CS 211 : Tues 01/23 (lecture 06)

Prof. Hummel
(he/him)

- **Topics: pointers, strings, memory management, stack vs. heap**

January 2024

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

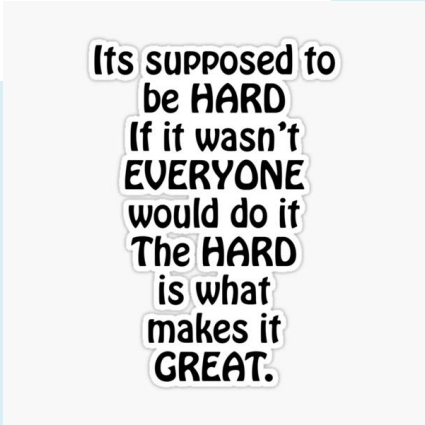
www.a-printable-calendar.com

Northwestern University

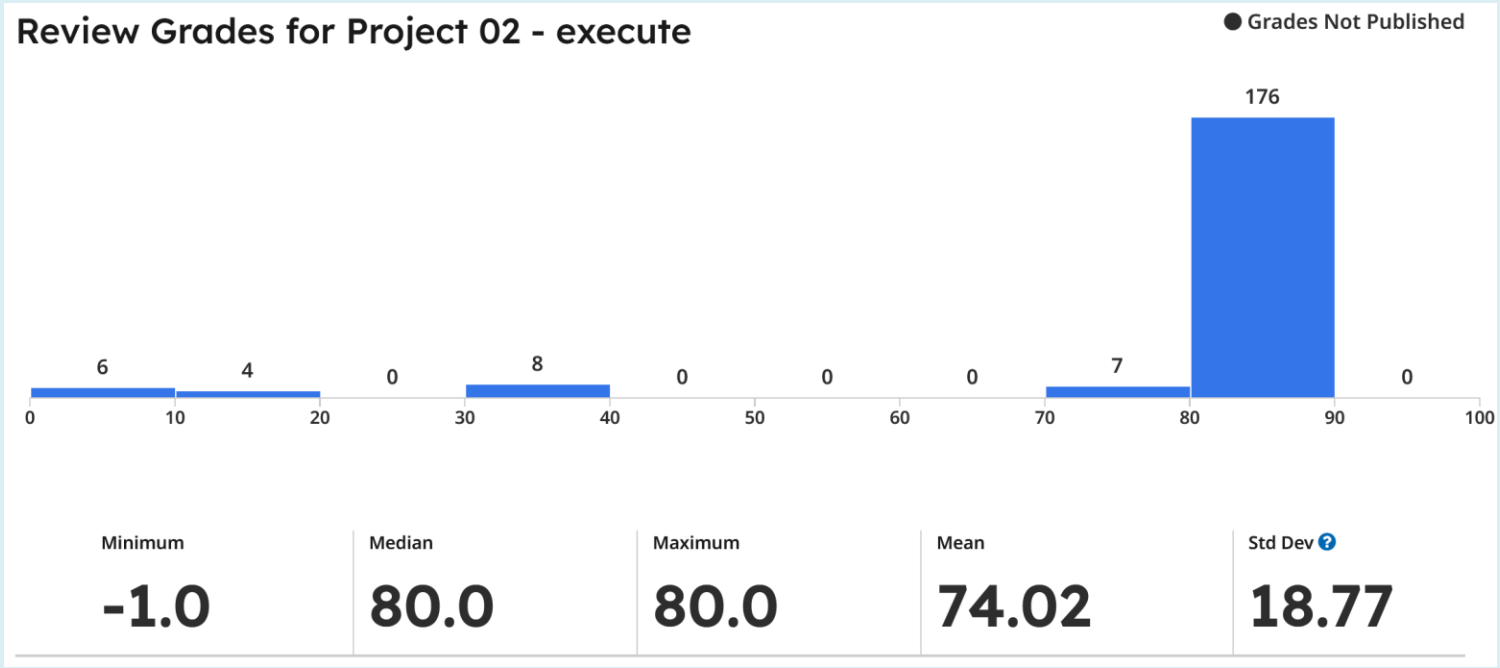
Notes:

