

Memory Allocation in C

C Programming and Software Tools

N.C. State Department of Computer Science

Computer Science
NC STATE UNIVERSITY

The Easy Way

- Java (JVM)
automatically
allocates and
reclaims
memory for
you, e.g...

Removed object is
implicitly reclaimed
(garbage collected)
when there are no
longer any references
to it

```
public class LinkedList {  
    ...  
    public void addFirst (Object obj) {  
        Node newNode = new Node();  
        newNode.data = ...;  
        newNode.next = first;  
        first = newNode;  
    }  
    public Object removeFirst() {  
        if (first == null)  
            throw new emptyException();  
        Object obj = first.data;  
        first = first.next;  
        return obj;  
    }  
    ...  
}
```

The Harder Way

C requires you to **manually** allocate and reclaim memory, e.g...

Programmer explicitly indicates there are no more references to the removed object

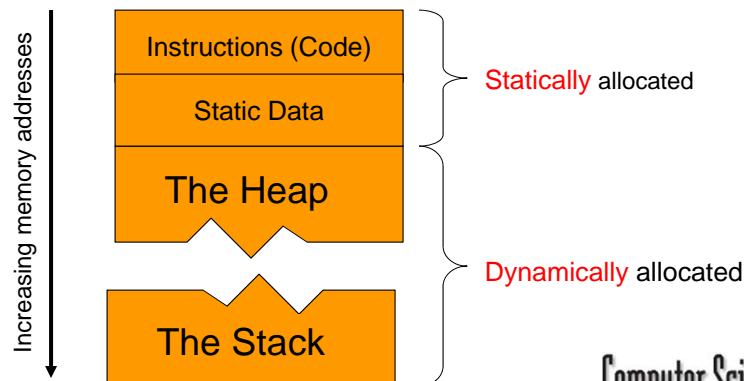
```
void addFirst (Object obj) {
    Node * newNode =
        (Node *) malloc (sizeof(Node));
    assert( newNode != NULL );
    newNode->data = ...;
    newNode->next = first;
    first = newNode;
}

Object removeFirst() {
    assert (first != NULL);
    Node * old = first;
    Object obj = first->data;
    first = first->next;
    free (old);
    return obj;
}
```

CSC230: C and Software Tools © NC State C

Memory Layout of a Program

- The **heap** is an area of *virtual memory* available for dynamic (runtime) memory allocation



CSC230: C and Software Tools © NC State Computer Science Faculty

Why Dynamic Memory Allocation?

- Don't know how much data will need to be stored until runtime; choices?

Choice 1: Declare **static array** of maximum size that could possibly occur

```
#define MAXCLASSSIZE 500
struct student { ...definition here... };
struct student students[MAXCLASSSIZE];

int i = 0;
while (more_students && (i < MAXCLASSSIZE))
    readstudents (students[i++]);
```

Why Dynamic ... (cont'd)

Choice 2: Declare **dynamic (auto) array** of specific size needed, at run time

```
int main (void) {
    int maxnum;
    printf("Number of students in class? \n");
    scanf("%d", &maxnum);
    struct student students[maxnum];

    int i = 0;
    while (more_students && (i < maxnum))
        readstudents (students[i++]);
}
```

Why Dynamic... (cont'd)

Choice 3: Allocate memory **dynamically** using a standard library function (**malloc** or **calloc**)

```
#include <stdio.h>
#include <stdlib.h>
...
int main(void) {
    struct student *sp;
    while (more_students) {
        sp = (struct student *)
            calloc (num, sizeof(struct student));
        if (sp != NULL)
            readstudents (sp);
    }
}
```

CSC230: C and Software Tools © NC State Computer Science Faculty

NC STATE UNIVERSITY


The **sizeof** Operator

- Not a function call; a **C operator**
 - returns **number of bytes** required by a data type
- Return value is of predefined type **size_t**

```
#include <stdlib.h>
size_t tsz1, tsz2, tsz3;
int a;
float b[100];
struct student { ...definition here... } st;

tsz1 = sizeof (a); /* 4 */
tsz2 = sizeof (b); /* ? */
tsz3 = sizeof (st); /* ? */
```

what are these sizes?



