CSE 2451 Final Exam

Name:

Introduction

duration: 105 minutes
you are allowed to use 1 A4-page cheat-sheet

True/False (15 pts)

True or False for the following statements:

1. include guard prevents the same header file from being copied to a source file multiple times:
2. -g tells gcc to produce symbolic debug information in the object file for downstream tools:
3. "#define MAGIC_NUMBER 10" defines a macro named as MAGIC_NUMBER:
4. void foo() is the function header for a function that can't take any input arguments and can't return any value:
5. If a file scope variable x is declared with "int x;" it has external linkage and static storage duration by default:

Multiple Choice (60 pts)

6. Assuming you have an integer pointer x (int* x) pointing to an address returned from malloc(sizeof(int)) function, and a floating-point variable y (double y). The input from stdin follows the "[integer number] [floating-point number]" format for each line, e.g., the input from stdin might look like:

```
10 3.5
2  4.55
12 3.14
```

which of the following statments is using fscanf() properly to extract the values in one line of the input from stdin?

```
A. fscanf(stdin, "%d %lf", &x, y);
B. fscanf(stdin, "%d %lf", &x, &y);
C. fscanf(stdin, "%d %lf", x, &y);
D. fscanf(stdin, "%d %lf", x, y);
```

7. Which of the following array initialization is incorrect?

```
A. int arr[] = {1,2,3};
B. char arr[3] = {'c'};
C. char arr[4] = "abc";
D. int arr[2] = {1,2,3};
```

8. Given the fact that int x = 3 and int y = 7, in which of the following cases does short-circuit evaluation occur?

```
A. if ( x<4 && y>8 ) {/* do something */}
B. while ( x<3 || y==8 ) {/* do something */}
C. if ( ((x=y-x)==4) || y>8 ) {/* do something */}
D. while ( x<=3 && ((x=y-x)==3) ) {/* do something */}
```

9. How many times are the power() function invoked during the execution of the following program?

```
#include <stdio.h>
// assume exp is always non-negative
int power(int base, int exp) {
    int ret = 1;    // base case
    if (exp > 0) ret = base*power(base, exp - 1);
    return ret;
}
int main() {
    int res = power(3,3);
}
```

A. 1 time
B. 2 times
C. 3 times
D. 4 times

10. Which of the following statements about the #define preprocessing directive is not correct?
A. #define follows the standard scoping rule of C.
B. #define can be used to define a macro.
C. #define can be used to define a parameterized macro.
D. #define can be used to create an include guard for the header file.

11. Given uint8_t x = 14 and uint8_t y = 5, which of the following statements is not true ?

```
A. (x & y) == 4;
B. (x | y) == 15;
C. (x ^ y) == 10;
D. (x >> 1) == 7;
```

12. Given enum {a=1,b,c=2,d=3,e}; what's the value of b and e?
A. b = 1, e = 3;
B. b = 0, e = 1;
C. b = 2, e = 3;
D. b = 2, e = 4;

13. which of the following statements about streams is incorrect?
A. stdin, stdout, and stderr are pre-opened streams if you include the stdio.h header file
B. printf() is connected to stdout
C. scanf() is connected to stdin
D. fscanf() is connected to stdout

14. Which of the following code snippets has a potential risk of memory leak?
A.

```c
#include <stdlib.h>
int main() {
    // void *malloc(size_t size);
    int *arr = malloc(sizeof(*arr)*4);
    if (arr != NULL) {
        /* do something*/
        free(arr);
    }
}
```

B.

```c
#include <stdlib.h>
int main() {
    // void *calloc(size_t nmemb, size_t size);
    int *arr = calloc(4,sizeof(*arr));
    if (arr != NULL) {
        /* do something*/
        free(arr);
    }
}
```

C.

```c
#include <stdlib.h>
int main() {
    // void *malloc(size_t size);
    int *arr = malloc(sizeof(*arr)*4);
    if (arr != NULL) {
        /* do something*/
        /* resize the allocated memory to include 6 int elements */
        // void *realloc(void *ptr, size_t size);
        arr = realloc(arr, sizeof(*arr)*6);
        /* do something*/
        free(arr);
    }
}
```

D.

```c
#include <stdlib.h>
int main() {
    // void *malloc(size_t size);
    int *arr = malloc(sizeof(*arr)*4);
    if (arr != NULL) {
        /* do something*/
        /* resize the allocated memory to include 6 int elements */
        // void *realloc(void *ptr, size_t size);
        int *rs_arr = realloc(arr, sizeof(*arr)*6);
        if (rs_arr != NULL) { /* do something*/ free(rs_arr);}
        else { /* do something*/ free(arr);}
    }
}
```

15. Given a function pointer void * (*fn_ptr)(double, int); which of the following statements about this function pointer is correct?
A. this function pointer is supposed to point to a function that doesn't return anything.
B. this function pointer is supposed to point to a function that takes two integer values as inputs.
C. this function pointer's identifier is fn_ptr.
D. you have to use the dereference operator on the function pointer to call the associated function.

