COP 3223: C Programming
Spring 2009

Pointers In C – Part 1

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop3223/spr2009/section1

School of Electrical Engineering and Computer Science
University of Central Florida

Pointers In C – More Details

- We've been using pointers in C for some time now. Recall that every array is referenced indirectly through a pointer to its first element. We've dealt extensively with pointers to strings (arrays of characters). Program 5 utilized a fair amount of explicit pointers being passed to functions.
- Now we want to take a closer look at pointer operations in general.
- In C, a pointer can be declared to any data type as well as any structure (we'll see these soon).
- Most modern computers are said to be byte addressable, which means that every byte in the computer's memory has a specific address. If the computer has `n` bytes in its memory, then the addresses of those bytes range from `0` to `n-1`.

Pointers In C – More Details

- Every C program that you write contains both object code (machine instructions which correspond to the statements in the original source code) and data (the variables in the original source program).
- Each variable in the source program occupies one or more bytes of memory, depending on its data type.
  - Recall that the `char` type requires 1 byte, the `int` type requires 2 bytes, and the `long` type requires 4 bytes.
- For every variable, no matter its type, the address of the first byte it occupies in memory is said to be the address of the variable.

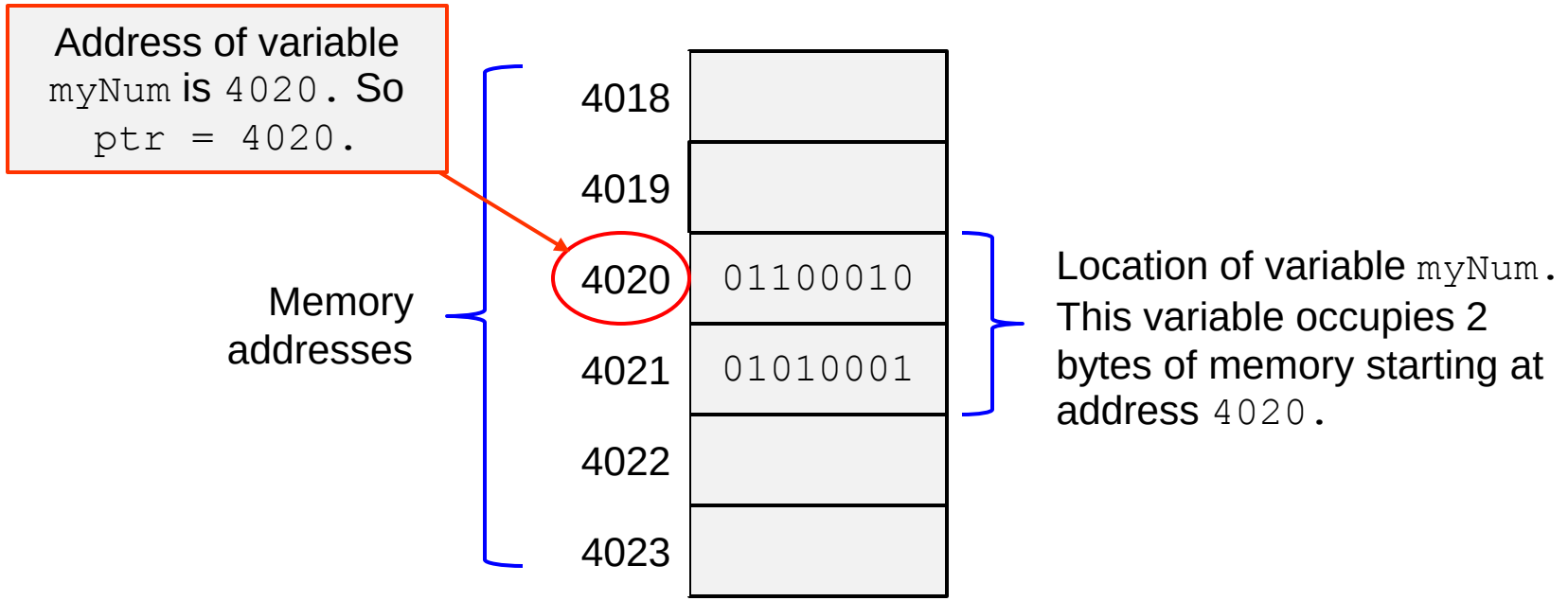
Pointers In C – More Details

- For example, suppose we have `int myNum = 25169;`

```
int *ptr;
*ptr = &myNum;
```