CSC230: C and Software Tools © NC State Computer Science Faculty

8 NC STATE UNIVERSITY

The `calloc()` Standard Library Function

Syntax:

```
void * calloc (size_t num, size_t sz)
```

Generic pointer, must be cast to type of result

OS allocates (`num * sz`) bytes of contiguous storage (all bytes initialized to zeros)

```
struct student * students;
students = (struct student *)
    calloc (num, sizeof(struct student));
int * ip;
ip = (int *) calloc (1, sizeof (int));
char *cp;
cp = (char *) calloc (1000, sizeof (char));
```

`calloc()` (cont'd)

- Return value is starting address of the storage allocated
- If not enough memory available, returns `NULL`
 - Could also be a unique pointer that could be passed to `free()`
 - **always** check for this error

*common source of bugs:
failure to check
return value*

```
cp = (char *) calloc (1000, sizeof (char));
if (cp == NULL) {
    printf("Cannot allocate memory; exiting\n");
    exit (-1);
}
```

The `malloc()` Std. Lib. Function

- Syntax: `void * malloc (size_t sz)`
- OS allocates `sz` bytes of contiguous storage
 - (uninitialized)
- Returns starting address of storage
 - If size is 0, returns NULL or unique pointer that can be freed

⚠ common source of bugs ⚠
malloc() does not initialize memory

```
students = (struct student *)  
           malloc ( num * sizeof(struct student));  
ip = (int *) malloc (sizeof (int));  
cp = (char *) malloc ( 1000 * sizeof (char));
```

CSC230: C and Software Tools © NC State Computer Science Faculty

Computer Science
11 NC STATE UNIVERSITY

The `realloc()` Std. Lib. Function

- Syntax: `void * realloc(void * ptr, size_t sz)`
- Grows or shrinks allocated memory
 - `ptr` must be dynamically allocated
 - Growing memory doesn't initialize new bytes
 - If can't expand, returns `NULL`
 - Old memory is unchanged
 - If `ptr` is `NULL`, behaves like `malloc`
 - If `sz` is `NULL`, behaves like `free`
 - Memory shrinks in place
 - Memory may NOT grow in place
 - If not enough space, will move to new location and copy contents
 - Old memory is freed
 - Update all pointers!!!

CSC230: C and Software Tools © NC State Computer Science Faculty

Computer Science
12 NC STATE UNIVERSITY

The `free()` Standard Library Function

- Syntax: `void free (void * ptr)`
 - no way to check for errors!
 - `ptr` **must** have been previously allocated by `malloc()` or `calloc()`
 - no need to specify **amount** of memory to be freed; why not?
- Frees (for other uses) memory previously allocated

```
free(students);  
free (ip);  
free (cp);
```

Why bother freeing up memory?

CSC230: C and Software Tools © NC State Computer Science Faculty



Dynamic Memory Allocation Mistakes

- These bugs can **really** be hard to find and fix
 - may run for hours before the bug pops up, and in a place that appears to have no relationship to the actual cause of the error

CSC230: C and Software Tools © NC State Computer Science Faculty

Mistake M1: Invalid Pointers

- Problems?

```
int i, j, result;
result = scanf ("%d %d", i, &j);
```

```
char *ptr;
...
ptr = 'A';
...
*ptr = 'B';
```

Invalid Pointers (cont'd)

- Problems?

```
int * f( void )
{
    int val;
    ...
    return &val;
}
```

why is this a problem?

Invalid Pointers (cont'd)

- Problems? Fix?

...dynamically allocate and construct a linked list...

...

```
/* now list is no longer needed,
 * free memory
 */
```

```
for (p = head; p != NULL; p = p->next)
    free(p);
```

why is this a problem?

