CSE 333
Lecture 3 - pointers, pointers, pointers

Hal Perkins
Department of Computer Science & Engineering
University of Washington
Administrivia

hw0 out, due Mon. 11 pm.

- You should set up a source code repository for the hw projects this quarter. Cheap backup, easy to work from multiple locations/computers, etc.
  
  ▪ If you use a hosted service like github, be careful that you don’t make it visible to anyone else

ex2 out now, due Monday before class

- Good idea to start using clint.py to check code

hw1, first big project, out next week
Registration

We were able to add a few more students

- All are CSE majors who are required to take the course
- Priority went to those further along in the major or who had fewer options for other things to do

Apologies to everyone else, but resources are maxed out

Should be plenty of room this summer if that is an option
Agenda

Today’s goals:
- pointers
- more pointers
- pointers and call-by-reference
- arrays and pointers
& and *

`&foo`  // virtual address of foo - “address of”
`*pointer`  // dereference a pointer
`*pointer = value;`  // dereference / assign

```c
int x = 42;
int *p;  // p is a pointer to an integer
p = &x;  // p now stores the address of x

printf("x is %d\n", x);
*p = 99;
printf("x is %d\n", x);
```

deref.c
Something curious

Let’s try running this program several times:

```c
#include <stdio.h>

int main(int argc, char **argv) {
    int x = 1;
    int *p = &x;

    printf("&x: %p;  p: %p;  &p: %p\n",
            &x, p, &p);
    return 0;
}
```

```
[bash]$ ./asr
&x: 0xbfa521dc;  p: 0xbfa521dc;  &p: 0xbfa521d8
[bash]$ ./asr
&x: 0xbf836f5c;  p: 0xbf836f5c;  &p: 0xbf836f58
[bash]$ ./asr
&x: 0xbfea39dc;  p: 0xbfea39dc;  &p: 0xbfea39d8
```
Linux uses address-space randomization for added security

- Linux randomizes:
  - base of stack
  - shared library (mmap) location

- makes stack-based buffer overflow attacks tougher

- makes debugging tougher

- google “disable linux address space randomization”
Box and arrow diagrams

```c
int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];

    printf("&x: %p;  x: %d\n", &x, x);
    printf("&arr[0]: %p;  arr[0]: %d\n", &arr[0], arr[0]);
    printf("&arr[1]: %p;  arr[1]: %d\n", &arr[1], arr[1]);
    printf("&arr[2]: %p;  arr[2]: %d\n", &arr[2], arr[2]);
    printf("&p: %p;  p: %p;  *p: %d\n", &p, p, *p);

    return 0;
}
```

<table>
<thead>
<tr>
<th>address</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;x</td>
<td>x</td>
<td>value</td>
</tr>
<tr>
<td>&amp;arr[0]</td>
<td>arr[0]</td>
<td>value</td>
</tr>
<tr>
<td>&amp;p</td>
<td>p</td>
<td>value</td>
</tr>
</tbody>
</table>
Box and arrow diagrams

```c
int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];

    printf("\&x: %p;  x: %d\n", &x, x);
    printf("\&arr[0]: %p;  arr[0]: %d\n", &arr[0], arr[0]);
    printf("\&arr[2]: %p;  arr[2]: %d\n", &arr[2], arr[2]);
    printf("\&p: %p;  p: %p;  *p: %d\n", &p, p, *p);

    return 0;
}
```

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;x</td>
<td>x</td>
</tr>
<tr>
<td>&amp;arr[0]</td>
<td>arr[0]</td>
</tr>
<tr>
<td>&amp;p</td>
<td>p &amp;arr[1]</td>
</tr>
</tbody>
</table>
Box and arrow diagrams

int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];

    printf("&x: %p;  x: %d\n", &x, x);
    printf("&arr[0]: %p;  arr[0]: %d\n", &arr[0], arr[0]);
    printf("&arr[1]: %p;  arr[1]: %d\n", &arr[1], arr[1]);
    printf("&arr[2]: %p;  arr[2]: %d\n", &arr[2], arr[2]);
    printf("&p: %p;  p: %p; *p: %d\n", &p, p, *p);

    return 0;
}

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0xbfff2dc</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>arr[0]</td>
<td>0xbfff2d0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>arr[1]</td>
<td>0xbfff2d4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>arr[2]</td>
<td>0xbfff2d8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>0xbfff2cc</td>
<td>0xbfff2d4</td>
<td></td>
</tr>
</tbody>
</table>
Box and arrow diagrams

```c
int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];

    printf("&x: %p;  x: %d\n", &x, x);
    printf("&arr[0]: %p;  arr[0]: %d\n", &arr[0], arr[0]);
    printf("&arr[2]: %p;  arr[2]: %d\n", &arr[2], arr[2]);
    printf("&p: %p;  p: %p;  *p: %d\n", &p, p, *p);

    return 0;
}
```

