ECS 36A, May 26, 2023
Announcements

1. Homework 4 will be out by Monday
2. If you want us to look at something on the midterm, please send the note *through Gradescope*; otherwise it’s very hard to change grades and validate that everything is correct at the end of the quarter
A Quick Review of Pointers

• A pointer is simply an address
  • It’s just like a constant or variable

• A pointer constant cannot be changed
  • int pc [30]; /* here pc is a pointer constant and cannot be changed */

• A pointer variable can be changed
  • int *p; /* here p is a pointer variable and can be changed */
Midterm Question 8

• Setup:
  
  ```
  char a[27] = "abcdefghijklmnopqrstuvwxyz";
  char *b = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  char c;
  ```

• In pictures:

```
                           0          1          24          25
                           'a'           'b'       ...        'Y'           'z'
```

```
        a (100) ->
```

```
                           0          1          24          25
                           'A'           'B'       ...        'Y'           'Z'
```

```
        b (200) ->
```

```
        c (300) ->
```
Midterm Question 8(a)

• Question:
  \[ b = \&c; \]

• Answer: The value stored in \( b \) changes to the address of \( c \)
• Question: 
  \[ a = b; \]

• Answer: \( a \) is a pointer constant, has no storage allocated to it like \( b \) does, and so cannot be changed. So this is an illegal assignment.
Midterm Question 8(c)

• Question:
  \[ c = *(a + 3); \]

• Answer: c becomes what is in the fourth element of a (counting from 0)
Midterm Question 8(d)

- **Question:**
  \[ c = 2[a]; \]

- **Answer:**
  \[ c = 2[a] = *(2+a) = *(a+2) = a[2] = 'c' \]
Midterm Question 8(e)

• Question:
  \[ b = a; \ c = b[25]; \]

• Answer: \[ c = 2[a] = *(2+a) = *(a+2) = a[2] = 'c' \]
Midterm Question 9

```c
int testandinc(int x) {
    return(x++);
}

int p1testandinc(int *x) {
    return(*x++);
}

int p2testandinc(int *x) {
    return((*x)++);
}

int a = 2;
int arr[3] = { 3, 4, 5 };
int *b = arr;
int *c = &arr[1];
d = testandinc(a);
e = p1testandinc(b);
f = p2testandinc(c);
```
Midterm Question 9

```c
int a = 2;
int arr[3] = { 3, 4, 5 };
int *b = arr;
int *c = &arr[1];
d = testandinc(a);
e = p1testandinc(b);
f = p2testandinc(c);
```

![Diagram showing the allocation of variables and their relationships.]

May 26, 2023

ECS 36A, Spring Quarter 2023
Approach

• Go through the program, and then get the values
int testandinc(int x)
{
    return(x++);
}

... 

d = testandinc(a)
Midterm Question 9

int testandinc(int x)
{
    return(x++);
}

... 

d = testandinc(a)

Return value of x
Midterm Question 9

```c
int testandinc(int x)
{
    return(x++);
}
...
d = testandinc(a)

Add 1 to the value of x
```
Midterm Question 9

```c
int testandinc(int x)
{
    return(x++);
}
...
d = testandinc(a)
```

Function ends
Midterm Question 9

```c
int p1testandinc(int *x)
{
    return(*x++);
}

...e = p1testandinc(b)
```

Diagram:

```
a

arr

b
c
d
e

f

x
```

May 26, 2023

ECS 36A, Spring Quarter 2023
Midterm Question 9

```c
int p1testandinc(int *x)
{
    return(*x++);
}
```

```c
e = p1testandinc(b)
```

---

May 26, 2023

ECS 36A, Spring Quarter 2023
Midterm Question 9

```c
int p1testandinc(int *x)
{
    return(*x++);
}
```

```c
e = p1testandinc(b)
```

May 26, 2023
Midterm Question 9

```c
int p1testandinc(int *x)
{
    return(*x++);
}
.
.
e = p1testandinc(b)
```

Function ends
Midterm Question 9

int p2testandinc(int *x)
{
    return((*x)++);
}

f = p2testandinc(c)
Midterm Question 9

```c
int p2testandinc(int *x) {
    return((*x)++);
}

... 

f = p2testandinc(c)
```
Midterm Question 9

```c
int p2testandinc(int *x)
{
    return((*x)++);
}
```

```c
f = p2testandinc(c)
```

May 26, 2023

ECS 36A, Spring Quarter 2023
Midterm Question 9

```c
int p2testandinc(int *x) {
    return(*(x)++);
}

f = p2testandinc(c)
```

May 26, 2023
ECS 36A, Spring Quarter 2023
Midterm Question 9

```c
int p2testandinc(int *x)
{
    return((*x)++);
}
```

```
f = p2testandinc(c)
```

Function ends
### Midterm Question 9 Answers

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>arr or &amp;arr[0]</td>
</tr>
<tr>
<td>c</td>
<td>arr+1 or &amp;arr[1]</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>3</td>
</tr>
<tr>
<td>f</td>
<td>4</td>
</tr>
<tr>
<td>arr[0]</td>
<td>3</td>
</tr>
<tr>
<td>arr[1]</td>
<td>5</td>
</tr>
<tr>
<td>arr[2]</td>
<td>5</td>
</tr>
</tbody>
</table>

