Points (1/4)  …review…

• Sometimes you want to have a procedure increment a variable?
• What gets printed?

```c
void AddOne(int x) {  
    x = x + 1;  
}  

int y = 5;  
AddOne( y);  
printf("y = %d\n", y);
```

Points (2/4)  …review…

• Solved by passing in a pointer to our subroutine.
• Now what gets printed?

```c
void AddOne(int *p) {  
    *p = *p + 1;  
}  

int y = 5;  
AddOne(&y);  
printf("y = %d\n", y);
```

Points (3/4)

• But what if what you want changed is a pointer?
• What gets printed?

```c
void IncrementPtr(int *p) {  
    *p = *p + 1;  
}  

int A[3] = {50, 60, 70};  
int *q = A;  
IncrementPtr( q);  
printf("*q = %d\n", *q);
```

Points (4/4)

• Solution! Pass a pointer to a pointer, declared as **h
• Now what gets printed?

```c
void IncrementPtr(int **h) {  
    **h = **h + 1;  
}  

int A[3] = {50, 60, 70};  
int *q = A;  
IncrementPtr(&q);  
printf("q = %d\n", *q);
```
Dynamic Memory Allocation (1/4)

- C has operator `sizeof()` which gives size in bytes (of type or variable)
- Assume size of types can be misleading and is bad style, so use `sizeof(type)`
- Many years ago an int was 16 bits, and programs were written with this assumption.
- What is the size of integers now?
- "sizeof" knows the size of arrays:
  ```
  int ar[3]; // Or: int ar[] = {54, 47, 99}
  ```
  `sizeof(ar)` ⇒ 12
  ```
  int n = 3;
  int ar[n]; // Or: int ar[fun_that_returns_3()];
  ```
  `sizeof(ar)` ⇒ 12

Dynamic Memory Allocation (2/4)

- To allocate room for something new to point to, use `malloc()` (with the help of a typecast and `sizeof()`):
  ```
  ptr = (int*) malloc (sizeof(int));
  ```
  - Now, `ptr` points to a space somewhere in memory of size `sizeof(int)` in bytes.
  - `(int*)` simply tells the compiler what will go into that space (called a typecast).
  - `malloc` is almost never used for 1 var
  ```
  ptr = (int*) malloc (n*sizeof(int));
  ```
  This allocates an array of `n` integers.

Dynamic Memory Allocation (3/4)

- Once `malloc()` is called, the memory location contains garbage, so don’t use it until you’ve set its value.
- After dynamically allocating space, we must dynamically free it:
  ```
  free(ptr);
  ```
- Use this command to clean up.
  - Even though the program `free`s all memory on exit (or when main returns), don’t be lazy!
  - You never know when your `main` will get transformed into a subroutine!

Dynamic Memory Allocation (4/4)

- The following two things will cause your program to crash or behave strangely later on, and cause VERY VERY hard to figure out bugs:
  - `free()`ing the same piece of memory twice
  - calling `free()` on something you didn’t get back from `malloc()`
- The runtime does not check for these mistakes
  - Memory allocation is so performance-critical that there just isn’t time to do this
  - The usual result is that you corrupt the memory allocator’s internal structure
  - You won’t find out until much later on, in a totally unrelated part of your code!

Binky Pointer Video (thanks to NP @ SU)

- **Pointer Fun with Binky**
  by Nick Parlante
  This is document 104 in the Stanford CS Education Library — please see calibr8.stanford.edu for this video, its associated documents, and other free educational materials.

Administrivia

- Next week is memory management and MIPS!
  - By this point next week, you’ll be able to code in assembly!
- Lab problems
  - Should be fixed!
  - Reminder, you must get checked off at the start of lab TODAY for Tuesday!
- C Help Session
  - How was it?
- HW1 due Friday night! If you aren’t sure about how submitting works, ask in lab!
Kissing Mel Gibson, Tom Petty exclaimed: “Zesty, kissing mediocre girls/guys teaches people (to) expect zero (from) you.

Kindergarten means giving teachers perfect examples (of) zeal (&) youth.

Killing messengers gives terrible people exactly zero,

Kindness means giving, teaching, permeating excess zeal yourself. –

Kissing Mentors Gives Testy Persistent Extremists Zealous Youthfulness. –

Kind men give ten percent extra, zestfully, youthfully.