The decimal value `25169` represented in binary is: `110001001010001`

If we look at part of the memory of our computer and supposing that `myNum` is stored at address `4020`, then we have the following configuration:



Pointers In C – More Details

- The type of object that a pointer references (points to) in C is called the referenced type. C places no basic restrictions on the referenced type. It is even possible to have a pointer point to another pointer (this would add a second level of indirection).
- C provides two operators specifically designed for use with pointers:
- The address operator (&), is used to find the address of a variable. If `x` is a variable, then `&x` is the address of `x` in memory.
- The indirection operator (*), is used to gain access to the object that a pointer references. If `p` is a pointer variable, then `*p` represents the object to which `p` currently points.

Pointers In C – More Details

- Recall that declaring a pointer variable does not cause it to refer to any specific location in memory.

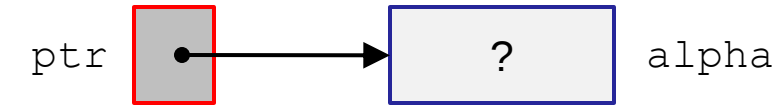
```
int *ptr;  //ptr can reference an integer
           //value but currently points nowhere
```

- It is crucial to initialize any pointer variable before it is used. A common technique is to simply assign it the address of some variable using the address operator (&).

```
int alpha, *ptr;
ptr = &alpha; //assigns address of alpha
              //to ptr
```

This two line declaration and assignment can be combined into the single line:

```
int alpha, *ptr = &alpha;
```



Pointers In C – More Details

- Once a pointer variable is referencing some object, the indirection operator (*) can be used to access what is stored in the object.
- If we again have:

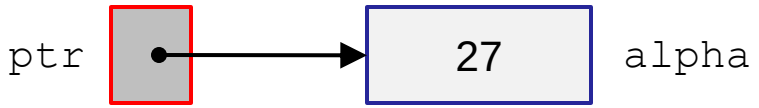
```
int alpha, ptr = &alpha;
```

and we make the following assignment:

```
alpha = 27;
```

then we can print the value stored in the variable `alpha` using the pointer variable `ptr` as follows: `printf("%d\n", *ptr);`

The output would be: `27`



Pointers In C – More Details

- If it helps you to understand the concept better, you can think of the indirection operator as the inverse of the address operator.
- Applying & to a variable produces a pointer to the variable; applying * to the pointer takes you back to the original variable.

```
j = *&i;  //same as j = i
```

- As long as a pointer references some object, that pointer is said to be an alias for the object. In other words, the object can be referred to in two different ways, either through the variable name or through the pointer referencing the object.

example 1.c

```c
//Pointers In C - Part 1 - example 1 - illustration of pointers
//April 6, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
     int alpha;  //a normal scalar integer value
     int *ptr;   //a pointer to an integer value
     int **ptrToPtr; //a pointer to a pointer to an integer

     alpha = 6;
     ptr = &alpha;
     ptrToPtr = &ptr;

     printf("The value of alpha is %d\n", alpha);
     printf("The address of alpha is %ld\n", &alpha);
     printf("The address of alpha in hex is %p\n\n", &alpha);
     printf("The value of ptr is %ld\n", ptr);
     printf("The address of ptr is %ld\n", &ptr);
     printf("The value referenced by ptr is %d\n\n", *ptr);
     printf("The value of ptrToPtr is %ld\n", ptrToPtr);
     printf("The address of ptrToPtr is %ld\n", &ptrToPtr);
     printf("The value referenced by ptrToPtr is %d\n", *ptrToPtr);
     printf("The value referenced by the address");
          printf(" referenced by ptrToPtr is %d\n\n", **ptrToPtr);
     printf("The value of *&ptr is %d\n", **&ptr);
     printf("\n\n");
     system("PAUSE");
     return 0;
}//end main
```


