csc/cpe 357 C Quiz

Spring 2021

Name: Sereen Benchohra

User ID (email): sbenchoh@calpoly.edu

Rules:

- Do all your own work. Nothing says your neighbor has any better idea what the answer is. Plus, this quarter working from home, you don't have a neighbor.
- This exam is open book, notes, internet, and anything inanimate.
- If you unsure if a resource is animate, ask it. If it answers, it is.
- Do not discuss this exam outside of class until after 11:59pm, Friday, April 23rd.
- If you need to add a picture or any other "extra" thing, put a note in the text box and submit your picture via handin along with the exam.
- Submit this exam via `handin` to `c-quiz` by 23:59 tonight. (I'm not expecting you to spend all day on this, but you get to chose when.)
- The programming problems should be submitted in the specified files.
- As insurance, you may wish to include plain text versions of the short-answer problems, too.
-

Suggestions(mostly the obvious):

- When in doubt, state any assumptions you make in solving a problem. **If you think there is a misprint, ask me.**
- **Read the questions carefully.** Be sure to answer all parts.
- Identify your answers *clearly.*
- Watch the time/point tradeoff: 50pts / 50 min works out to 60.0s/pt.
- Problems are not necessarily in order of difficulty. They are in the order in which they fit.
- Be sure you have all pages. Pages other than this one are numbered "n of 7".

Encouragement:

- Good Luck!

Problem	Possible	Score
1	6	
2	6	
3	8	
4	15	
5	15	
Total:	**50**	

`All programmers are optimists. Perhaps this modern sorcery especially attracts those who believe in happy endings and fairy godmothers. Perhaps the hundreds of nitty frustrations drive away all but those who habitually focus on the end goal. Perhaps it is merely that computers are young, programmers are younger, and the young are always optimists. But however the selection process works, the result is indisputable: ‘‘This time it will surely run,’’ or ‘‘I just found the last bug.’’`

`-- Frederick Brooks, ‘‘The Mythical Man Month’’` — /usr/games/fortune

Answer clearly, concisely, and (where possible) correctly:

1. (6) While we're on the subject of macros, the C stdio library contains the following definitions:

   int fputc(int c, FILE *stream);
   int putc(int c, FILE *stream);

   Each one writes a character to the given stream. The *only* difference between the two is that `fputc()` is guaranteed to be a function while `putc()` may be implemented as a macro. What is the purpose of having both?

   The purpose of having one call implemented as a function and is the other one be sometimes implemented as a macro is to allow the user to decide which definitions they need based on their needs. The macro version is useful is the user wants to use the function for multiple types and increase the speed of exectution and is useful where small code appears many times. Likewise, there are downsides to using a macro so hence why a user might use a function since type checking is down, there are no side effects and is useful where larger code appears many times. In addition, it more secure since it checks for compiler errors, however speed of execution is slower. Hence having both allows the user options based on what they want in their program.

2. (6) It is said, "Never include anything in a header file that either allocates memory or defines (rather than declares) a function." Why would this be a problem?

   The reason why it is not recommended to include anything in the header file that either allocates or defines a function is because it would mean the function would be defined multiple times, which is an error. In addition you would never want to allocate memory in the header because you would allocate memory each time you include the header, which would have   pretty severe and unexpected side-effect like using up all your memory or not knowing how much is allocated so free memory is inaccurate leading to memory leaks.

3. (8) Implement a C function `mean()` that takes an array of integers `A` and its length `len` and returns the average of all elements of `A` as a double. You may assume that you will receive proper arguments and that `A` has at least one element.

   This is the space I would've given if this were an in-person exam. Submit your code as `p3.c`.

4. (15) Implement a robust version of the C library function `strrchr(3)`:

   Name:

   ```
   char *strrchr(const char *s, int c);
   ```

   Description:

   ```
   The strrchr() function returns a pointer to the last occurrence of the
   character c in the string s.
   ```

   Return Value:

   ```
   strrchr() function returns a pointer to the matched character or NULL if the
   character is not found.  The terminating null byte is considered part of the
   string, so that if c is specified as '\0', this function returns a pointer
   to the terminator.
   ```

   Write robust code (even though the library version is fragile). That is, return `NULL` on failure, but do not crash. Do not use any of the C library's string functions. Think before you write anything.

   This is the space I would've given if this were an in-person exam. Submit your code as `p4.c`.

5. (15) Given the following structure definition:

```
typedef struct node_st {
    int data;
    struct node_st *next;
} Node;
```

Write a C function, `merge_lists()`, that takes pointers to two NULL-terminated linked lists made up of these structures, sorted in ascending numerical order, and merges them together into one sorted list. Return a pointer to the head of the merged list.

This is the space I would've given if this were an in-person exam. Submit your code as `p5.c`.

```
Node *merge_lists(Node *a, Node *b) {
```

(Continued on the following page...)

Optional extra space for problem 5.

```
}
```

Useful Information

Selected Useful Prototypes

```
void *   calloc(size_t nmemb, size_t size);
int      fclose(FILE *stream);
FILE *   fdopen(int fildes, const char *mode);
int      feof( FILE *stream);
int      fgetc(FILE *stream);
char *   fgets(char *s, int size, FILE *stream);
FILE *   fopen(const char *path, const char *mode);
int      fprintf(FILE *stream, const char *format, ...);
int      fputc(int c, FILE *stream);
int      fputs(const char *s, FILE *stream);
void     free(void *ptr);
FILE *   freopen(const char *path, const char *mode, FILE *stream);
int      getc(FILE *stream);
int      getchar(void);
char *   gets(char *s);
char *   index(const char *s, int c);
int      isalnum(int c);
int      isalpha(int c);
int      isascii(int c);
int      isblank(int c);
int      iscntrl(int c);
int      isdigit(int c);
int      isgraph(int c);
int      islower(int c);
int      isprint(int c);
int      ispunct(int c);
int      isspace(int c);
int      isupper(int c);
int      isxdigit(int c);
void *   malloc(size_t size);
void     perror(const char *s);
int      printf(const char *format, ...);
int      putc(int c, FILE *stream);
int      putchar(int c);
int      puts(const char *s);
void *   realloc(void *ptr, size_t size);
int      rand(void);
int      random(void);
char *   rindex(const char *s, int c);
int      snprintf(char *str, size_t size, const char *format, ...);
int      sprintf(char *str, const char *format, ...);
char *   strcat(char *dest, const char *src);
char *   strchr(const char *s, int c);
int      strcmp(const char *s1, const char *s2);
char *   strcpy(char *dest, const char *src);
int      strlen(const char *s);
char *   strerror(int errnum);
char *   strncat(char *dest, const char *src, size_t n);
int      strncmp(const char *s1, const char *s2, size_t n);
char *   strncpy(char *dest, const char *src, size_t n);
char *   strrchr(const char *s, int c);
char *   strstr(const char *haystack, const char *needle);
int      ungetc(int c, FILE *stream);
int      tolower(int c);
int      toupper(int c);
```