![Array Diagram]

May 26, 2023

ECS 36A, Spring Quarter 2023
Rules for Pointers

• Treat a pointer like a constant or a variable
  • If it’s used as an array name, assume it’s a constant
  • Otherwise, assume it’s a variable
  • Note: in a function parameter list, it’s a variable, even if declared as an array

• A pointer p is an *address*
  • *p is the value stored at the address in p
  • &x is the address of the variable x
  • You can’t take the address of a constant, so this is illegal: char c[10]; d = &c;

• Draw pictures! They are very helpful
More C Library Functions

• time
• (pseudo)random numbers
• string functions
• memory functions
• math functions
Get Time

• Use system call `time_t time(time_t *tick)`
  • If `tick` is NULL, then the current time is returned
  • Time measured in seconds from the epoch (Jan 1, 1970, 00:00:00)

• To get time as a string: `char *ctime(&tick)`
  • On success, generates a string of the following form:
    Sun Sep 16 01:03:52 1973
    (This has a trailing newline)
  • On failure, it returns NULL
# Time Structure

```c
struct tm {
    int tm_sec;    /* 0-59 seconds */
    int tm_min;    /* 0-59 minutes */
    int tm_hour;   /* 0-23 hour */
    int tm_mday;   /* 1-31 day of month */
    int tm_mon;    /* 0-11 month */
    int tm_year;   /* 0-year - 1900 */
    int tm_wday;   /* 0-6 day of week (Sunday = 0) */
    int tm_yday;   /* 0-365 day of year */
    int tm_isdst;  /* flag: daylight savings time in effect */
    long tm_gmtoff; /* offset from GMT in seconds */
    char **tm_zone; /* abbreviation of timezone name */
};
```
Getting Structure Values for Time

• struct tm *localtime(const time_t *timep): fills in local time
• struct tm *gmtime(const time_t *timep): fills in GMT (UTC) time
  • Here timep is a pointer to what time returns
• char *asctime(struct tm *tm): return a ctime-type string for tm
• time_t mktime(struct tm *tm): return time since the epoch given by tm
Random Numbers

• int rand(void)
  • Generate pseudorandom number between 0 and RAND_MAX inclusive
  • This function is dangerous — avoid it!! In older versions, it is not pseudorandom in the low order bits. (On newer Linux systems, it’s OK)

• long random(void)
  • Generate pseudorandom number between 0 and $2^{31}-1$ inclusive

• All require a starting point – called a seed
Random Number Seeds

• void srand(unsigned int seed)
  • Initialize the $rand()$ pseudorandom number generator with $seed$
• void srandom(unsigned int seed)
  • Initialize the $random()$ pseudorandom number generator with $seed$
• Pick $seed$ as randomly as possible
• There are defaults, useful for regenerating the same sequence for debugging
  • rand/srand default seed is 1
  • random/srandom default seed is 1
String Functions

• strcpy, strcat, strcmp, strncpy, strncmp, strlen
  • You’ve seen these
• char *strdup(char *s): make a duplicate of string s
  • Space is malloc’ed
• char *strchr(char *s, int c): return pointer to first occurrence of character c in s; NULL if not there
• char *strrchr(char *s, int c): like strchr, but points to last occurrence
• char *strstr(char *s, char *t): like strchr, but looks for first occurrence of string t
String Functions

• char *strtok(char *s, char *delim): breaks a string into a sequence of 0 or more nonempty tokens (substrings)
  • On first call, s points to string to be parsed
  • On subsequent calls for the same string, set s to NULL
  • delim is a string of characters that delimit tokens
  • strtok returns NULL when there are no more tokens to return
  • strtok always returns a nonempty token
  • Warning: strtok overwrites delimiters with ‘\0’, so don’t give it a read-only string

• int strcasecmp(char *a, char *b): useful for homework; look it up
Memory Functions

• void *memcpy(void *dest, void *src, unsigned int n): copy n bytes from src to dest
  • Behavior undefined if src, dest overlap

• int memcmp(void *s1, void *s2, unsigned int n): compare first n bytes of s1 and s2; returns negative, zero, positive depending on whether s1 is less than, equal to, greater than s2
Math Functions

- `double floor(double d), double ceil(double d)`: round `d` down, up to the nearest integer
- `double log(double d), double log10(double d)`: return the natural log, base 10 log of `d`
- `double exp(double d), double pow(double m, double e)`: return $e^d$, $m^e$
- `double sin(double d)`: compute sine of $d$ in radians
  - same with cos, tan
- `double atan(double x)`: return principal value of arctan of $d$
  - In range $[-\pi/2, +\pi/2)$
- `double atan2(double x, double y)`: return arctan of $y/x$
  - Handles cases where $x$ is 0; returns value in range $[-\pi, \pi]$