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>arr[0]</td>
<td>1</td>
</tr>
<tr>
<td>arr[1]</td>
<td>3</td>
</tr>
<tr>
<td>arr[2]</td>
<td>4</td>
</tr>
<tr>
<td>p</td>
<td>0xbfff2dc</td>
</tr>
</tbody>
</table>

main()'s stack frame

CSE333 lec 3 C.3 // 04-04-14 // Perkins
Box and arrow diagrams

```c
int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];

    printf("&x: %p; x: %d\n", &x, x);
    printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
    printf("&arr[1]: %p; arr[1]: %d\n", &arr[1], arr[1]);
    printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
    printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);

    return 0;
}
```

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
<td>0xbfff2dc</td>
</tr>
<tr>
<td>arr[0]</td>
<td>2</td>
<td>0xbfff2d0</td>
</tr>
<tr>
<td>arr[1]</td>
<td>3</td>
<td>0xbfff2d4</td>
</tr>
<tr>
<td>arr[2]</td>
<td>4</td>
<td>0xbfff2d8</td>
</tr>
<tr>
<td>p</td>
<td>0xbfff2cc</td>
<td>0xbfff2d4</td>
</tr>
</tbody>
</table>
Box and arrow diagrams

```c
int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];
    int **dp = &p;

    (*dp) += 1;
    p += 1;
    (*dp) += 1;

    return 0;
}
```

<table>
<thead>
<tr>
<th>address</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xbfff2dc</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>0xbfff2d0</td>
<td>arr[0]</td>
<td>2</td>
</tr>
<tr>
<td>0xbfff2d4</td>
<td>arr[1]</td>
<td>3</td>
</tr>
<tr>
<td>0xbfff2d8</td>
<td>arr[2]</td>
<td>4</td>
</tr>
<tr>
<td>0xbfff2c8</td>
<td>dp</td>
<td>0xbfff2cc</td>
</tr>
<tr>
<td>0xbfff2cc</td>
<td>p</td>
<td>0xbfff2d4</td>
</tr>
</tbody>
</table>
```c
int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];
    int **dp = &p;

    (*(dp) += 1);
    p += 1;
    (*(dp) += 1);

    return 0;
}
```

<table>
<thead>
<tr>
<th>address</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xbfff2dc</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>0xbfff2d0</td>
<td>arr[0]</td>
<td>2</td>
</tr>
<tr>
<td>0xbfff2d4</td>
<td>arr[1]</td>
<td>4</td>
</tr>
<tr>
<td>0xbfff2d8</td>
<td>arr[2]</td>
<td>4</td>
</tr>
<tr>
<td>0xbfff2c8</td>
<td>dp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>0xbfff2d4</td>
</tr>
</tbody>
</table>
Box and arrow diagrams

```c
#include <stdio.h>

int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];
    int **dp = &p;
    
    (*dp) += 1;
    p += 1;
    (*dp) += 1;
    
    return 0;
}
```

<table>
<thead>
<tr>
<th>address</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xbfff2dc</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>0xbfff2d0</td>
<td>arr[0]</td>
<td>2</td>
</tr>
<tr>
<td>0xbfff2d4</td>
<td>arr[1]</td>
<td>4</td>
</tr>
<tr>
<td>0xbfff2d8</td>
<td>arr[2]</td>
<td>4</td>
</tr>
</tbody>
</table>
int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];
    int **dp = &p;
    *(dp) += 1;
    p += 1;
    *(dp) += 1;
    return 0;
}
Pointer arithmetic

Pointers are typed

- int *int_ptr; vs. char *char_ptr;
- pointer arithmetic obeys those types
- see pointerarithmetic.c
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 2;   // uh oh
    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);

