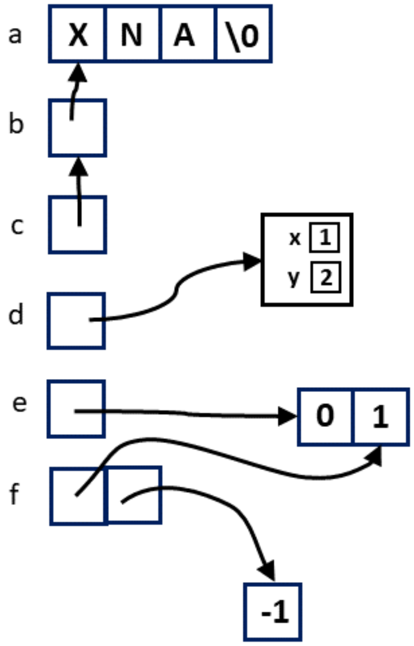
CPSC 223 - Fall 2017 - Exam #1

Problem 0 (1 point): Write your name on the *back* of this exam package. What did Prof. Glenn do this morning before the exam?

Problem 1 (20 points): Write the declarations and initializations to reflect the memory diagram shown below. For this and subsequent diagrams, there should be no local variables other than `a`, `b`, `c`, `d`, `e`, and `f`, everything not labelled with a local variable name is dynamically allocated, the numeric values are `int`s, the character values are `char`s, there is a `struct point` with `int` fields `x` and `y`, and you may assume that the appropriate header files have been included.



Problem 2 (5 points): Write the calls to `free` required so that there are no memory leaks after the local variables in the diagram from Problem 1 go out of scope (again, assume that there are no other variables aside from those shown in the diagram).

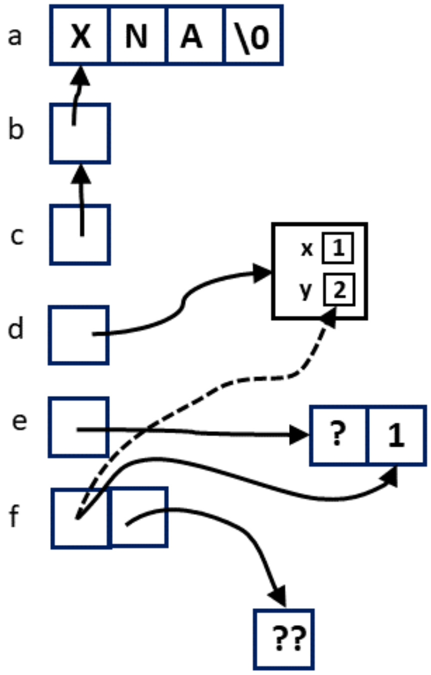
Problem 3 (12 points): Assuming that local variables `a` through `f` have been set up according to the diagram in Problem 1, indicate for each of the following pairs of statements whether one of the statements is invalid (will not compile without warnings or errors), or if both are valid then give the output of the `printf` or state that the behavior is undefined (for example, when pointers are used dangerously). Assume the results of the valid statements are cumulative and ignore the effects of the invalid statements.

```
a) (*b)++;
   printf("%s\n", a);

b) b++;
   printf("%c\n", *b);

c) c = &a;
   printf("%c\n", c[0]);

d) d->x = f[1];
   printf("%d\n", d->x);

e) *f = e;
   printf("%d\n", *f[0]);

f) f[1]--;
   printf("%d\n", *f[1]);
```

Problem 4 (6 points): Assuming that local variables `a` through `f` have been set up according to the diagram below, write the statements to

(a) set up the pointer shown with the dashed line

(b) copy the value marked "??" (whatever it is) to the location marked "?".



Problem 5 (16 points): The function `count_spaces` in the program below is intended to count the leading and trailing whitespace characters in the string passed to it; the counts will be returned through (simulated) reference parameters. As written, the function does not work; `main` prints `leading=0 trailing=0` for any command-line argument passed to it. Edit the program to make the `count_spaces` function work correctly and to make `main` call it correctly. You may edit (delete, modify, or insert) no more than eight lines total (and you may edit fewer).

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

void count_spaces(char  *s , int   leading, int   trailing);

int main(int argc, char **argv)
{
  if (argc > 1)
    {
      int leading = 0, trailing = 0;

      count_spaces(   argv[1],   leading,   trailing);

      printf("leading=%d trailing=%d\n",   leading,   trailing);
    }
}

void count_spaces(char   *s, int  leading, int  trailing)
{
  int len = strlen(  s  );

  char *start = s;
  while (isspace(  *start  ))
    {
      start++;
    }
  leading = start - s;

  char *end = s + len;
  while ( 1 != 0                                                 )  // FIX THIS TOO!
    {
      end--;
    }
  trailing = len - (end - s);
}
```

Problem 6 (20 points):

(a) Complete the following function `increment` that takes two arguments: an array of `int`s and its size. The function returns a new array of the same size as the given array with the values in the new array equal to one more than the corresponding value in the one passed in. For example, if `a` is {`1, 2, 3`} then `increment(a, 3)` returns {`2, 3, 4`}. For each blank in the function choose one of items A-S to fill the blank and write the corresponding letter. Each item may be used zero, one, or more times. You may not add other code.

