Class 05 - News

- You should have received HW1 by email. If not check spam or check website: https://cscie26.dce.harvard.edu/~dce-info113/gr/

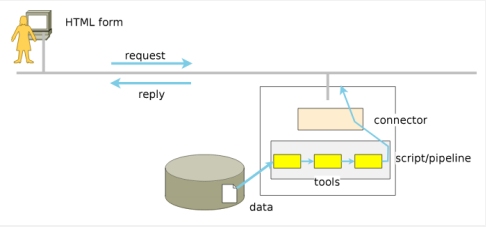
- HW2 due Sunday at 11:59pm ET
  submit from devel dir
- Section: Brandon Thu 7:30-8:30PM
- Off Hrs: Alexis Tue 8-9pm
- Off Hrs: Bruce Sat 2-3:30PM, 7-8PM
  Sun 2-5M
- HW3 is paper and pencil and is due a week from Sunday. Will be posted by Monday morning.
- Midterm meetings next week
- Meet other students in "Study Lounge" on Zoom page

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
/* warm up exercises for class 05 -- addresses and memory */

int my_strlen(char s[]);
void count_upper_case();
void add_to_day();
void string_subtract();
void copy_string();
#define LEN 200

int main()
{
      count_upper_case();
      add_to_day();
      string_subtract();
      copy_string();
}

/* this function is not incorrect, but it is poor design. why? */
void count_upper_case()
{
      int   i, ncaps;
      char line[LEN];

      printf("enter a string of text: ");
      fgets(line, LEN, stdin);

      for( ncaps = 0, i = 0; i < strlen(line) ; i++ )
            if ( isupper(line[i]) )
                  ncaps++;
      printf("that line has %d upper case letters.\n\n", ncaps);
}

/* **** PREDICT : the output of these functions */
void add_to_day()
{
      int   ans = strlen( "Wednesday" + 3 );

      printf("length of \"Wednesday\" + 3 is %d\n\n", ans);
}

void string_subtract()
{
      int   ans = "three" - "one";
      printf("three - one is %d\n\n", ans);
}

void copy_string()
{
      char  a[] = "short";
      char  b[] = "a longer string of chars";

      strcpy(a, b);
      printf("a = \"%s\"\n", a);
      printf("b = \"%s\"\n", b);
}
```

Class 5: Addresses: Arrays, Pointers, Functions

Computers store data in memory. Each piece of data has an address. Tonight we focus on addresses: storing addresses and using addresses in programs.

Recall the big picture:



1) User enters data into an HTML form
2) Browser sends request and data to server
3) Server examines request
4) If request is for a file, server sends back file
5) If request is for a program, server calls the program
6) The program unpacks user request data then runs the requested application.
7) Application combines tools using pipelines and scripts to produce a reply

Many tools are written in C, many store data in memory. Today we look at details of working with memory in C.
Focus: Addresses are just numbers, memory is simple

Warmup Discussion

discussion of strings and addresses

BIG IDEAS:
A string is a sequence of chars in memory
That sequence has a starting address
The value of "abc" is the address of that array

In int t[10];, the value of t is the address of 1st element

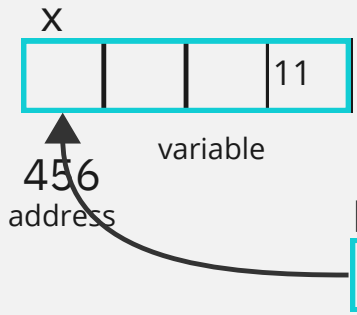
You can do arithmetic with addresses
"hello" + 4 is address of 'o'

C does not check for array over-runs

Programming with Addresses

8:20

```
int x;  // create an int
x = 3;  // store val in x
```



Fact: Every variable has an address: A number. You can store and process addresses!

&x is address of x

pointer : A variable that stores an address