Computer Science

17 NC STATE UNIVERSITY

M2: Not Initializing Memory

- Problems?

```
int * sumptr;
int ival[100] = { ...initial values here... };
int i;

sumptr = (int *) malloc ( sizeof(int) );

for (i = 0; i < 100; i++)
    *sumptr += ival[i];
```

Computer Science

18 NC STATE UNIVERSITY

M3: Stack Buffer Overflows

```
void bufoverflow (void)
{
    char buf[64];

    (void) gets(buf);
    return;
}
```

- Problems?
- One of the biggest sources of **security** problems

M4: Writing Past End of Dyn. Allocated Memory

```
int i, sz;
int *ip, *jp;

(void) scanf ("%d", &sz);
ip = (int *) calloc (sz, sizeof(int));
...check for errors here...

jp = ip;
for (i = 0; i <= sz; i++)
    (void) scanf ("%d", jp++);
```

why is this a problem?

M5: Freeing Unallocated Memory

Problems?

```
int i;
int *ip;

ip = &i;
...
free(ip);
```

why is this a problem?

CSC230: C and Software Tools © NC State Computer Science Faculty

Freeing Unallocated ...(cont'd)

• Problems?

```
int *ip;

ip = (int *) calloc (1000, sizeof(int));
...
free(ip);
...
free(ip);
```

CSC230: C and Software Tools © NC State Computer Science Faculty

M6: Memory Leaks

- Allocated memory is referenced using pointer returned by allocation
- If you lose pointers (free them, change to another address), you can no longer reference or free allocated memory
- Common problem in large, long-running programs (think: servers)
 - over time, memory footprint of program gets bigger, bigger, ...

M6: Memory Leaks

```
void leak (int n)
{
    int * xp;
    xp = (int *) malloc (n * sizeof(int));
    ...memory is used and then no longer needed...
    return;
}
```

why is this a problem?

M6: Memory Leaks

- Valgrind – software tool for detecting memory leaks on actual program executions
 - Compile with `-g` option
 - Arguments: `--leak-check=yes`

```
% gcc -Wall -std=c99 -g program.c -o program
% valgrind --leak-check=yes ./program
```

```
--15703== Memcheck, a memory error detector
--15703== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
--15703== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
--15703== Command: ./memory
--15703==
--15703== Invalid write of size 1
--15703== at 0x40055E: f (memory.c:9)
--15703== by 0x40058E: main (memory.c:15)
--15703== Address 0x4c41043 is 0 bytes after a block of size 3 alloc'd
--15703== at 0x4A0577B: calloc (vg_replace_malloc.c:593)
--15703== by 0x400523: f (memory.c:6)
--15703== by 0x40058E: main (memory.c:15)
--15703==
--15703== Invalid read of size 1
--15703== at 0x3AF5C480AC: fprintf (in /lib64/libc-2.12.so)
--15703== by 0x3AF5C4F409: printf (in /lib64/libc-2.12.so)
--15703== by 0x400579: f (memory.c:10)
--15703== by 0x40058E: main (memory.c:15)
--15703== Address 0x4c41043 is 0 bytes after a block of size 3 alloc'd
--15703== at 0x4A0577B: calloc (vg_replace_malloc.c:593)
--15703== by 0x400523: f (memory.c:6)
--15703== by 0x40058E: main (memory.c:15)
--15703==
String = abc
--15703==
--15703== HEAP SUMMARY:
--15703==   in use at exit: 3 bytes in 1 blocks
--15703== total heap usage: 1 allocs, 0 frees, 3 bytes allocated
--15703==
--15703== 3 bytes in 1 blocks are definitely lost in loss record 1 of 1
--15703== at 0x4A0577B: calloc (vg_replace_malloc.c:593)
--15703== by 0x400523: f (memory.c:6)
--15703== by 0x40058E: main (memory.c:15)
--15703==
--15703== LEAK SUMMARY:
--15703==   definitely lost: 3 bytes in 1 blocks
--15703==   indirectly lost: 0 bytes in 0 blocks
--15703==   possibly lost: 0 bytes in 0 blocks
--15703==   still reachable: 0 bytes in 0 blocks
--15703==   suppressed: 0 bytes in 0 blocks
--15703==
--15703== For counts of detected and suppressed errors, rerun with: -v
--15703== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 6 from 6)
```

Garbage Collection

- Some language run-time systems free up unused memory **automatically** for the programmer
 - accomplished through "reachability analysis"

Java

```
Student st = new Student("John Smith");  
...  
st = null; // space for student st is  
           // automatically reclaimed
```