Kid meets giant Texas people exercising like yoga. – Rolf G

Gary M C structures: Pointers to them

• Usually, more efficient to pass a pointer to the struct.

• The C arrow operator (->) dereferences and extracts a structure field with a single operator.

• The following are equivalent:
  
  struct point *p;
  /* code to assign to pointer */
  printf("x is %d\n", (*p).x);
  printf("x is %d\n", p->x);

Arrays not implemented as you’d think

void foo() {
  int *p, *q, x;
  int a[4];
  p = (int *) malloc (sizeof(int));
  q = a;
  *p = 1; // p[0] would also work here
  printf("p:%x, q:%x, &q:%x\n", *p, p, &p);
  *q = 2; // q[0] would also work here
  printf("q:%x, q:%x, &q:%x\n", *q, q, &q);
  *a = 3; // a[0] would also work here
  printf("a:%x, a:%x, &a:%x\n", *a, a, &a);
}

K&R: “An array name is not a variable”

C structures: Points to them

• A struct is a data structure composed from simpler data types.

  • Like a class in Java/C++ but without methods or inheritance.

  struct point {
  // type definition */
  int x;
  int y;
};

  As always in C, the argument is passed by “value” – a copy is made.

void PrintPoint(struct point p) {
  printf("x=%d, y=%d\n", p.x, p.y);
}

struct point pl = {0,10}; /* x=0, y=10 */
PrintPoint(pl);

How big are structs?

• Recall C operator sizeof() which gives size in bytes (of type or variable)

• How big is sizeof(p) ?

  struct p {
    int y;
  };

  • 5 bytes? 8 bytes?
  • Compiler may word align integer y

C structures: Overview

Kilo, Mega, Giga, Tera, Peta, Exa, Zetta, Yotta

1. Kid meets giant Texas people exercising like yoga. – Rolf G
2. Kind men give ten percent extra, zestfully, youthfully. – Hava E
3. Kissing Mentors Gives Testy Persistent Extremists Zealous Youthfulness. – Gary M
4. Kindness means giving, teaching, permeating excess zeal yourself. – Hava E
5. Killing messengers gives terrible people exactly zero, yo
6. Kindergarten means giving teachers perfect examples (of) zeal (&) youth
7. Kissing mediocre girls/guys teaches people (to) expect zero (from) you
9. Kissing me gives ten percent extra zeal & youth! – Dan G (borrowing parts)

Peer Instruction
Which are guaranteed to print out 5?

I: main()
   int *a_ptr = (int*)malloc(sizeof(int));
   *a_ptr = 5;
   printf("%d", *a_ptr);
}

II: main()
   int *p, a = 5;
   p = &a; ...
   /* code; a, p NEVER on LEFT of = */
   printf("%d", a);

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>II</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>YES</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
<td>YES</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
<td>No idea</td>
</tr>
</tbody>
</table>

"And in Conclusion..."

- Use handles to change pointers
- Create abstractions with structures
- Dynamically allocated heap memory must be manually deallocated in C.
  - Use malloc() and free() to allocate and deallocate memory from heap.

- Have a great weekend!

Reference slides

You ARE responsible for the material on these slides (they’re just taken from the reading anyway); we’ve moved them to the end and off-stage to give more breathing room to lecture!

Linked List Example

- Let’s look at an example of using structures, pointers, malloc(), and free() to implement a linked list of strings.

```c
/* node structure for linked list */
struct Node {
    char *value;
    struct Node *next;
};

typedef struct Node NodeStruct;

... OR ...

typedef NodeStruct Node;

... THEN

typedef NodeStruct *List;
typedef char *String;
```

```c
/* Add a string to an existing list */
List cons(String s, List list)
{
    List node = (List) malloc(sizeof(NodeStruct));
    node->value = (String) malloc(strlen(s) + 1);
    strcpy(node->value, s);
    node->next = list;
    return node;
}

String s1 = "abc", s2 = "cde";
List thelist = NULL;
thealist = cons(s2, thealist);
thealist = cons(s1, thealist);
/* or, just like (cons s1 (cons s2 nil)) */
thealist = cons(s2, cons(s1, cons(s2, NULL)));
```
/* Add a string to an existing list, 2nd call */
List cons(String s, List list)
{
    List node = (List) malloc(sizeof(NodeStruct));
    node->value = (String) malloc(strlen(s) + 1);
    strcpy(node->value, s);
    node->next = list;
    return node;
}