

Pointers

COMP 1402/1002

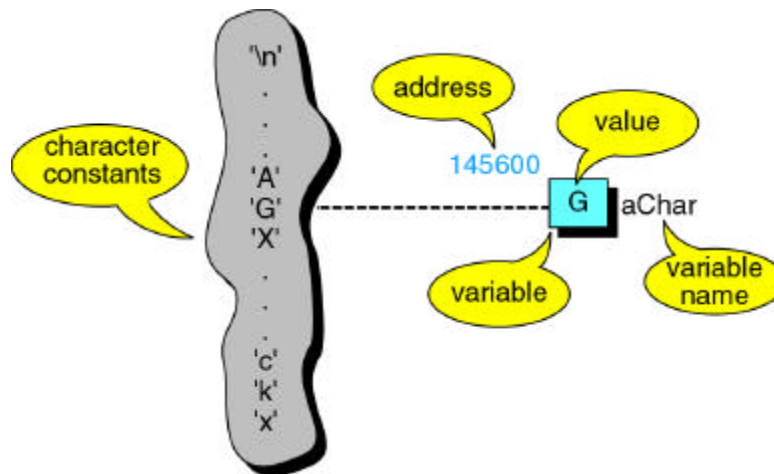
Character Values

- There are a fixed number of **char** values

Examples: 'a' 'b' '\n'

Number of possibilities is the number of
chars

A **char** Variable



Pointer Values

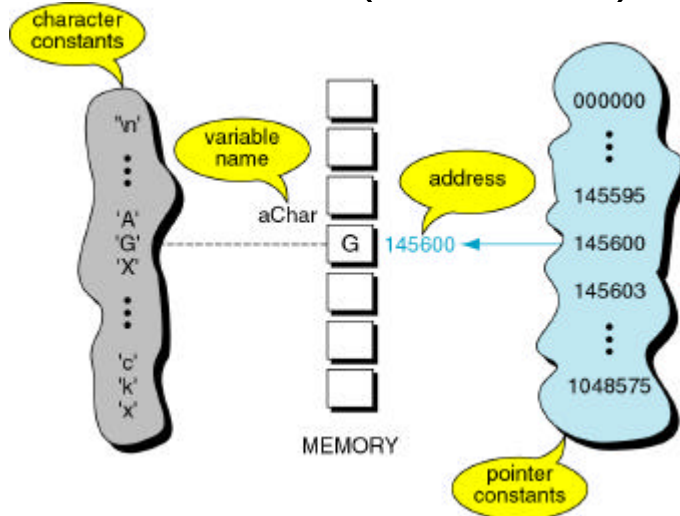
There are a fixed number of pointer values

Assume a 32 bit address space

Examples: 00000000, FACC432F

One different pointer value per address

Pointer Values(Constants)



& operator

& is the address operator

Provides a pointer constant to
any named location in memory

Example: &aChar

Use %p to print an address

Printing Pointers

```
/* Print character addresses */
#include <stdio.h>

int main (void)
{
    /* Local Definitions */
    char a;
    char b;
    /* Statements */
    printf ("%p %p\n", &a, &b);

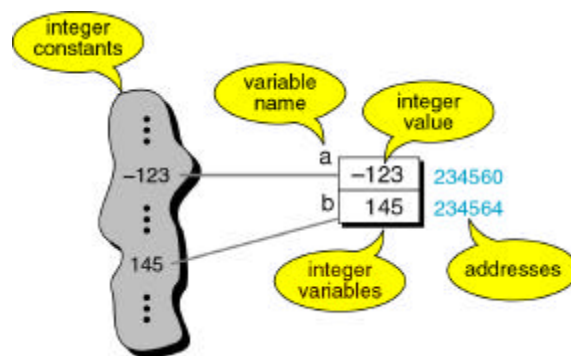
    return 0;
} /* main */
```

a 142300
b 142301

4 byte Integers

Address of a
variable is:

Address of the
first byte of
the variable



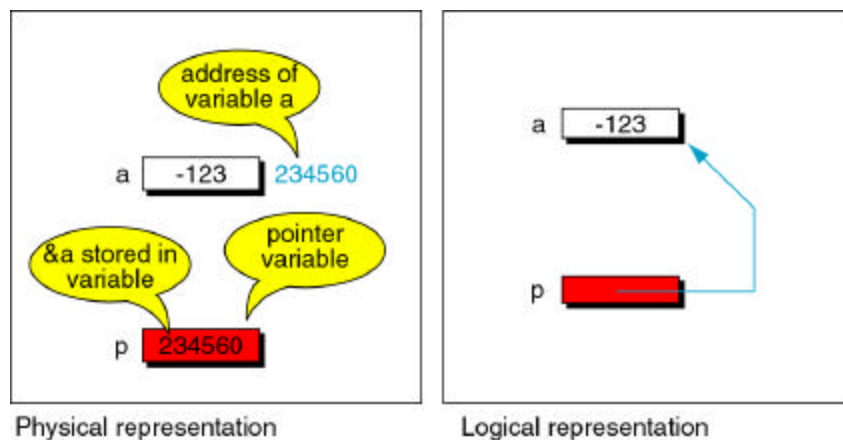
Pointer Variables

Possible to define variables holding pointers

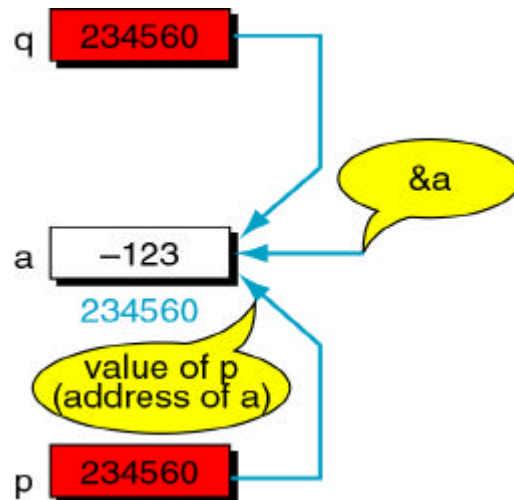
Pointer Variables:

- Contain a pointer constant (an address)
- Can have value changed
- “Point” to a specific type of data
- Two variables can contain same value!

Pointer Variable



Pointer Variables



Accessing the Variables

* Is the Indirection operator

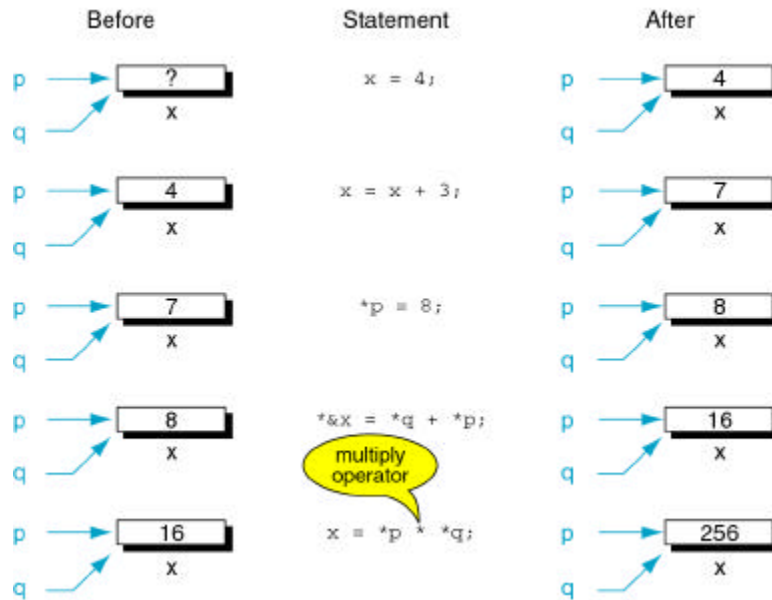
“Dereferencing” often read as “contents of”

p = &a;

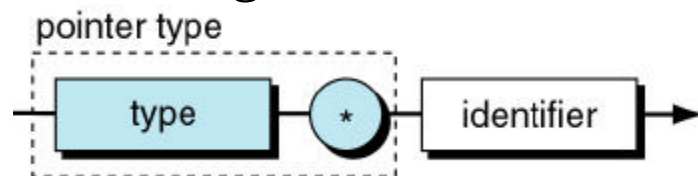
c = *p + 43;

