

**Lecture 5 – Introduction to C (pt 3)
C Memory Management**



2007-01-26

There is one handout today at the front and back of the room!

Lecturer SOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Norway: iTunes illegal! ⇒ Norway ruled that iTunes was illegal because it did not allow downloaded songs encoded with their proprietary Fairplay system to be played on non-iPods. They are asking Apple to open their system up by Oct 1.



www.msnbc.msn.com/id/16793043/



CS61C L05 Introduction to C (pt 3) (1)

Garcia, Spring 2007 © UCB

Review

- Pointers and arrays are **virtually same**
- C knows how to **increment pointers**
- C is an efficient language, with little protection
 - Array bounds **not checked**
 - Variables **not automatically initialized**
- (Beware) The cost of efficiency is more overhead for the programmer.
 - “C gives you a lot of extra rope but be careful not to hang yourself with it!”



CS61C L05 Introduction to C (pt 3) (2)

Garcia, Spring 2007 © UCB

C Strings

- A **string** in C is just an array of characters.

```
char string[] = "abc";
```

- How do you tell how long a string is?

- Last character is followed by a 0 byte (null terminator)

```
int strlen(char s[])  
{  
    int n = 0;  
    while (s[n] != 0) n++;  
    return n;  
}
```



CS61C L05 Introduction to C (pt 3) (3)

Garcia, Spring 2007 © UCB

Pointers (1/4)

...review...

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

```
void AddOne(int x) { x = x + 1; }  
  
int y = 5;  
AddOne(y);  
printf("y = %d\n", y);
```



CS61C L05 Introduction to C (pt 3) (4)

Garcia, Spring 2007 © UCB

Pointers (2/4)

...review...

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

```
void AddOne(int *p) { *p = *p + 1; }  
  
int y = 5;  
AddOne(&y);  
printf("y = %d\n", y);
```



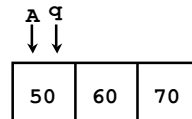
CS61C L05 Introduction to C (pt 3) (5)

Garcia, Spring 2007 © UCB

Pointers (3/4)

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

```
void IncrementPtr(int *p) { p = p + 1; }  
  
int A[3] = {50, 60, 70};  
int *q = A;  
IncrementPtr(q);  
printf("*q = %d\n", *q);
```



CS61C L05 Introduction to C (pt 3) (6)

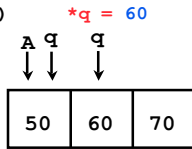
Garcia, Spring 2007 © UCB

Pointers (4/4)

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

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

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



CS61C L05 Introduction to C (pt 3) (7)

Garcia, Spring 2007 © UC

Dynamic Memory Allocation (1/4)

- C has operator `sizeof()` which gives size in bytes (of type or variable)
- Assume size of objects 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
```

 - ...as well for arrays whose size is determined at run-time:

```
int n = 3;
int ar[n]; // Or: int ar[fun_that_returns_3()];
sizeof(ar) ⇒ 12
```



CS61C L05 Introduction to C (pt 3) (8)

Garcia, Spring 2007 © UC

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.



CS61C L05 Introduction to C (pt 3) (9)

Garcia, Spring 2007 © UC

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!



CS61C L05 Introduction to C (pt 3) (10)

Garcia, Spring 2007 © UC

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!



CS61C L05 Introduction to C (pt 3) (11)

Garcia, Spring 2007 © UC

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 cslibrary.stanford.edu for this video, its associated documents, and other free educational materials.

Copyright © 1999 Nick Parlante. See copyright panel for redistribution terms.
Carpe Post Meridiam!



CS61C L05 Introduction to C (pt 3) (12)

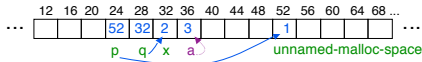
Garcia, Spring 2007 © UC

Arrays not implemented as you'd think

```
void foo() {
    int *p, *q, x, a[1]; // a[] = {3} also works here
    p = (int *) malloc (sizeof(int));
    q = &x;