    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);

    return 0;
}
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p;  *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p;  *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2;  // uh oh
    printf("int_ptr: %p;  *int_ptr: %d\n",
           int_ptr, *int_ptr);

    printf("char_ptr: %p;  *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p;  *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p;  *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}
```c
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 2;  // uh oh
    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);

    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);

    return 0;
}
```
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p;  *int_ptr: %d
    int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p;  *int_ptr: %d
    int_ptr, *int_ptr);
    int_ptr += 2;  // uh oh
    printf("int_ptr: %p;  *int_ptr: %d
    int_ptr, *int_ptr);

    printf("char_ptr: %p;  *char_ptr: %d\n    char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p;  *char_ptr: %d\n    char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p;  *char_ptr: %d\n    char_ptr, *char_ptr);

    return 0;
}
```c
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p;  *int_ptr: %d\n",
            int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p;  *int_ptr: %d\n",
            int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p;  *int_ptr: %d\n",
            int_ptr, *int_ptr);

    printf("char_ptr: %p;  *char_ptr: %d\n",
            char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p;  *char_ptr: %d\n",
            char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p;  *char_ptr: %d\n",
            char_ptr, *char_ptr);

    return 0;
}
```

```c
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p;  *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p;  *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2;  // uh oh
    printf("int_ptr: %p;  *int_ptr: %d\n",
           int_ptr, *int_ptr);

    printf("char_ptr: %p;  *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p;  *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p;  *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}
```
```c
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 2;  // uh oh
    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);

    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);

    return 0;
}
```

`pointerarithmetic.c`
```c
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p;  *int_ptr: %d\n",
            int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p;  *int_ptr: %d\n",
            int_ptr, *int_ptr);
    int_ptr += 2;  // uh oh
    printf("int_ptr: %p;  *int_ptr: %d\n",
            int_ptr, *int_ptr);

    printf("char_ptr: %p;  *char_ptr: %d\n",
            char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p;  *char_ptr: %d\n",
            char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p;  *char_ptr: %d\n",
            char_ptr, *char_ptr);

    return 0;
}
```
```c
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p;  *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p;  *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2;  // uh oh
    printf("int_ptr: %p;  *int_ptr: %d\n",
           int_ptr, *int_ptr);

    printf("char_ptr: %p;  *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p;  *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p;  *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}
```
```c
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n", int_ptr, *int_ptr);

    printf("char_ptr: %p; *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n", char_ptr, *char_ptr);

    return 0;
}
```

`pointerarithmetic.c`
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2;  // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);

    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);

    return 0;
}
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);

    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);

    return 0;
}
```c
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n", int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n", int_ptr, *int_ptr);

    printf("char_ptr: %p; *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n", char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n", char_ptr, *char_ptr);

    return 0;
}
```
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n", 
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n", 
           int_ptr, *int_ptr);
    int_ptr += 2;  // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n", 
           int_ptr, *int_ptr);

    printf("char_ptr: %p; *char_ptr: %d\n", 
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n", 
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n", 
           char_ptr, *char_ptr);

    return 0;
}
```c
#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p;  *int_ptr: %d\n",
            int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p;  *int_ptr: %d\n",
            int_ptr, *int_ptr);
    int_ptr += 2;  // uh oh
    printf("int_ptr: %p;  *int_ptr: %d\n",
            int_ptr, *int_ptr);

    printf("char_ptr: %p;  *char_ptr: %d\n",
            char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p;  *char_ptr: %d\n",
            char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p;  *char_ptr: %d\n",
            char_ptr, *char_ptr);

    return 0;
}
```
Pass-by-value

C passes arguments by **value**

- callee receives a **copy** of the argument
- if the callee modifies an argument, caller’s copy isn’t modified

```c
void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(a, b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}

brokenswap.c
```
Pass-by-value (stack)

<table>
<thead>
<tr>
<th>OS kernel [protected]</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
</tr>
<tr>
<td>main</td>
</tr>
<tr>
<td>heap (malloc/free)</td>
</tr>
<tr>
<td>read/write segment</td>
</tr>
<tr>
<td>.data, .bss</td>
</tr>
<tr>
<td>read-only segment</td>
</tr>
<tr>
<td>.text, .rodata</td>
</tr>
</tbody>
</table>

```c
void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(a, b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}
```

brokenswap.c
Pass-by-value (stack)