```
int *p;  // create int ptr
p = &x;  // addr of x in p
```

& : "where is" operator

* : "thing pointed to by" operator (dereference)

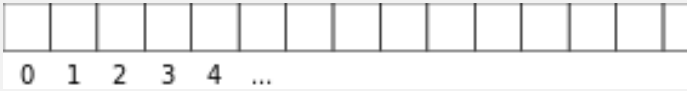
```
*p = 10;  // store 10 in x
(*p)++;   // incr x via p
```

Content vs Address

8:25

Useful for: linked data structures, dynamic memory, pass by reference

Class 5: Addresses: Arrays, Pointers, Functions



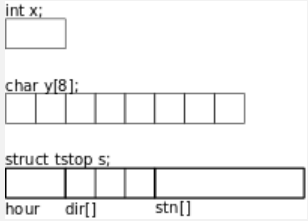
Memory Cells and Addresses

Computer memory is a numbered sequence of char-sized boxes. When a program runs, the code and the data for that program are stored somewhere in memory.

The position of each memory cell is a number: the address of the memory cell. We learn to program with addresses.

What Do We Store in Memory?

1) Single values: char, int, float ...
2) arrays: contiguous sequence of one type
3) struct: varied types in one container
4) Functions: machine language code is stored in memory



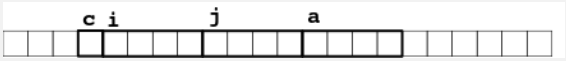
Every variable, of every type, has an address.

We look at programming with addresses for each of these three shapes of storage.

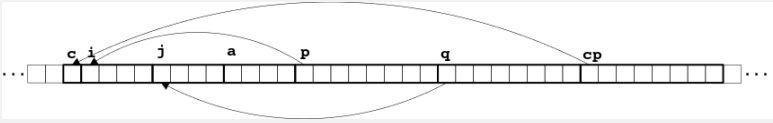
Single Value Variables

I. Simple variables: ex1.c
the compiler assigns memory cells for each variable
The compiler assigns a size and a location: 4 bytes at L

Q1: But WHERE in memory are these variables?
A: We can as C where these are stored by using &varname
ex1pa.c -- print addresses

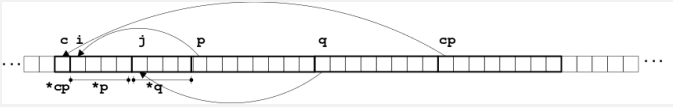


Q2: Can we store these addresses?
A2: y: We need a variable type that can hold an address
The type is a pointer variable and we create them in
ex1sa.c -- store addresses




```
int i;    // create an int
int *p;   // create ptr to int
p = &i;   // store addr of i
```

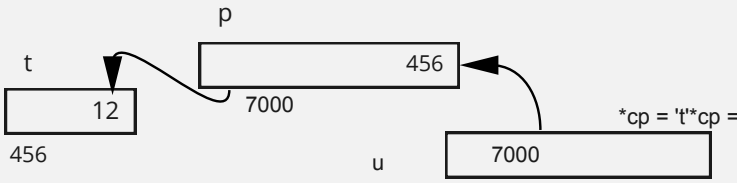
Q3: What else can we do with pointer variables?
A3: We can use the address to get back to the original variable.The term for this is 'dereference'. The operator is "*" applied to a pointer variable..
ex1dp.c -- dereference pointers



Q4: What OTHER operations can we do on pointers to simple vars?
A: compare pointers using ==,. !=, <, >, <=. >=
ex1cp.c - compare pointers

A4b: We can pass addresses to functions; functions can return addrs
ex1pf -- pass to functions

Q5: Can we get the ADDRESS of a POINTER VARIABLE?
A5: What do you think?
What would be the notation for storing a pointer to a pointer?
What does dereferencing do?

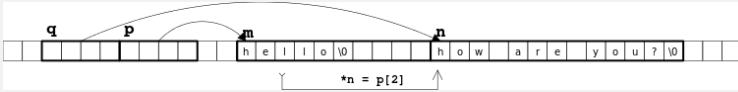