- *Lecture slides available on Canvas*
- ***HW 03** due tonight @ 11:59pm*
- ***Project 03** due Friday @ 11:59pm, may be submitted up to 48 hours late. Gradescope will open soon (tonight?) for submissions*
- *Attendance, HW and Project scores posted to Canvas – please check for accuracy*

Project 02



- **Statistics:**
  - *205 students*
  - *201 submissions (98%)*
  - *176 finished with autograder score of 80/80 (86%)*



Don't do this...

```
void perform_int_binary_operation(int operator, ...)
{
  if (operator == 0)
  {
    ...
  }
  else if (operator == 1)
  {
    ...
  }
  else if (operator == 2)
  {
    ...
  }
  else if (operator == 3)
  {
```

```
if (rhs->types.expr->operator == 1){
  ram_write_cell_by_id(memory, ramVal, var_name);

    if ((ram_read_cell_by_id(memory,rhs->types.expr->rhs->element->e

     if ((rhs->types.expr->r
       if ((ram_read_cell_by
          printf("**SEMANTIC
           return false;
        }
       if((ram_read_cell_by_
          printf("**SEMANTIC
           return false;
        }



 if ((helper(stmt, memory, 1))
   return true;
 }
        else{
          return false;
        }
      }

      if (rhs->types.expr->rh

        struct RAM_VALUE *d =

        int a = d->types.i;
        int b = atoi(rhs->typ
        d->value_type = RAM_T
        d->types.i =  b - a;
        ram_write_cell_by_id(
        return true;
      }

       if (rhs->types.expr-

          struct RAM_VALUE *
           int a = d->types.

           int b = atoi(rhs-
           d->value_type = R

           d->types.i =  a -
           ram_write_cell_by
           return true;
        }
    }
     if ((ram_read_cell_by_id
       printf("**SEMANTIC ERR
        return false;
     }
    if ((ram_read_cell_by_id(
      printf("**SEMANTIC ERRO
       return false;
    }
      int b = atoi(rhs->types
      int a = atoi(rhs->types

      struct RAM_VALUE tval;
        tval.value_type = RAM
        tval.types.i = b - a;
```

