address of variable.
- &x is read as "address of x"

About Asterisk (*)

- In declaration `int * p`
  - * tells the compiler that a new variable is to be a pointer (read as "pointer to")
- In Regular use
  - It provides the content of the memory location specified by a pointer. It mean "follow the pointer". `x = * p`
  - It can also stand on the left side of an assignment.

```
* p = 10
```

Typical Uses Of Operators

- int thing; /* declare an integer (a thing) */
- thing = 4;
- int * thing_ptr; /* declare a pointer to an integer */
  - Don't need _ptr, just helps readability.
  - At this moment, it is pointing to a random values since it has not been initialized yet. Such garbage can be harmful.
- thing_ptr = NULL;
  - thing_ptr is a null pointer means "pointing to nothing".
- thing_ptr = &thing; /* store address in pointer, points to thing */

Cont.

- *thing_ptr = 5; /* set thing to 5 */
  - *thing_ptr is a thing.
  - thing_ptr is a pointer.
- Of course, thing_ptr could be set to point to other variables besides thing.
  - thing_ptr = &something_else;

Illegal and Strange Uses

- *thing is illegal. thing is not a pointer variable.
- &thing_ptr is legal, but strange. "a pointer to a pointer"

Pointing to the Same Thing

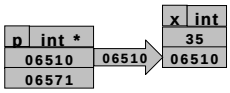
```
int something;
int   *first_ptr;   /* one pointer */
int   *second_ptr; /* another pointer */
something=1;
first_ptr = &something;
second_ptr = first_ptr;
```

```
something=1;
*first_ptr=1;
*second_ptr=1;
```

} All identical results (Only one integer)

Again uses of & and *

p = &x;



// p points at x now.

y = * p;

// y has the value pointed out by the pointer p.

* p= 13;

// 13 was inserted to the

// place pointed out by p.