Pointers In C – More Details

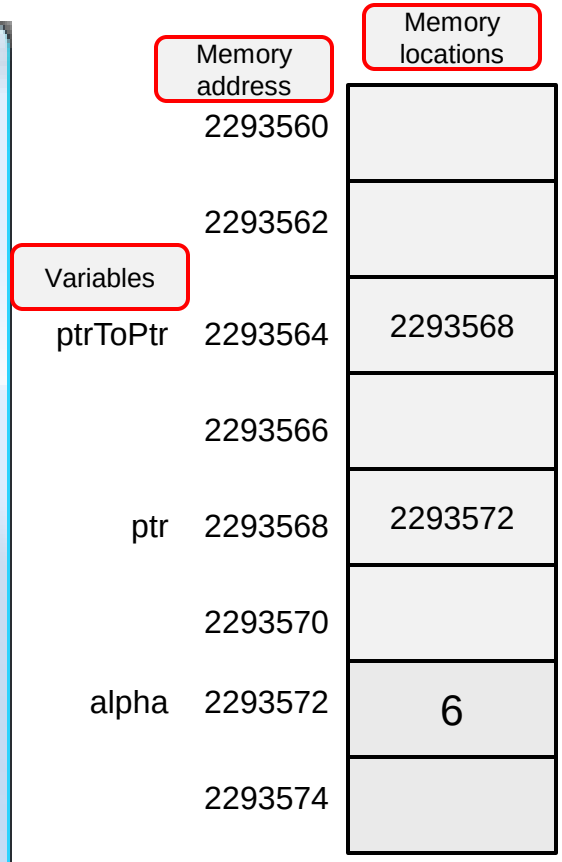
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files...

```
The value of alpha is 6
The address of alpha is 2293572
The address of alpha in hex is 0022FF44

The value of ptr is 2293572
The address of ptr is 2293568
The value referenced by ptr is 6

The value of ptrToPtr is 2293568
The address of ptrToPtr is 2293564
The value referenced by ptrToPtr is 2293572
The value referenced by the address referenced by ptrToPtr is 6

The value of *&ptr is 6
```



Representation of the memory configuration of the example

Pointers In C – More Details

- C allows the use of the assignment operator to copy pointer values, provided that they reference the same type of object.
- If we have:

```
i, j, *ptr1, *ptr2;
```

then the statement

`ptr1 = &i;` is an example of pointer assignment; the address of `i` is copied into `ptr1`.

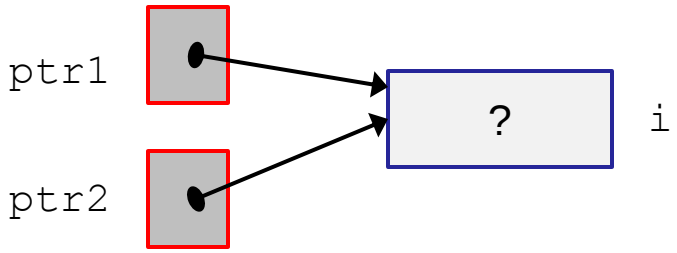
Here is another example of pointer assignment:

```
ptr2 = ptr1;
```

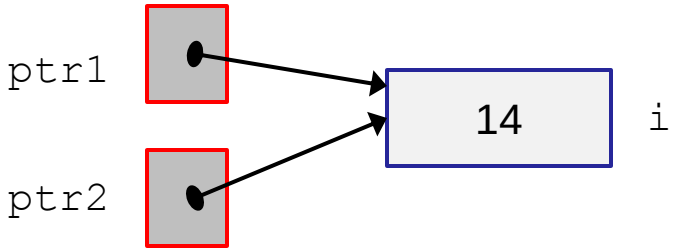
This statement copies the contents of `ptr1` (the address of `i`) into `ptr2`. The effect of this is to make `ptr1` and `ptr2` reference the same object, in this case the address containing the value of variable `i`.