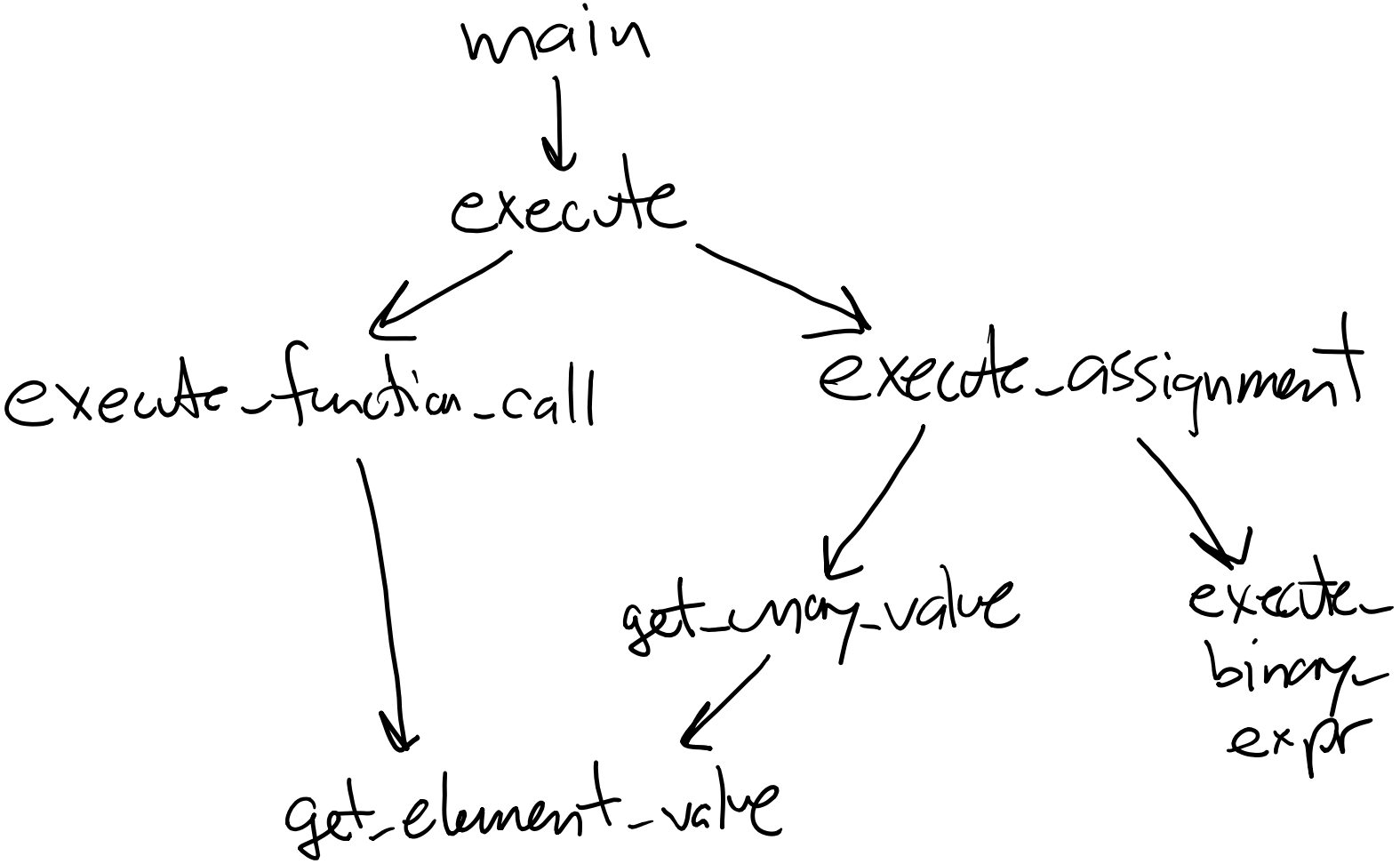
```
if (rhs->types.expr->operator == 2){

    if ((ram_read_cell_by_id(memory,rhs->types.expr->rhs->element->e

      if ((rhs->types.expr->rhs->element->element_
        if ((ram_read_cell_by_id(memory,rhs->types
           printf("**SEMANTIC ERROR: name '%s' is
            return false;
        }
       if((ram_read_cell_by_id(memory,rhs->types.
           printf("**SEMANTIC ERROR: name '%s' is
            return false;
        }
       ram_write_cell_by_id(memory, ramVal, var_n


if ((helper(stmt, memory, 2)) == 1){
  return true;
}
        else{
          return false;
        }
      }
      if (rhs->types.expr->rhs->element->element_t

        struct RAM_VALUE *d = ram_read_cell_by_id(
        int a = d->types.i;

        int b = atoi(rhs->types.expr->lhs->element
        d->value_type = RAM_TYPE_INT;
        d->types.i =  b * a;
        ram_write_cell_by_id(memory, *d, var_name)
        return true;
      }

       if (rhs->types.expr->lhs->element->elemen

          struct RAM_VALUE *d = ram_read_cell_by_
           int a = d->types.i;

           int b = atoi(rhs->types.expr->rhs->ele
           d->value_type = RAM_TYPE_INT;

           d->types.i =  b * a;
           ram_write_cell_by_id(memory, *d, var_n
           return true;
        }
    }
     if ((ram_read_cell_by_id(memory,rhs->types.ex
       printf("**SEMANTIC ERROR: name '%s' is not
        return false;
     }
    if ((ram_read_cell_by_id(memory,rhs->types.exp
      printf("**SEMANTIC ERROR: name '%s' is not d
       return false;
    }
      int b = atoi(rhs->types.expr->lhs->element->
      int a = atoi(rhs->types.expr->rhs->element->

      struct RAM_VALUE cat;
      cat.value_type = RAM_TYPE_INT;
      cat.types.i = b * a;
       ram_write_cell_by_id(memory, cat, var_name)
```

```
if (rhs->types.expr->operator == 3){