<table>
<thead>
<tr>
<th>OS kernel [protected]</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
</tr>
<tr>
<td>main</td>
</tr>
<tr>
<td>swap</td>
</tr>
<tr>
<td>heap (malloc/free)</td>
</tr>
<tr>
<td>read/write segment</td>
</tr>
<tr>
<td>.data, .bss</td>
</tr>
<tr>
<td>read-only segment</td>
</tr>
<tr>
<td>.text, .rodata</td>
</tr>
</tbody>
</table>

```c
void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(a, b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}
```

brokenswap.c
Pass-by-value (stack)

<table>
<thead>
<tr>
<th>OS kernel [protected]</th>
<th>void swap(int a, int b) {</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
<td>int tmp = a;</td>
</tr>
<tr>
<td>main</td>
<td>a = b;</td>
</tr>
<tr>
<td>swap</td>
<td>b = tmp;</td>
</tr>
<tr>
<td>temp</td>
<td>}</td>
</tr>
<tr>
<td>heap (malloc/free)</td>
<td>int main(int argc, char **argv) {</td>
</tr>
<tr>
<td>read/write segment</td>
<td>int a = 42, b = -7;</td>
</tr>
<tr>
<td>.data, .bss</td>
<td>swap(a, b);</td>
</tr>
<tr>
<td>read-only segment</td>
<td>printf(&quot;a: %d, b: %d\n&quot;, a, b);</td>
</tr>
<tr>
<td>.text, .rodata</td>
<td>return 0;</td>
</tr>
</tbody>
</table>

brokenswap.c
Pass-by-value (stack)

<table>
<thead>
<tr>
<th></th>
<th>OS kernel [protected]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>stack</td>
</tr>
<tr>
<td>main</td>
<td>a 42 b -7</td>
</tr>
<tr>
<td>swap</td>
<td>a 42 b -7</td>
</tr>
<tr>
<td>tmp</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>heap (malloc/free)</td>
</tr>
<tr>
<td></td>
<td>read/write segment</td>
</tr>
<tr>
<td>.data, .bss</td>
<td></td>
</tr>
<tr>
<td></td>
<td>read-only segment</td>
</tr>
<tr>
<td>.text, .rodata</td>
<td></td>
</tr>
</tbody>
</table>

```c
void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(a, b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}
```

brokenswap.c
Pass-by-value (stack)

<table>
<thead>
<tr>
<th>OS kernel [protected]</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
</tr>
<tr>
<td>main</td>
</tr>
<tr>
<td>a 42 b -7</td>
</tr>
<tr>
<td>swap</td>
</tr>
<tr>
<td>a -7 b -7</td>
</tr>
<tr>
<td>tmp 42</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>heap (malloc/free)</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>read/write segment</td>
</tr>
<tr>
<td>.data, .bss</td>
</tr>
<tr>
<td>read-only segment</td>
</tr>
<tr>
<td>.text, .rodata</td>
</tr>
</tbody>
</table>

```c
void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(a, b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}
```

brokenswap.c
Pass-by-value (stack)

OS kernel [protected]

stack

main

swap

heap (malloc/free)

read/write segment

.read, .bss

read-only segment

.text, .rodata

void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(a, b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}

brokenswap.c
Pass-by-value (stack)

<table>
<thead>
<tr>
<th>OS kernel [protected]</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
</tr>
<tr>
<td>main</td>
</tr>
</tbody>
</table>

heap (malloc/free)

read/write segment
.data, .bss

read-only segment
.text, .rodata

void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(a, b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}

brokenswap.c
Pass-by-value (stack)

OS kernel [protected]
- stack

main
- a 42
- b -7

heap (malloc/free)

read/write segment
- .data, .bss

read-only segment
- .text, .rodata

void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(a, b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}

brokenswap.c
Pass-by-reference

You can use pointers to pass by reference

- callee still receives a copy of the argument
  - but, the argument is a pointer
  - the pointer’s value points-to the variable in the scope of the caller
- this gives the callee a way to modify a variable that’s in the scope of the caller

```c
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(&a, &b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}
```

swap.c
Pass-by-reference (stack)