```
int t = 12
int *p ;
p = &t;
int **u;
u = &p;
```

take temp

II. Pointers and Arrays

An array is a contiguous seq of memory cells, all containing values of the same type. Each cell has an address. The array has a starting address. The address of an array is the address of the first element.
ex2.c -- arrays



Q1: How do we use pointers to get other elements in the array?
A1: Use normal indexing (with [ ]). You can ALSO use *
See ex2ia.c [optional: ex2lq.c]

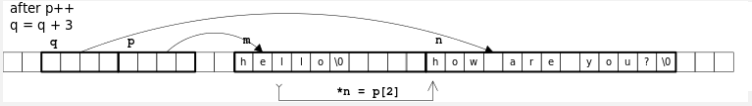
*n = p[2]; // same as n[0] = m[2]; because p is start, [2] is offset

*cp = 't'

FACT: addr[index] MEANS item at [index] spots from addr
FACT: *addr MEANS thing at addr
addr can be an array name or a pointer variable
FACT: base[pos] === *(base + pos)
ex: int t[10]; *(t+3) = 2; is SAME AS t[3] = 2;

Q2: What other operations can we do with pointers to arrays?
A2: assign, compare, increment, decrement, subtract, +/- int

Exercise: predict output of ex2ao.c -- arithmetic operations
Exercise: predict output of ex2ae.c -- arithmetic exercises
[Exercise: use pointers to write strcpy(p, q) or strchr(s,c) -- if lots of time]

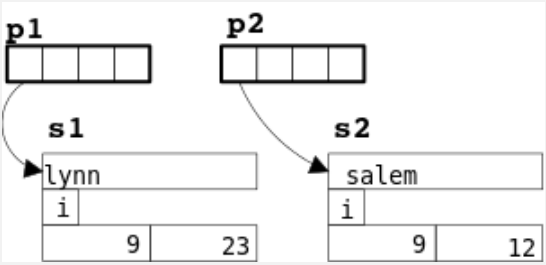


what does this do: p++, q = q + 3

ex2ae.mempic (send to back)

III. Pointers to Structs

A struct is a region of memory holding several members.
A struct has an address.
Use the & operator to get the address of a struct.
Use ptr->membername to select a member using a pointer



Note: We could have drawn p1, p2, s1, and s2 as rectangles in our memory diagram, but this style is also common.

Notation: s.membername when s is a struct
ptr->membername when ptr points to a struct

s1.stn vs p2->stn

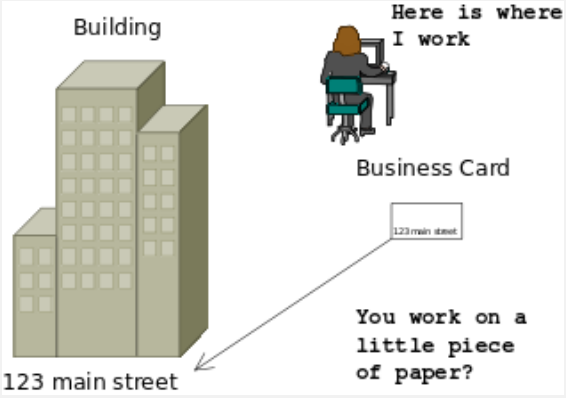
BIG IDEA:

Array: holds many value

Pointer: holds one address

```
char z[12];
```

```
char *w;
```



a 3D array of rooms

- stores things

a pointer

- stores an address

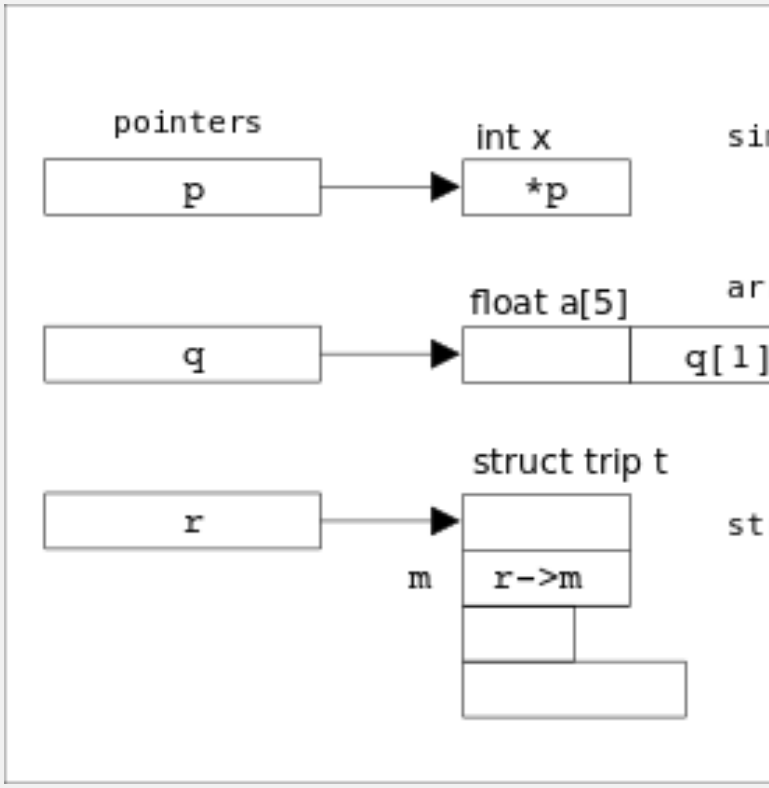
IV. Arrays of Pointers, Pointers to Arrays of Arrays

Draw a picture and trace the code in ex4.c

Important questions:

a. What does char food[4][20] look like? What is stored?

b. What does char *p[4] look like? What is stored?




```
b[i] == *(b+i)
```