    *p = 1; // p[0] would also work here
    *q = 2; // q[0] would also work here
    *a = 3; // a[0] would also work here

    printf("p: %u, q: %u, x: %u\n", *p, *q, *a);
    printf("p: %u, q: %u, x: %u\n", *p, *q, *a);
    printf("a: %u, a: %u, a: %u\n", *a, *a, *a);
}
```



```
*p: 1, p: 52, &p: 24
*q: 2, q: 32, &q: 28
*a: 3, a: 36, &a: 36
```



CS61C L05 Introduction to C (pt 3) (13)

Garcia, Spring 2007 © UCB

C structures : Overview

• 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;
};
```

```
void PrintPoint(struct point p)
{ As always in C, the argument is passed by "value" – a copy is made.
  printf("%d, %d", p.x, p.y);
}
```

```
struct point p1 = {0,10}; /* x=0, y=10 */
```

```
PrintPoint(p1);
```



CS61C L05 Introduction to C (pt 3) (14)

Garcia, Spring 2007 © UCB

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);
```



CS61C L05 Introduction to C (pt 3) (15)

Garcia, Spring 2007 © UCB

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

1. Kind Meek Giggles Tease Peering Excited Zealous Youngsters. – Yiding J
2. Kissing me gives tears per extra zebra YO! – Peter D
3. Kiss me, gimme tea, persistently extol zee. You! – Hava E
4. Kia Mechanics (are) Giant Terrible People Exclaiming Zealous Yodels. – Gary M
5. Kiss me, gimme tea, pet exaltingly, zestful you. – Hava E
6. Kid meets giant Texas people exercising zen-like yoga. -Rolf O
7. Kicking methods gives teaching people extra zest, youbetcha! – Peter D
8. Kind men give ten percent extra, zestfully, youthfully. – Hava E
9. Kissing Mentors Gives Testy Persistent Extremists Zealous Youthfulness. – Gary M
10. Kindness means giving, teaching, permeating excess zeal yourself. – Hava E
11. Kissing me gives ten percent extra zeal & youth! – Dan (taking ideas from all)
 1. Killing messengers gives terrible people exactly zero, yo
 2. Kindergarten means giving teachers perfect examples (of zeal (&) youth
 3. Kissing mediocre girls/guys teaches people (to) expect zero (from) you
 4. Kinky Mean Girls Teach Penis-Extending Zen Yoga
 5. Kissing Mel Gibson, Teddy Pendergrass exclaimed: "Zesty, yo!" – Dan Garcia



CS61C L05 Introduction to C (pt 3) (16)

Garcia, Spring 2007 © UCB

Pointer Arithmetic Peer Instruction Q

How many of the following are **invalid**?

- I. pointer + integer
- II. integer + pointer
- III. pointer + pointer
- IV. pointer – integer
- V. integer – pointer
- VI. pointer – pointer
- VII. compare pointer to pointer
- VIII. compare pointer to integer
- IX. compare pointer to 0
- X. compare pointer to NULL

#invalid
1
2
3
4
5
6
7
8
9
(1) 0



CS61C L05 Introduction to C (pt 3) (17)

Garcia, Spring 2007 © UCB

Peer Instruction

Which are guaranteed to print out 5?

```
I: main() {
    int *a-ptr; *a-ptr = 5; printf("%d", *a-ptr); }
```

```
II: main() {
    int *p, a = 5;
    p = &a;
    /* code; a & p NEVER on LHS of = */
    printf("%d", a); }
```

```
III: main() {
    int *ptr;
    ptr = (int *) malloc (sizeof(int));
    *ptr = 5;
    printf("%d", *ptr); }
```

	I	II	III
0:	-	-	-
1:	-	-	YES
2:	-	YES	-
3:	-	YES	YES
4:	YES	-	-
5:	YES	-	YES
6:	YES	YES	-
7:	YES	YES	YES

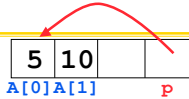


CS61C L05 Introduction to C (pt 3) (19)

Garcia, Spring 2007 © UCB

Peer Instruction

```
int main(void){
int A[] = {5,10};
int *p = A;
```



```
printf(“%u %d %d %d\n”, p, *p, A[0], A[1]);
p = p + 1;
printf(“%u %d %d %d\n”, p, *p, A[0], A[1]);
*p = *p + 1;
printf(“%u %d %d %d\n”, p, *p, A[0], A[1]);
}
```

If the first printf outputs `100 5 5 10`, what will the other two printf output?