  ram_write_cell_by_id(memory, ramVal, var_name);
    if ((ram_read_cell_by_id(memory,rhs->types.expr->rhs-

      if ((rhs->types.expr->rhs->element->element_type ==
        if ((ram_read_cell_by_id(memory,rhs->types.expr->
           printf("**SEMANTIC ERROR: name '%s' is not def
            return false;
        }
       if((ram_read_cell_by_id(memory,rhs->types.expr->r
           printf("**SEMANTIC ERROR: name '%s' is not def
            return false;
        }



if ((helper(stmt, memory, 3)) == 1){
  return true;
}
        else{
          return false;
        }
      }

      if (rhs->types.expr->rhs->element->element_type ==
        struct RAM_VALUE *d = ram_read_cell_by_id(memory,
        int a = d->types.i;
        int b = atoi(rhs->types.expr->lhs->element->eleme

        d->value_type = RAM_TYPE_INT;
        d->types.i =  pow(b,a);
        ram_write_cell_by_id(memory, *d, var_name);
        return true;
      }

       if (rhs->types.expr->lhs->element->element_type

          struct RAM_VALUE *d = ram_read_cell_by_id(memo
           int a = d->types.i;

           int b = atoi(rhs->types.expr->rhs->element->e

           d->value_type = RAM_TYPE_INT;

           d->types.i =  pow(a,b);
         ram_write_cell_by_id(memory, *d, var_name);
           return true;
        }
    }
     if ((ram_read_cell_by_id(memory,rhs->types.expr->lhs
       printf("**SEMANTIC ERROR: name '%s' is not defined
        return false;
     }
    if ((ram_read_cell_by_id(memory,rhs->types.expr->rhs-
      printf("**SEMANTIC ERROR: name '%s' is not defined
       return false;
    }
      int b = atoi(rhs->types.expr->lhs->element->element
      int a = atoi(rhs->types.expr->rhs->element->element

      struct RAM_VALUE tval;
        tval.value_type = RAM_TYPE_INT;
        tval.types.i = pow(a,b);
       ram_write_cell_by_id(memory, tval, var_name);
```