16. which of the following statements about struct is incorrect?
A. the tag name of a struct is optional.
B. the size of a struct object equals the sum of the sizes of all its members.
C. a struct can't have a member of its own type, e.g., struct x { int a; struct x b;}; is invalid.
D. simple assignment between objects of the same struct type is allowed as long as the struct type doesn't have const-qualified members or a flexible array member.

17. which of the following statements about union is incorrect?
A. the tag name of a union is optional.
B. the size of a union object equals the size of its largest member.
C. flexible array is not allowed to be a union member.
D. more than one member of the union can hold different values at the same time.

18. which of the following statements about the building process of a multi-file program is incorrect?
A. One can compile the .c source files one by one, then link the associated object files later.
B. One can compile the .c source files and link the associated object files to form an executable all at once.
C. One can use the make tool and its Makefile to build a multi-file program with a better track of changes made to the source files.
D. the make tool will always compile all source files when used to build a program.

19. which of the following statements about the storage-class specifiers is incorrect?
A. file scope identifiers, unless explicitly declared with static specifier, have external linkage by default.
B. object declared with static specifier may have an internal linkage or no linkage.
C. object declared with static storage duration inside a function scope will be initialized only once.
D. the storage duration of objects always ranks in the following order: allocated < automatic < static.

20. Which of the following statements about the const qualifier is incorrect?
A. any attempt to modify a const-qualified object results in undefined behavior.
B. "int const * x;" declares a const-qualified pointer to an integer.
C. "const int * x;" declares a pointer to a const-qualified integer.
D. "const int * const x;" declares a const-qualified pointer to a const-qualified integer.

coding (25 pts)

21. Below are the source files main.c and util.c, please write and header file util.h for util.c, and set include guard in util.h **[5 pts]**
[hint] you may use some of the following preprocessing directives to setup the include guard: #define, #ifndef, #ifdef, #endif, #pragma once

main.c:

```c
#include <stdio.h>
#include "util.h"

int main() {
    printf("hello world!\n");
    foo(3.14);
    bar(10);
    return 0;
}
```

util.c:

```c
#include <stdio.h>
#include "util.h"

void foo(double x) {
    printf("call foo(%lf)\n", x);
}

void bar(int x) {
    printf("call bar(%d)\n", x);
}
```

util.h:

22. implement a function to reverse the elements' order in the array **[10 pts]**

function header:

```
void reverse_array(int * arr, int size);
```

example output of the function:

```
int x[1] = {3};
reverse_array(x,1); // x after reverse: {3};

int y[2] = {5,2};
reverse_array(y,2); // y after reverse: {2,5};

int z[5] = {-1,3,1,7,9};
reverse_array(z,5); // z after reverse: {9,7,1,3,-1};
```

write your reverse_array() function here (the full function definition):

23. Implement the general iterator function specified below. It takes a C-style string and a mapping function (function pointer) as input. It applies the input mapping function to characters in the C-style string to map the associated alphabetical letters to lowercase or uppercase using the given to_lower() or to_upper() functions respectively.
[Note:] you may assume the string.h header file is included for you, and use size_t strlen(const char * str) to acquire the length of the input string (this length excludes the terminating character at the end). And you need to come up with the proper declaration of a function pointer to the to_lower() and to_upper() functions. **[10 pts]**

The to_lower() function takes a char as input, and if this char is an uppercase letter (A-Z), return the corresponding lowercase letter. Otherwise, return the original input character.

```
char to_lower(char c) {
    return (c>=65 && c<=90) ? c+= 32 : c;
}
```

The to_upper() function is similar to the to_lower() function but converts lowercase input character to uppercase character instead.

```
char to_upper(char c) {
    return (c>=97 && c<=122) ? c -= 32 : c;
}
```

The general iterator() function use cases:

```
void iterator(char *str, /* declare a function pointer here */);
// output examples
char str[] = "Hello, World!"; // strlen(str) returns 13
iterator(str, to_upper);
printf("%s\n", str); // stdout: HELLO, WORLD!
iterator(str, to_lower);
printf("%s\n", str); // stdout: hello, world!
```

write your iterator() function here (the full function definition):