

CS 537

Section 3: Memory in C

Michael Swift

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

1

Quiz

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

2

Project 0 problems

- Memory allocation for loop

```
for (i=0; i < 10; i++) {  
    p = malloc(10);  
    use(p);  
    free(p);  
}
```

```
p = malloc(10*sizeof(int));  
for (i=0; i < 10; i++) {  
    use(p);  
}  
free(p);
```

- use of fgets()

```
#define MAX_LINE 10  
char s[MAX_LINE];  
fgets(s, MAX_LINE, file)
```

```
#define MAX_LINE 10  
char s[MAX_LINE];  
fgets(s, MAX_LINE+1, file)
```

- Pointer arithmetic

```
int * arr;  
arr = malloc(10*sizeof(*int));  
*(arr+10) = 4;
```

```
int * arr;  
arr = malloc(10*sizeof(*int));  
arr[10] = 4;
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

3

Project 0 problems

- Malloc() does not zero memory; use calloc() or call memset();
- Boundary conditions
 - Many programs failed to read the last word on the last line.

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

4

Memory Debugging

```
int main(int argc, char * argv[])
{
    char * x;
    x = malloc(10);
    strcpy(x,argv[1]);
    printf("Hello, world: %s\n",x);
    free(x);
    strcpy(x,argv[2]);
    printf("Bye, world: %s\n",x);

    return(0);
}
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

5

What happens if you run this program?

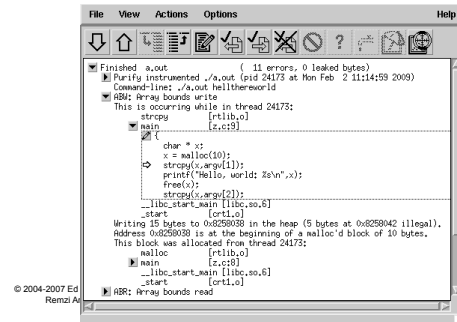
- It works correctly?
- It crashes?
- [try it]

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

6

Purify

```
manaslu(4)% purify gcc -g z.c
manaslu(5)% ./a.out hellthereworld
Hello, world: hellthereworld
Segmentation fault
```



© 2004-2007 Ed
Remzi Arpaci-Dusseau, Michael Swift

Valgrind

```
[swift] gcc -g z.c
[swift] valgrind ./a.out hellothereworld
==858== Memcheck, a memory error detector.
==858== ERROR
==858==
==858== Invalid write of size 1
==858== at 0x26DB0: strcpy+160 (in /usr/local/lib/
valgrind/x86-darwin/vgpreload_memcheck.so)
==858== by 0x1F84: main+50 (z.c:9)
==858== Address 0x3ec35a is 0 bytes after a block of size
10 alloc'd
==858== at 0x22E53: malloc+99 (in /usr/local/lib/
valgrind/x86-darwin/vgpreload_memcheck.so)
==858== by 0x1F6A: main+24 (z.c:8)
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

8

Pointers in C

- Pointers are addresses
 - `char * c = malloc(10 * sizeof(char));`
 - `c` now contains the **address** of some memory
- '*' operator returns what is at an address
 - `*c` returns the character at address `c`
- `p[n]` operator returns what is at address:
 - `p + n * sizeof(*p)` – the size of the type `p` points to
- two dimensional arrays:
 - `int **p;`
 - `p[n]` = what is at `p+n*sizeof(int *)`; call this `q`: a pointer to integers
 - `q[m]` = what is at `q + m*sizeof(int)`; an integer

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

9

Data types in C

- structures: like a class without functions

```
struct node {
    int value;
    char name[10];
};
struct node node_instance;
struct node * ptr_to_node;
```
- typedef: give a name to another type

```
typedef struct node node_type;
typedef struct node * ptr_node_type;
```
- Recursive structures

```
struct node {
    int value;
    struct node * next;
};
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

10

Rules for allocating memory

- Call `malloc()` for any data returned from a function:

```
char * reverse(char * line) {
    char tmp[100];
    for (int j=0, i = strlen(line); i >= 0; i--)
        tmp[j++] = line[i];
    return(tmp);
}
```
- Call `free()` when you are done:

```
char * output;
fgets(line, MAX_LEN, file);
output = reverse(line);
free(output);
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

11

Data Structures in C – Linked List

- Linked Lists