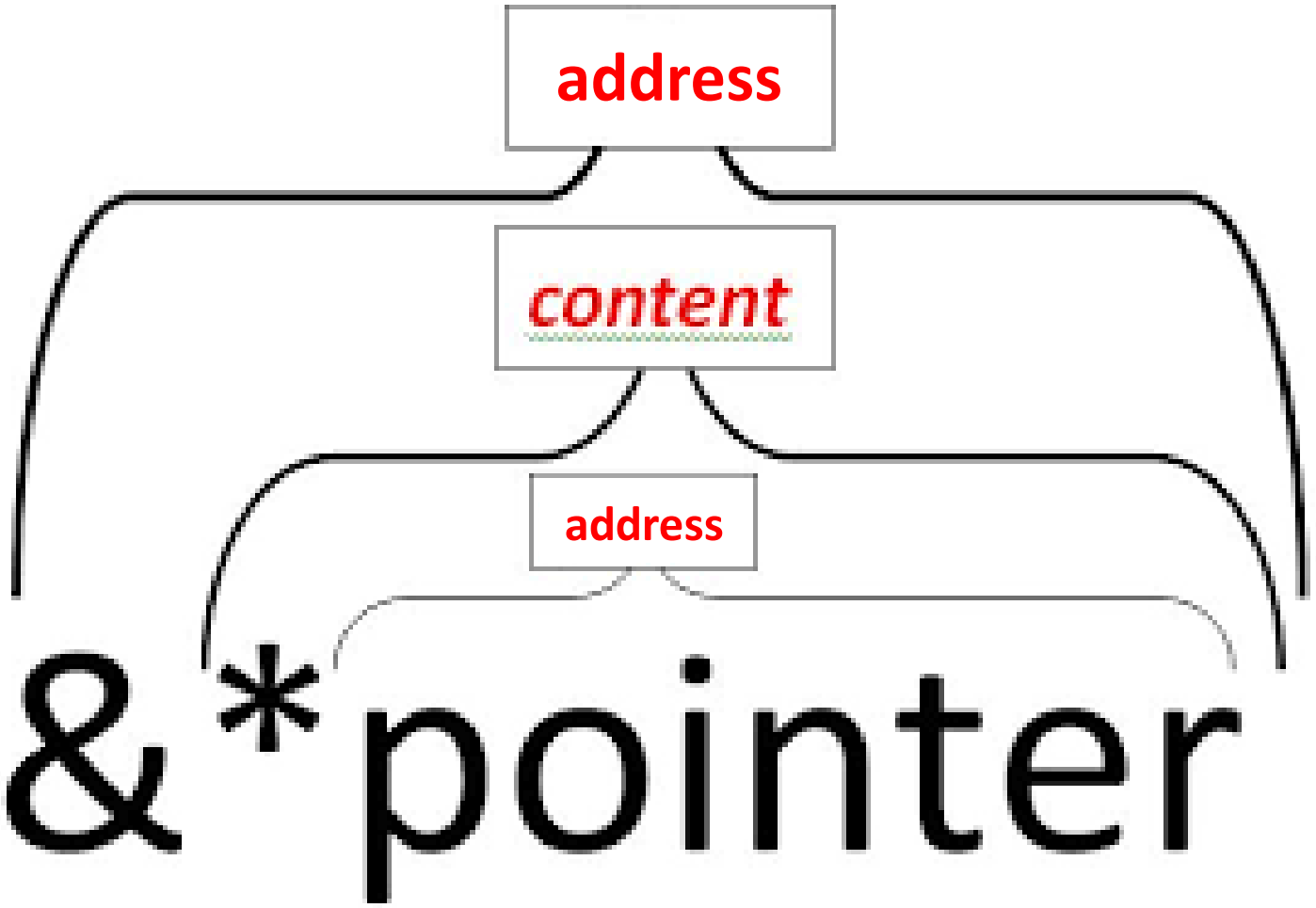
Call Graph

- **Call graph depicts how functions call each other… Good function design is critical to successful Software Engineering. Here's the call graph of our solution:**

Our project 02 solution



Project 03?


address
content
address
&*pointer

(1) What does this program output?

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

int main()
{
  char  buffer[10];
  char* cp;

  strcpy(buffer, "Project");

  cp = &buffer[4];

  *cp += 1;

  printf("%c\n", buffer[4]);
```

A) j

B) k

C) e

D) f

E) None of the above

(2) And what does this output?

- **Assume the user enters "northwestern" without the quotes…**

```
char* helper_function()
{
  char buffer[32];

  printf("enter a word> ");
  scanf("%s", buffer);

  return &buffer[4];
}
```

```
int main()
{
  char* cp;

  cp = helper_function();