Fig 9-8

Examples



Declaring Pointer Variables



Examples:

`char *p;`

`int *q;`

`float *r;`

`long double *s;`

`long long int *t;`

Uninitialized Pointers

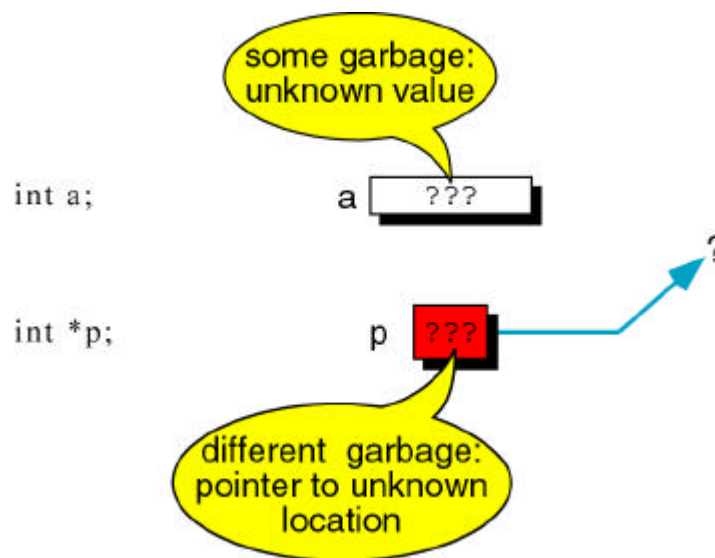
As with all variables in C:

If you don't initialize you get *whatever*

BEWARE:

Uninitialized pointers contain some address

Uninitialized Pointers



Initializing to: nowhere

What if a pointer variable shouldn't point anywhere?

Answer: set it to NULL

```
int *p;  
p = NULL;
```

NULL is defined in stdio.h or stddef.h

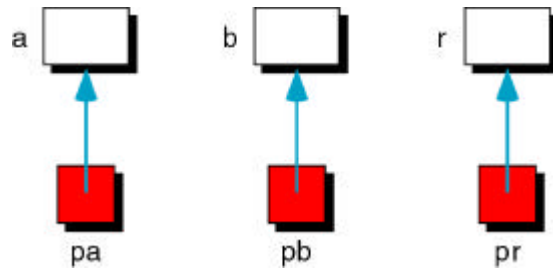
Initialization to Somewhere

```
int a;  
int *p;  
p = &a;
```

Never try to dereference a NULL pointer

Gives run-time error (normally segmentation fault)

Add **a**, **b** using **pa**, **pb**



Work out the code...

Pointers as Parameters

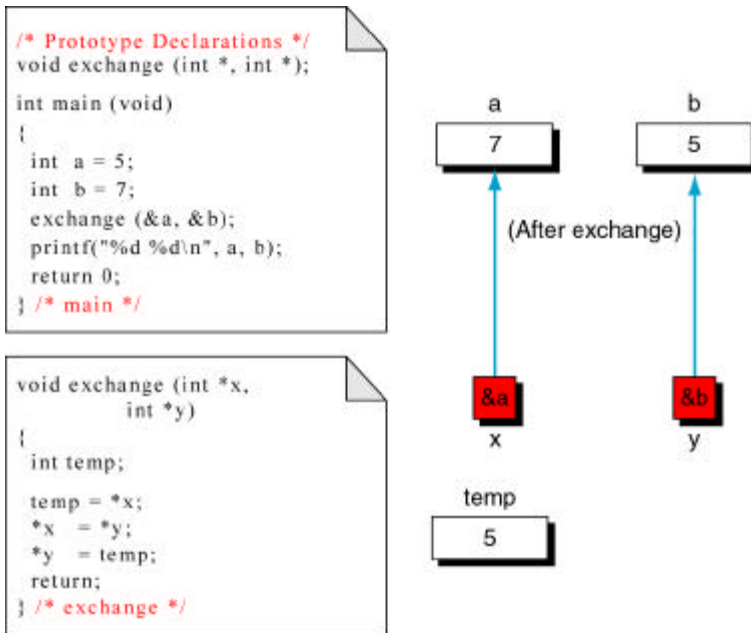
C uses pass by value!!