```
_____ increment(_____ a, _____ n)
{
  ______ result = ______(______(______) * ______);

  for (int i = 0; i < ______ ; i++)
    {
      ______ = _____ + ______;
    }

  return ______;
}
```

```
A: int               J: *a
B: int *             K: a++
C: void              L: a->i
D: void *            M: a[i]
                     N: *result
E: 1                 O: result->i
F: a                 P: result[i]
G: i
H: n                 Q: malloc
I: result            R: free
                     S: sizeof
```

(b) Add some appropriate error-checking to the `increment` function – there are some implicit preconditions and you should add a check for at least one of those preconditions and return `NULL` if it is not met. Add your code to what you filled out above.

(c) Show below how to modify the header of `increment` and the line of code in the body of the `for` loop to make a function `apply` that behaves like `increment`, except the values in the new array are determined by passing each value in the old array to a function that is passed as the third argument to `apply`, so `apply(a, n, add_one)` would have the same effect as `increment` if `add_one` is defined as

```
int add_one(int n) { return n + 1; }
```

Problem 7 (20 points):

(a) Consider a plist ADT with the following functions (note the addition of plist_remove_end).

```
plist *plist_create();
void plist_destroy(plist *l);
int plist_size(const plist *l);
bool plist_add_end(plist *l, const point *p);
void plist_remove_end(plist *l);
void plist_get(const plist *l, int i, point *p);
bool plist_contains(const plist *l, const point *p);
void plist_fprintf(FILE *stream, const char *fmt, const plist *l);
```

(i) Complete the function called truncate_plist that has a pointer to a plist as its parameter and removes the second half of the points from the list (rounding down the number to remove if the number of points is odd). The plist structure is opaque, so truncate_plist cannot access its members directly.

```
void truncate_plist(                    )
{
  // compute the number of points to remove
  int num =                                ;

  // remove that many points
  for (int i = 0; i < num; i++)
    {

    }
}
```

(ii) What is the asymptotic running time of your truncate_plist if plist is implemented using a dynamically allocated array?

(iii) What is the asymptotic running time of your truncate_plist if plist is implemented using a doubly-linked list?

(b) Write plist_truncate to have the same effect as truncate_plist, except plist_truncate is part of the plist module (written in plist.c so has access to the members of the structure). Assume that plist is defined as

```
struct plist
{
  int capacity;
  int size;
  point *items;
};
```

with the appropriate typedef. Your implementation should run in $O(1)$ time (do not resize the array). You may assume that the pointer passed to plist_truncate points to a valid plist.

(c) Unscramble the `plist_truncate` function written as part of the `plist` module, where `plist` is implemented as a doubly-linked list with dummy head and tail nodes and structs defined as follows

```
typedef struct plist_node {              struct plist {
  point data;                              int size;
  struct plist_node *next;                 plist_node *head;
  struct plist_node *prev;                 plist_node *tail;
} plist_node;                            }
```

with the appropriate `typedef` for `plist`. Write the sequence of numbers for the lines of code to complete the body of the function. Some numbers may not be used and some may be used more than once.

```
void plist_truncate(plist *l)
{
  if (l != NULL)
    {
      int kill = l->size / 2;
      l->size -= kill;

      // move curr to first node to remove (select/unscramble 1-9)

      // link around second half of list (10-15)

      // free the removed nodes (16-23)

    }
}
```

```
1)      plist_node *curr = l->head;
2)      plist_node *curr = l->head->next;
3)      plist_node *curr = l->tail;
4)      plist_node *curr = l->tail->prev;
5)      for (int i = 0; i < kill; i++) {
6)          curr = *curr;
7)          curr = curr->prev;
8)          curr = curr->next;
9)      }

10)     l->tail = curr->prev;
11)     l->tail = curr->prev->next;
12)     l->tail->prev = curr->prev;
13)     curr->prev = l->tail;
14)     curr->prev = l->tail->prev;
15)     curr->prev->next = l->tail;

16)     while (curr != l->tail) {
17)     while (curr != NULL) {
18)       free(curr);
19)       free(curr->prev);
20)       curr = curr->next;
21)       plist_node *temp = curr->next;
22)       curr = temp;
23)     }
```