  printf("%c\n", *cp);
```

A) h

B) i

C) A memory address

D) Crashes

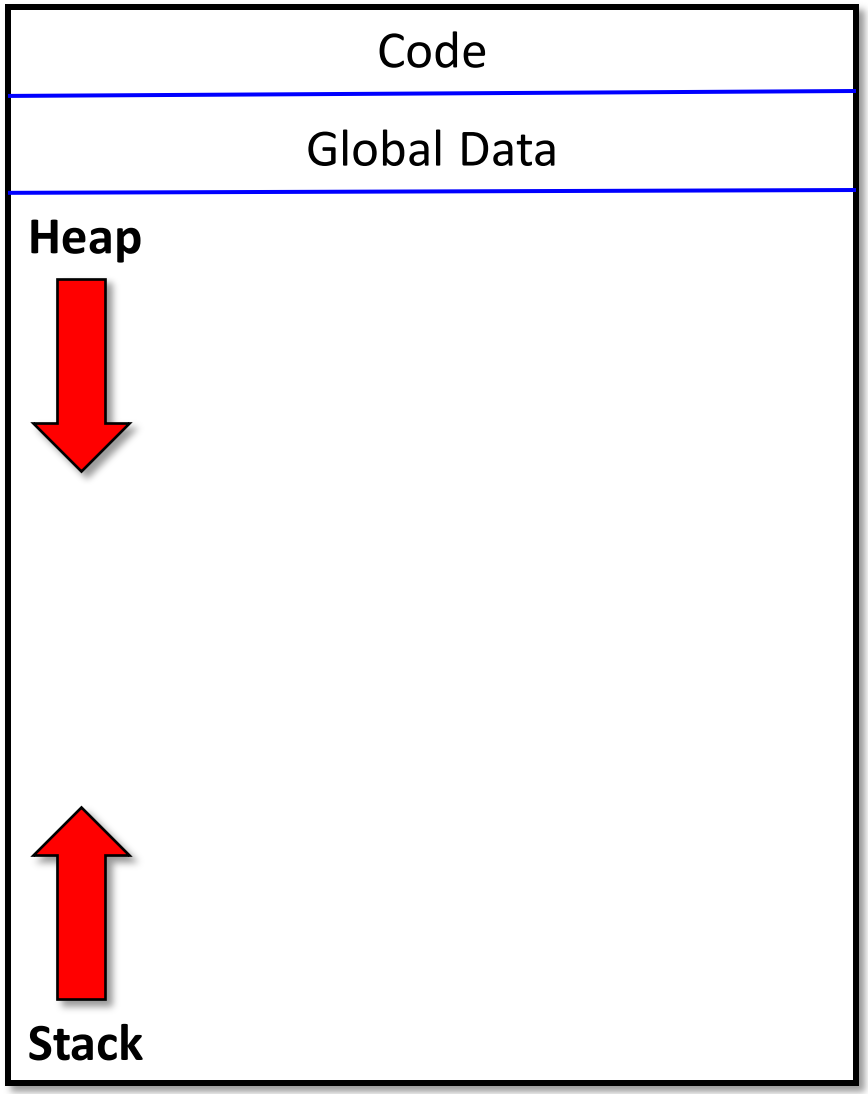
E) None of the above

RAM

- **Memory is divided into 4 sections…**

```
char* helper_function(int i)
{
  char buffer[32]
  .
  .
  printf("enter a word> ");
  .
  .
}
```

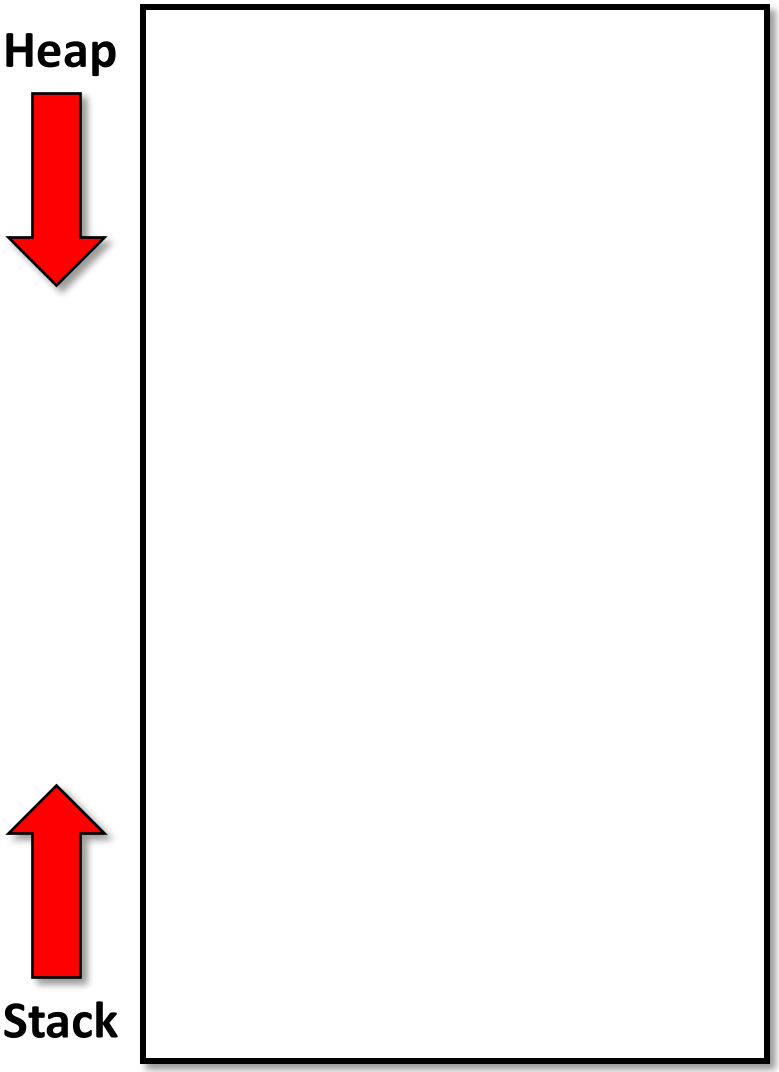
```
int main()
{
  int   x = 2
  char* cp;