```
1: 101 10 5 10      then 101 11 5 11
2: 104 10 5 10      then 104 11 5 11
3: 101 <other> 5 10 then 101 <3-others>
4: 104 <other> 5 10 then 104 <3-others>
5: One of the two printf causes an ERROR
6: I surrender!
```



CS61C L05 Introduction to C (pt 3) (20)

Garcia, Spring 2007 © UCB

“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.



CS61C L05 Introduction to C (pt 3) (21)

Garcia, Spring 2007 © UCB

Bonus slides

- These are extra slides that used to be included in lecture notes, but have been moved to this, the “bonus” area to serve as a supplement.
- The slides will appear in the order they would have in the normal presentation

BONUS



CS61C L05 Introduction to C (pt 3) (22)

Garcia, Spring 2007 © UCB

How big are structs?

- Recall C operator `sizeof()` which gives size in bytes (of type or variable)
- How big is `sizeof(p)`?

```
struct p {
    char x;
    int y;
};
```

- 5 bytes? 8 bytes?
- Compiler may word align integer `y`



CS61C L05 Introduction to C (pt 3) (23)

Garcia, Spring 2007 © UCB

Linked List Example

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

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



CS61C L05 Introduction to C (pt 3) (24)

Garcia, Spring 2007 © UCB

typedef simplifies the code

```
struct Node {
    char *value;
    struct Node *next;
};
```

String value;

```
/* "typedef" means define a new type */
typedef struct Node NodeStruct;
```

```
... OR ...
typedef struct Node {
    char *value;
    struct Node *next;
} NodeStruct;
```

... THEN

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

```
/* Note similarity! */
/* To define 2 nodes */
```

```
struct Node {
    char *value;
    struct Node *next;
} node1, node2;
```



CS61C L05 Introduction to C (pt 3) (25)

Garcia, Spring 2007 © UCB

Linked List Example

```

/* 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;
    theList = cons(s2, theList);
    theList = cons(s1, theList);
}
/* or, just like (cons s1 (cons s2 nil)) */
theList = cons(s1, cons(s2, NULL));

```



CS61C L05 Introduction to C (pt 2) (26)

Garcia, Spring 2007 © UC Berkeley

Linked List Example

```

/* Add a string to an existing list, 2nd call */
List cons(String s, List list)
{
    List node = (List) malloc(sizeof(NodeStruct));

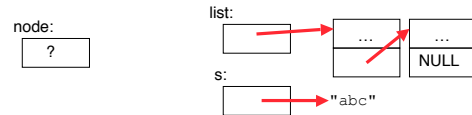
    node->value = (String) malloc (strlen(s) + 1);
    strcpy(node->value, s);
    node->next = list;
    return node;
}

```



CS61C L05 Introduction to C (pt 3) (27)

Garcia, Spring 2007 © UC Berkeley



Linked List Example

```

/* Add a string to an existing list, 2nd call */
List cons(String s, List list)
{
    List node = (List) malloc(sizeof(NodeStruct));

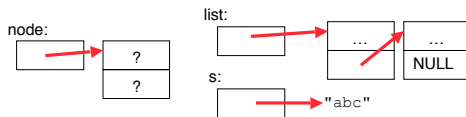
    node->value = (String) malloc (strlen(s) + 1);
    strcpy(node->value, s);
    node->next = list;
    return node;
}

```



CS61C L05 Introduction to C (pt 3) (28)

Garcia, Spring 2007 © UC Berkeley



Linked List Example

```

/* Add a string to an existing list, 2nd call */
List cons(String s, List list)
{
    List node = (List) malloc(sizeof(NodeStruct));

