csc/cpe 357 C Quiz

Fall 2020

Name: Sereeen 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, October 9th.
- 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.)
- As insurance, you may wish to submit the programming portions as separate files. If you do so, please leave the name of the file in the fill-in box.
-

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: 60pts / 50 min works out to 50.0s/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 8".

Encouragement:

- Good Luck!

Problem	Possible	Score
1	5	
2	5	
3	5	
4	5	
5	10	
6	15	
7	15	
Total:	**60**	

```
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. (5) Why is it important always to use the `sizeof()` operator when allocating space for a structure?

It is important to always use the sizeof when allocating space for a structure because it automatically computes the size of the object and we would not have to worry about calculating the size of the object again in the case if the type changes which would greatly decrease the risk of errors and memory issues. In addtion, this is important since most of the time the time the size is determined at complie which sizeof() handles.

2. (5) Given an implementation of `fw` implemented (at the last minute, of course) using an ordinary binary tree and the two invocations:

   a) `% fw /usr/share/dict/words /usr/share/man/*/*`
   b) `% fw /usr/share/man/*/* /usr/share/dict/words`

   Which would you expect to complete its execution more quickly and why?

Option a would complete faster because option b would have to go through more levels and directions to find the location, and if it did not find the location on the first go and would have to back a lot more often causing the perfomance to be affected more.

3. (5) What is the meaning of a *static* declaration in C?

The meaninig of a static declaration in C has two uses for global and local. Static is used with global variables and functions to set their scope in the file. Static variables that are used in local scope are used to store the variable in the statically allocated memory instead of the automatically allocated memory and remain in existence rather than coming and going each time.

4. (5) The stdio function `getchar(3)` reads a character from stdin and, on success, returns it. On failure, `getchar(3)` returns `EOF` which is defined to be -1. In spite of the fact that it's reading chars, `getchar(3)` returns an int. Why? (and explain)

The reason why getchar returns an int rather than a char is because char is a type of int, but it has more limited range to represent the char characters (chars are represented by numbers). As a result, the char type does not have the EOF, which is a value needed to indicate the end of fille. Hence why int is used since it has all the char types and the EOF, increasing the portability of the code. In additon, if we used char, it could be problematic since some machines might read it as unsigned char or signed char, decreasing portability and increasing the risk of errors.

5. (10) Implement a C function `sum()` that takes two integers `x` and `y` such that $x \leq y$ and returns the sum of all integers from x to y, inclusive. You may assume that you will receive proper arguments and do not need to be concerned about overflow.

```
int sum(int x, int y){
```

```
int sum;

do {

   sum +=  x;
   x+= 1;

    }while(x <= y);

return sum;
```

```
}
```

6. (15) Implement a robust version of the C library function `strspn()`:

   Name: `size_t strspn(const char *s, const char *accept);`

   Description: `The strspn() function calculates the length of the initial segment of s which consists entirely of characters in accept.`

   Return Value: `The strspn(3) function returns the number of characters in the initial segment of s which consist only of characters from accept, or −1 if it is unable to complete its task.`

   Write robust code (even though the library version is fragile). That is, return −1 on failure, but do not crash. Do not use any of the C library's string functions. Think before you write anything.

   `size_t strspn(const char *s, const char *accept){`

```
size_t numAccepted;
numAccepted = 0;  /*start with 0 characters accepted */
int i, j;
i = 0;
while(accept[i] != '\0' ) {  /* loops untill the last character is read */

 /* goes through s and compares each charto the accept char if so increase the counter */

    for(j = 0; s[j] != '\0'; j ++) {
      if(s[j] == accept[i]) {
          numAccepted++;
          i++;
    }

    i++;

}

/*  returns the number of char accepted, returns 0 if there was not anything that was accept  cannot
return -1 since size_t is the return type */

return numAccepted;
```

7. (15) The standard unix command `tac(1)` reads a file and prints out all its lines in reverse order (it's `cat` backwards, after all). Given the function, `char *rll(FILE *whence)`, that reads a line of arbitrary length from the given FILE * and returns a pointer to it in a newly-allocated buffer or `NULL` on end-of-file, write `tac(1)`. In particular:

   - `tac` works as a filter, copying its standard input to its standard output (in reverse order).
   - `tac` should make sure to deallocate any memory allocated during its run.
   - `tac` should indicate success or failure with its exit status.

   Example:

   ```
   $ cat input
   one
   two
   $ tac < input
   two
   one
   ```

   Write robust code.

```
int main( ) {

char *line, *listofLines[];
int listLength, i, j;
listLength = i= j = 0
line = rll(stdin);    /* read an initial line */

/* now, keep reads until there are no lines */

while ( (NULL != line) )
    listLength++;  /* gets length for the array */

listofLines = (char **)calloc(listLength, sizeof(char*)); /*allocates memory for array);

while ( (NULL != line) ) {
    listofLines[i] = line;  /* stores line in array */
    i++;
}

for(j = listLength - 1; j > 0; j--) {
   fputs(listofLines[i], stdout); /* prints in reverse order */
    free(line);

}

free(listofLines);



return 0;

}
```

(Continued on the following page. . .)

Optional extra space for problem 7.

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);
```