  cp = helper_function(x);
```



Program execution: stack and heap

```
char* helper_function(int i)
{
  char buffer[32];

  printf("enter a word> ");
  scanf("%s", buffer);

  return &buffer[i];
}
```

```
int main()
{
  int   x = 2;
  char* cp;

  cp = helper_function(i);

  printf("%c\n", *cp);
```

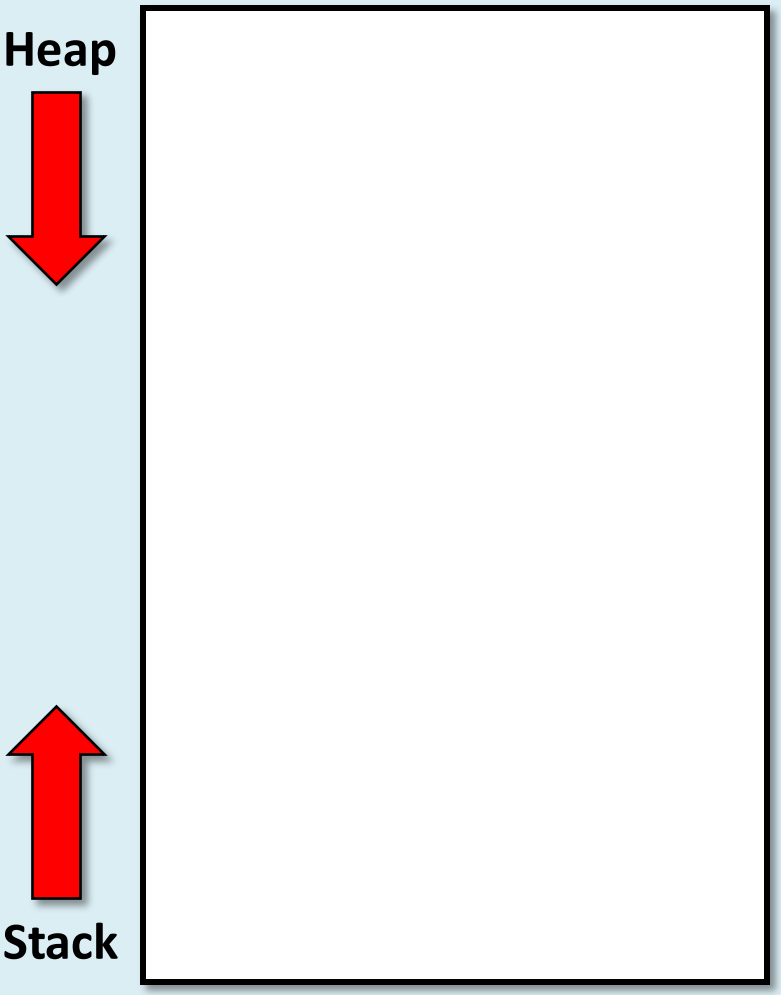


Dynamic memory allocation

- **Let's rewrite helper function using malloc( )…**

```
char* helper_function()
{
  char buffer[32];

  printf("enter a word> ");
  scanf("%s", buffer);

  size_t bytes = sizeof(char) *
                 (strlen(buffer) + 1);
  char* s = (char*) malloc(bytes);
  strcpy(s, buffer);
  return s;
}
```

```
int main()
{
  char* cp;

  cp = helper_function();

  printf("%c\n", *cp);
```



This code crashes… Where?