### OS kernel [protected]

- stack

### heap (malloc/free)

### read/write segment
- .data, .bss

### read-only segment
- .text, .rodata

```c
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(&a, &b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}
```

swap.c
Pass-by-reference (stack)

OS kernel [protected]

- stack
- \textbf{main} \ a \ 42 \ b \ -7
- \textbf{swap} \ a \ ? \ b \ ? \ \textbf{tmp} \ ?

heap (malloc/free)

read/write segment

- .\textit{data}, .\textit{bss}

read-only segment

- .\textit{text}, .\textit{rodata}

\begin{verbatim}
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(&a, &b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}
\end{verbatim}

swap.c
Pass-by-reference (stack)

void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(&a, &b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}

swap.c
Pass-by-reference (stack)

- OS kernel [protected]
  - stack
  - main
    - a 42 b -7
  - swap
    - a
    - b
    - tmp 42
- heap (malloc/free)
- read/write segment
  - .data, .bss
- read-only segment
  - .text, .rodata

```c
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;

    swap(&a, &b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}

swap.c
```
Pass-by-reference (stack)

OS kernel [protected]

stack

main

swap

heap (malloc/free)

read/write segment

.data, .bss

read-only segment

.text, .rodata

void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(&a, &b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}

swap.c
Pass-by-reference (stack)

OS kernel [protected]

<table>
<thead>
<tr>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
</tr>
<tr>
<td>swap</td>
</tr>
</tbody>
</table>

heap (malloc/free)

read/write segment

.data, .bss

read-only segment

.text, .rodata

void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(&a, &b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}

swap.c
Pass-by-reference (stack)

void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;
    swap(&a, &b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}
Pass-by-reference (stack)

<table>
<thead>
<tr>
<th>OS kernel [protected]</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
</tr>
<tr>
<td>main</td>
</tr>
</tbody>
</table>

heap (malloc/free)

read/write segment
.data, .bss

read-only segment
.text, .rodata

void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main(int argc, char **argv) {
  int a = 42, b = -7;
  swap(&a, &b);
  printf("a: %d, b: %d\n", a, b);
  return 0;
}

swap.c
Arrays and pointers

A pointer can point to an array element

- An array's name can be used as a pointer to its first element
- And, you can use [] notation to treat a pointer like an array

\[ \text{pointer}[i] \] is i elements' worth of bytes forward from pointer

```c
int a[5] = {10, 20, 30, 40, 50};
int* p1 = &a[3]; // refers to a's fourth element
int* p2 = &a[0]; // refers to a's first element
int* p3 = a; // refers to a's first element

*p1 = 100;
*p2 = 200;
p1[1] = 300;
p2[1] = 400;
p3[2] = 500; // final: 200, 400, 500, 100, 300
```
Passing arrays as parameters

array parameters are really passed as pointers to the first array element

- the [] syntax for parameter types is just for convenience

```c
void f(int a[]);
int main(...) {
    int a[5];
    ...
    f(a);
    return 0;
}

void f(int *a) {
    your code
}
```

```c
void f(int *a);
int main(...) {
    int a[5];
    ...
    f(&a[0]);
    return 0;
}

void f(int *a) {
    equivalent to
```
Self-Exercise 1

Use a box-and-arrow diagram for the following program to explain what it prints out:

```c
#include <stdio.h>

int foo(int *bar, int **baz) {
    *bar = 5;
    *(bar+1) = 6;
    *baz = bar+2;
    return *((*baz)+1);
}

int main(int argc, char **argv) {
    int arr[4] = {1, 2, 3, 4};
    int *ptr;
    arr[0] = foo(&arr[0], &ptr);
    printf("%d %d %d %d\n",
            arr[0], arr[1], arr[2], arr[3], *ptr);
    return 0;
}
```
Self-Exercise 2

Write a program that prints out whether the computer it is running on is little endian or big endian.

- (hint: see pointerarithmetic.c from today’s lecture)
Self-Exercise 3

Write a function that:

- accepts an (array of ints) and an (array length) as arguments
- malloc’s an (array of (int *)) of the same length
- initializes each element of the newly allocated array to point to the corresponding element in the passed-in array
- returns a pointer to the newly allocated array
See you on Monday!