To modify a passed variable:

- Pass the address

- Modify the contents

That is: pass the pointer!!



Returning Pointer Variables

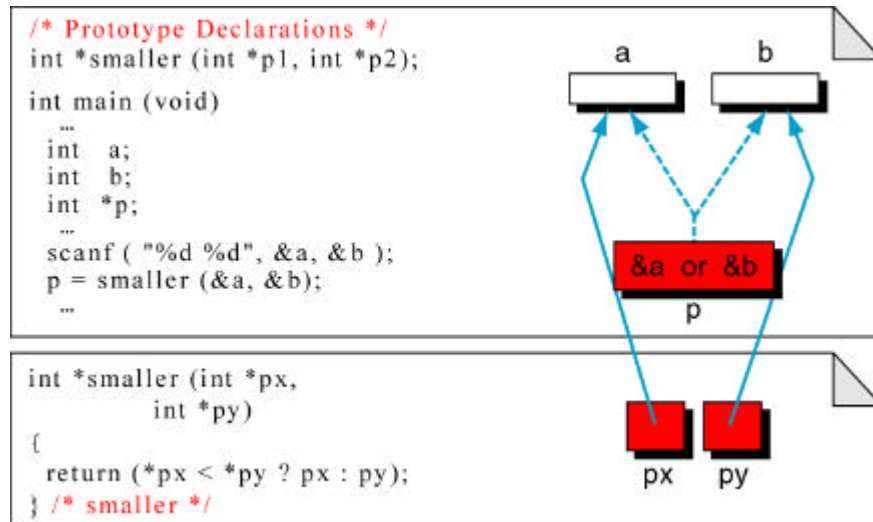
Suppose we want to return a pointer?

Then declare the return type to be a pointer

Example:

```
int *smaller (int *p1, int *p2);
```

Returning Pointer Variables



A Serious Error

Never return a pointer to a local variable

```
float *mistake() {
    float temp = 12;
    return &temp;
}
```

Points to space on stack that isn't used!!

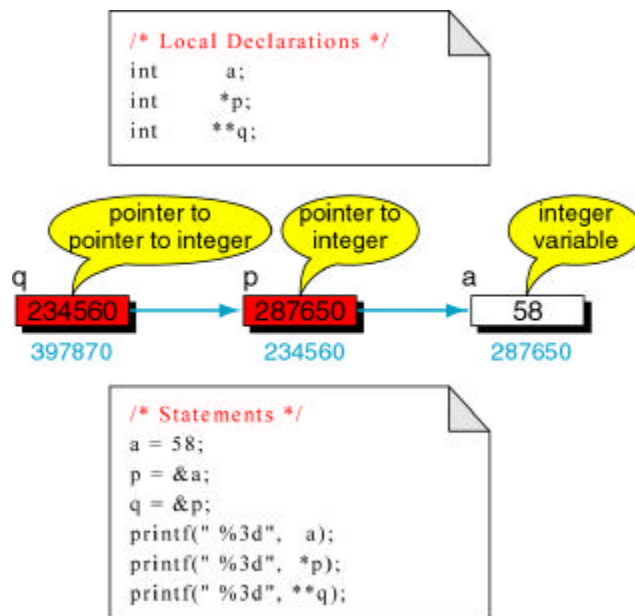
Pointers to pointers

If a pointer variable “points” somewhere:
Why not to another pointer variable??

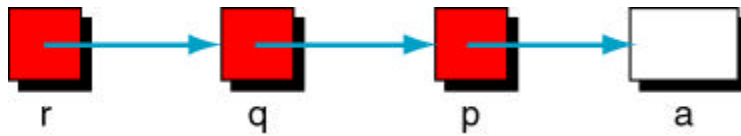
Ans. Of course you can do it.

```
int **p;
```

Pointers
to
Pointers



Using Pointer to Pointers...