```
typedef struct node_s {
    char field_n;
    struct node *next;
} node, node_ptr;

node_ptr my_ptr, ptr;
struct header {
    node_ptr first, last;
} my_list;
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

12

Allocating a List

```
int i;
/* allocate the first list record */
my_ptr = (node_ptr)malloc( sizeof(node));
if (!my_ptr) exit(1);
my_list.first = my_ptr;
my_list.last = my_ptr;
my_ptr->field_n = 'A';
my_ptr->next = my_list.first; /* makes the list circular */
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

13

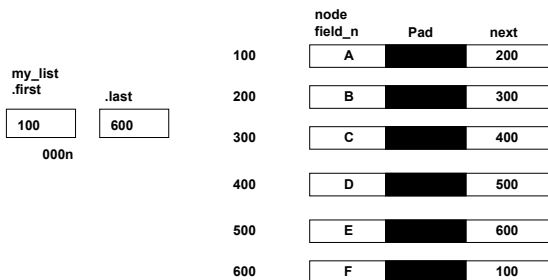
Allocating a List

```
/* allocate records 2 through 6 to the list */
for (i=2; i <= 6; i++)
{
    my_ptr = (node_ptr)malloc(sizeof(node));
    if(!my_ptr) exit(1);
    ptr = my_list.last; // find end of list
    my_list.last = my_ptr;
    ptr->next = my_ptr; // update ptr in former last
    my_ptr->field_n = ptr->field_n + 1; // move to next char
    my_ptr->next = my_list.first; // make circular again
}
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

14

Layout in Memory



Freeing a list

```
ptr = my_list.first;
while (my_list.first != NULL ) {
    my_ptr = my_list.first;
    my_list.first = my_ptr->next;
    if (my_list.last == my_ptr)
        my_list.last = NULL;
    free(my_ptr);
}
```

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and
Remzi Arpaci-Dusseau, Michael Swift

16

Memory Manipulation Functions

Header File:

```
<string.h>
```

```
void *memcpy (void *dest, const void *src, size_t n);
void *memmove(void *dest, const void *src, size_t n);
int memcmp (const void *s1, const void *s2, size_t n);
void *memset (void *s, int c, size_t n);
```

Memory Manipulation Functions

```
/* memcpy.c: memcpy example */
```

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

```
#include <string.h>
```

```
void main(void)
```

```
{
    char src[ ] = "*****";
    char dest[ ] = "abcdefghijklmnopqrstuvwxy01234";
    char *ptr;
    printf("destination before memcpy: %s\n", dest);
    ptr = (char *) memcpy(dest, src, strlen(src));
    if (ptr)
        printf("destination after memcpy: %s\n", dest);
    else
        printf("memcpy failed\n");
}
```

Memory Manipulation Functions

```
/* memmove.c memmove example */
```

```
#include <string.h>
```

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

```
void main(void)
```

```
{
    char dest [80] = "abcdefghijklmnopqrstuvwxy012345" ;

    printf ( "dest prior to memmove: %s\n", dest);
    memmove (&dest[5], &dest[26], 6);
    printf ("dest after memmove: %s\n", dest);
}
```

Memory Manipulation Functions

```
/* memset.c: memset example */
```

```
#include <string.h>
```

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

```
#include <mem.h>
```

```
void main(void)
```

```
{
    char buffer[ ] = "Hello world\n";

    printf("Buffer before memset: %s\n", buffer);
    memset(buffer, '*', strlen(buffer) - 1);
    printf("Buffer after memset: %s\n", buffer);
}
```

Memory Manipulation Functions

```
char *buf1 = "aaa", *buf2 = "bbb", *buf3 = "ccc";
int stat;
stat = memcmp(buf2, buf1, strlen(buf2));
if (stat > 0)
    printf("buffer 2 is greater than buffer 1\n");
else
    printf("buffer 2 is less than buffer 1\n");
stat = memcmp(buf2, buf3, strlen(buf2));
if (stat > 0)
    printf("buffer 2 is greater than buffer 3\n");
else
    printf("buffer 2 is less than buffer 3\n");
```