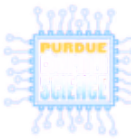
CS 240 - Programming in C

Spring 2025



CS 240 Example Exam 2 Questions

Revised: Monday, March 31, 2025

NOTE: You MAY see some of these questions on the exam. YOU WILL ALSO SEE QUESTIONS ON THE EXAM THAT ARE NOT LISTED HERE.

```
Dynamic Memory Allocation:
==========================

 - Given the following definitions:

     #include <malloc.h>

     int   *pi = NULL;
     float *pf = NULL;
     char  *pc = NULL;
     char   my_string[] = "Hello, World!";

   write statements to do the following memory operations:

     . reserve space for 100 integers and assign a pointer to that space to pi
     . reserve space for 5 floats and assign a pointer to that space to pf
     . unreserve the space that pi points to
     . reserve space for enough characters to hold the string in my_string and
       assign a pointer to that space to pc.  Copy my_string into that space.
     . free everything that hasn't been unreserved yet.

 - What happens if you reserve memory and assign it to a pointer named p
   and then reserve more memory and assign the new pointer to p?  How can
   you refer to the first memory reservation?

 - Does it make sense to free() something twice?  What's a good way to
   prevent this from happening?

Pointers to Structures:
=======================

 - Suppose p is a pointer to a structure and f is one of its fields.
   What is a simpler way of saying:

      x = (*p).f;

 - Given the following declarations and definitions:

     struct s {
       int x;
       struct s *next;
     };

   what will the following code print?
```

```
    struct s *p1 = NULL;
    struct s *p2 = NULL;
    struct s *p3 = NULL;
    struct s *p4 = NULL;
    struct s *p5 = NULL;

  p5 = malloc(sizeof(struct s));
  p5->x = 5;
  p5->next = NULL;
  p4 = malloc(sizeof(struct s));
  p4->x = 4;
  p4->next = p5;
  p3 = malloc(sizeof(struct s));
  p3->x = 3;
  p3->next = p4;
  p2 = malloc(sizeof(struct s));
  p2->x = 2;
  p2->next = p3;
  p1 = malloc(sizeof(struct s));
  p1->x = 1;
  p1->next = p2;

  printf("%d %d\n",
         p1->next->next->next,
         p2->next);
```

 - Be ready to write any of the subroutines in Homeworks 5 through 9 given their description.

Pointers to Pointers:

 - Write a subroutine called do_allocate that is passed a pointer to the head pointer to a list of block structures: do_allocate(struct block **). If the head pointer is NULL, do_allocate should allocate a new struct block and make the head pointer point to it. If the head is not NULL, the new struct block should be prepended to the list and the head pointer set to point to it.

 - Write a subroutine called my_free that will accept a pointer to a pointer of some arbitrary type and:
   . free the space pointed to by the pointer
   . set the pointer to NULL

Pointers Inside Structures:

 - Given the following declaration:

```
    struct employee
    {
      char *name;
      char *title;
      int   id;
    };
```

  write a subroutine called create_employee that accepts two string parameters for the new name and title and one integer parameter for the ID. It should return a newly allocated Employee structure with all of the fields filled in.

- Write a subroutine called fire_employee that accepts a pointer to pointer to struct employee, frees its storage and sets the pointer that points to the storage to NULL.

Recursion

- Create a recursive function to compute the factorial function.

- Create a recursive function to compute the Nth element of the Fibonacci sequence: 0 1 1 2 3 5 8 13 21 34 55 ...

- Implement a recursive list search. e.g. each function call should either return the list node that it's looking at because it matches the search item or it should return the value from calling itself on the next item in the list.

Trees

- Building
- Searching
- Traversal
- Deletion

Anything involving linked lists and trees - review your homeworks and slides.