Declare code for **r**, **q**, **p**, **a** where **a** is a float

Use scanf with each of these variables

Sizes and Compatibility

```
char c;  
char *pc;  
int a;  
int *pa;  
double x;  
double *x;
```

Print

```
sizeof(c), sizeof(pc), sizeof(*pc)...
```

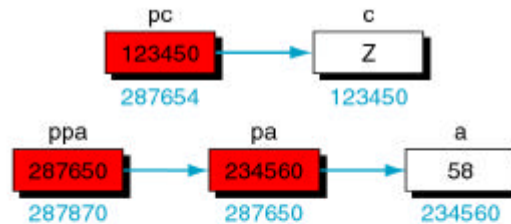
Size and Compatibility

sizeof(c)= 1 sizeof(pc)= 4 sizeof(*pc)= 1
sizeof(a)= 4 sizeof(pa)= 4 sizeof(*pa)= 4
sizeof(x)= 8 sizeof(px)= 4 sizeof(*px)= 8

Pointers are all the same size

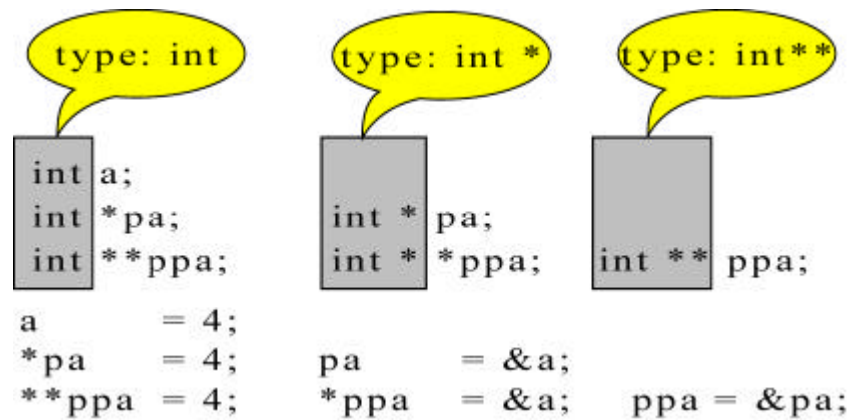
The contents of a pointer know their size & type

Types & Validity



```
char c;  
char *pc;  
  
int a;  
int *pa;  
int **ppa;  
  
pc = &c;          /* Good and valid */  
pa = &a;          /* Good and valid */  
ppa = &pa;       /* Good and valid */  
  
/* Invalid pointers will generate errors */  
pc = &a;          /* Different types */  
ppa = &a;         /* Different levels */
```

Type and Validity



Using Expressions

Expressions evaluate to a value

Expressions are classified by:

How they can be used

This results in **lvalue** and **rvalue** expressions

lvalue and rvalue

An **lvalue** expression is to be used whenever it is receiving a value

An **rvalue** expression is to be used to supply a value for further use

The 7 lvalues (blue not covered yet)

	Expression Type	Comments
1	identifier	Variable identifier
2	expression[]	Array indexing
3	(expression)	Expression must lvalue
4	*expression	Dereference expression
5	expression.name	Structure selection
6	expression->name	Structure indirect selection
7	function call	If returning an address

lvalue expressions

a = ...

a[5] = ...

(a) = ...

***p** = ...

All non-lvalue expressions are rvalue
expressions

Some Operators Require **lvalues**

Type of Operator	Examples
Address Operator	&score
Postfix increment / decrement	x++ y--
Prefix increment / decrement	++x --y
Assignment (left operand)	x= 1 y +=4

Mistakes

The following are rvalue expressions where there should be an lvalue.

Expression	Problem
a+2 = 6;	A+2 is an rvalue can't be modified
&(a+2);	A+2 is rvalue address requires lvalue
&4;	Same as above
(a+2)++;	Postfix inc. needs an lvalue
++(4-b);	Prefix inc. needs an lvalue

Getting multiple values

Definition:

```
/* returns an error code */
int secToHours(long time, int *hours,
               int *minutes, int *secs) {
    /* what is this code? */
}
```

Calling code:

```
secToHours(totalTime, &hrs, &min, &s);
```