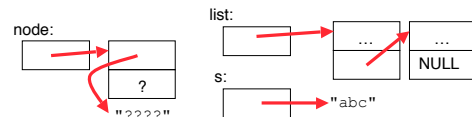
    node->value = (String) malloc (strlen(s) + 1);
    strcpy(node->value, s);
    node->next = list;
    return node;
}

```



CS61C L05 Introduction to C (pt 3) (29)

Garcia, Spring 2007 © UC Berkeley



Linked List Example

```

/* Add a string to an existing list, 2nd call */
List cons(String s, List list)
{
    List node = (List) malloc(sizeof(NodeStruct));

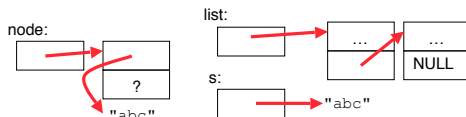
    node->value = (String) malloc (strlen(s) + 1);
    strcpy(node->value, s);
    node->next = list;
    return node;
}

```



CS61C L05 Introduction to C (pt 3) (30)

Garcia, Spring 2007 © UC Berkeley



Linked List Example

```

/* Add a string to an existing list, 2nd call */
List cons(String s, List list)
{
    List node = (List) malloc(sizeof(NodeStruct));

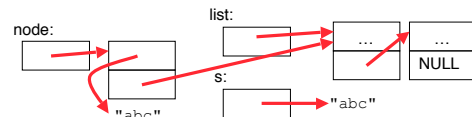
    node->value = (String) malloc (strlen(s) + 1);
    strcpy(node->value, s);
    node->next = list;
    return node;
}

```



CS61C L05 Introduction to C (pt 3) (31)

Garcia, Spring 2007 © UC Berkeley



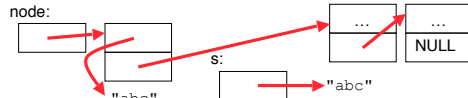
Linked List Example

```

/* 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;
}

```



CS61C L05 Introduction to C (pt 3) (32)

Garcia, Spring 2007 © UCB

Pointer Arithmetic Summary

- $x = *(p+1) ?$
 $\Rightarrow x = *(p+1) ;$
- $x = *p+1 ?$
 $\Rightarrow x = (*p) + 1 ;$
- $x = (*p)++ ?$
 $\Rightarrow x = *p ; *p = *p + 1 ;$
- $x = *p++ ? (*p++) ? *(p)++ ? *(p++) ?$
 $\Rightarrow x = *p ; p = p + 1 ;$
- $x = *++p ?$
 $\Rightarrow p = p + 1 ; x = *p ;$
- Lesson?



CS61C L05 Introduction to C (pt 3) (33)

Garcia, Spring 2007 © UCB

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

- Kim's melodious giddiness terrifies people, excepting zealous yodelers
- Kirby Messed Gigglypuff Terribly, (then) Perfectly Exterminated Zelda and Yoshi
- Killed meat gives teeth peace except zebra yogurt
- Kind Men Give Tense People Extra Zeal (for) Yoga
- Killing melee gives terror; peace exhibits Zen yoga
- Young Zebras Exclaim, "People Teach (Giraffes, Girls) Messy Kissing!" – Omar Akkawi
- "King me," Gina tells Perry, expert zebra yodeler – Diana Ko
- Kirk met Gibson's team, perilously expecting zealous youngsters – Diana Ko
- Kind Men Give Ten Percent Expressly Zee Yoorphans – Daniel Gallagher
- King Mel Gibson Tells People "Examine Ze Yoodle!" – Daniel Gallagher
- Kizzle Meh Gizzle The Pezzle Exizzle Zeh Yo! – Daniel Gallagher
- Killer Mechanical (Giraffe / Giant) Teaches Pet, Extinct Zebra, to Yodel – Larry Ly
- Killed Men Given Testosterone Perform Exceedingly Zealous Yoga –David Wu



CS61C L05 Introduction to C (pt 3) (34)

Garcia, Spring 2007 © UCB