Pointers In C – More Details

- Continuing with the example from the previous page, we how have the following situation:

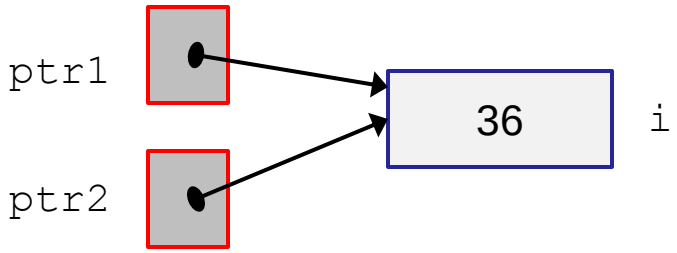


- Both `ptr1` and `ptr2` reference the address of variable `i`, so we can change the value of `i` by assigning a new value through either `ptr1` or `ptr2` (or `i`).
- If, for example, we do: `*ptr1 = 14;`, then we'll have the following situation:



Pointers In C – More Details

- Continuing with the example from the previous page, if, for example, we do: `*ptr2 = 36;`, then we'll have the following situation:



- All of the following statements will print the same result, namely 36.

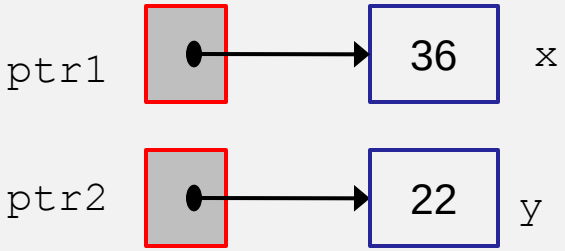
```
printf("The value referenced by ptr1 is %d\n", *ptr1);
printf("The value referenced by ptr2 is %d\n", *ptr2);
printf("The value of i is %d\n", i);
printf("The value of **&ptr1 is %d\n", **&ptr1);
```

i

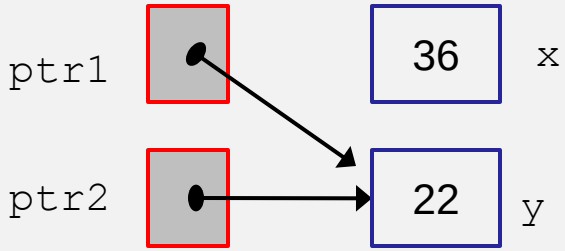
* * CAUTION * * *

Be careful not to confuse statements like `ptr1 = ptr2` with `*ptr1 = *ptr2`.

The first statement is a pointer assignment, it assigns the value (an address) of `ptr2` to the value (an address) of `ptr1`.

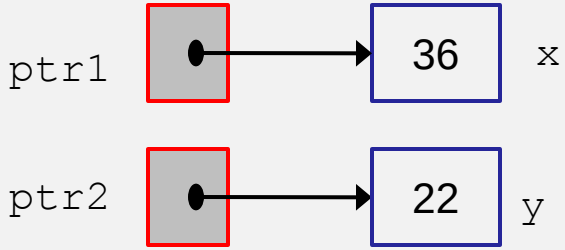


Before statement: `ptr1 = ptr2`

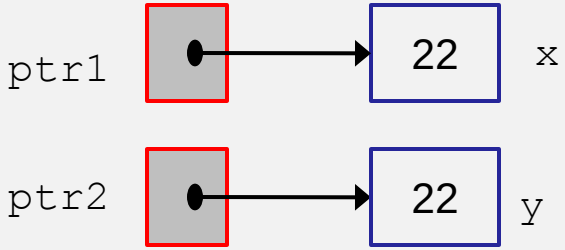


After statement: `ptr1 = ptr2`

The second statement above, is a normal assignment operation which sets the value of the object referenced by `ptr1` to have the same value as the object referenced by `ptr2`.



Before statement: `*ptr1 = *ptr2`



After statement: `*ptr1 = *ptr2`

Practice Problems

1. Write a program that will read a double from the keyboard and then break the number into its integer part and fractional part and store the integer and fractional part in locations referenced by pointers. Then using the pointers will reconstruct and print out the complete value.

   Example: user enters 42.789. Break into 42 and 789 stored in two different locations referenced by pointers. Then read the values at those locations and print the value 42.789.