```
1  struct Student* create(char* name) {
2    struct Student* s;
3
4    size_t bytes = sizeof(struct Student);
5    s = (struct Student*) malloc(bytes);
6
7    strcpy(s->name, name);
8    s->year = 1;
9    s->gpa = 0.0;
10
11   return s;
12 }
```

```
struct Student {
  char*  name;
  int    year;
  double gpa;
};
```

```
14 int main()
15 {
16   struct Student* s;
17   char name[32];
18
19   printf("Enter student's name> ");
20   scanf("%s", name);
21
22   s = create(name);
23
24   printf("%s: %lf\n", s->name, s->gpa);
```

A) Line 7

B) Line 8

C) Line 9

D) Line 20

E) Line 24

Solution?

```
struct Student {
  char*  name;
  int    year;
  double gpa;
};
```

```
1  struct Student* create(char* name) {
2    struct Student* s;
3
4    size_t bytes = sizeof(struct Student);
5    s = (struct Student*) malloc(bytes);
6
7    strcpy(s->name, name);
8    s->year = 1;
9    s->gpa = 0.0;
10
11   return s;
12 }
```

```
14 int main()
15 {
16   struct Student* s;
17   char name[32];
18
19   printf("Enter student's name> ");
20   scanf("%s", name);
21
22   s = create(name);
23
24   printf("%s: %lf\n", s->name, s->gpa);
```

Memory management

- If you malloc( ) memory…
- You are supposed to free( ) that memory when you're done using it…

- Failure to do so is called a memory leak

Example

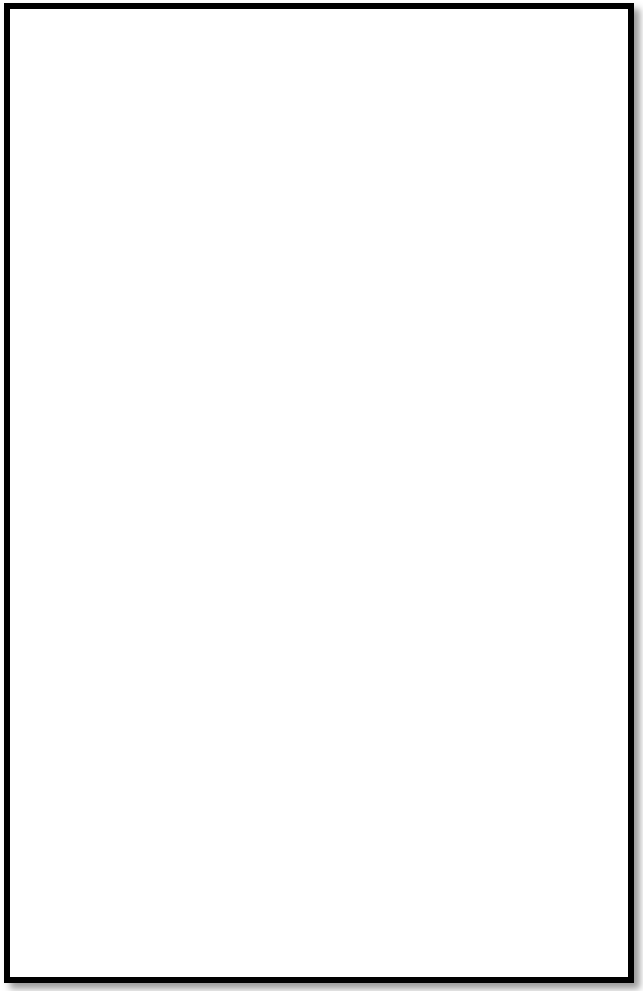
```
struct Student {
  char*  name;
  int    year;
  double gpa;
};
```

```
struct Student* create(char* name)
{
  struct Student* s;
  size_t bytes = sizeof(struct Student);
  s = (struct Student*) malloc(bytes);

  bytes = sizeof(char) * (strlen(name)+1);
  s->name = (char*) malloc(bytes);
  strcpy(s->name, name);
  s->year = 1;
  s->gpa = 0.0;

  return s;
}
```

```
int main()
{
  struct Student* s;
  char name[32];

  printf("Enter student's name> ");
  scanf("%s", name);

  s = create(name);

  printf("%s: %lf\n", s->name, s->gpa);

  free(s->name);
  free(s);
```

Heap

Stack



What should I be working on?

HW 03 *is due tonight…*

Project 03 *is due Friday night…*