CS 120 Practice Exam

1. Rewrite the following expressions using the subscript `[]` operator:

```
int a[10] = { /* various comma-separated initializers */ }
int *p = a + 4;
```

a) `p + 5` _____

b) `*(p+2) + 4` _____

c) `(*p + 1) - 2` _____

d) `&(p + 2)` _____

2. Write the prototype of a function called `foo` that takes two parameters: a read-only array of `unsigned ints` and the number of elements in the array while returning a pointer to an `unsigned int`.

3. Rewrite the following while statement without using `if`, `if-else`, `for`, `while`, or `do` statements.

```
while (a < b) ++a;
```

Answer:

4. Replace the following `while` statement with an equivalent `for` statement:

```
int i = 0;
while (++i <= 5)
   printf("%d", i);
```

Answer:

5. What is the output produced by the following fragment of code? Answer: _____

```
int i = 5;
switch(i % 4) {
   case 0: printf("zero");
   case 1: printf("one");
   case 2: printf("two");
   case 3: printf("three");
}
```

6. What is the output produced by the following fragment of code? Answer: _____

```
int i, sum = 0;
for (i = 0; i < 10; ++i) {
   if (i % 2) continue;
   sum += i*3;
}
printf("%d", sum);
```

7. What is the output produced by the following fragment of code? Answer: _____

```
int j = 0;
if (j++ == ++j) printf("IF");
else printf("ELSE");
```

8. What is the output produced by the following fragment of code? Answer: _____

```
int j = 0;
if (j++ && ++j) printf("IF");
else printf("ELSE");
```

9. What is the output produced by the following fragment of code? Answer: _____

```
int a = 3, b = 8, c = 4, d = 1, e = 5;
printf("%d", a > b && b < 10 || ++a == 4);
```

10. What is the output produced by the following fragment of code? Answer: _____

```
int a = 3, b = 8, c = 4, d = 1, e = 5;
printf("%d", a > b && b < 10 && ++a == 4);
```

11. What is the output produced by the following fragment of code? Answer: _____

```
int a = 3, b = 8, c = 4, d = 1, e = 5;
printf("%d", a < b || b < c && c < d || d < e);
```

12. Write the C declaration for each of the English statements below:

a) `foo` is a pointer to a function that takes an `int` and returns an `int`:

b) `foo` is a function that takes a pointer to an array of `5` `int`s and returns a pointer to an `int`:

13. Does the following code compile? If so, what is the output? If not, explain the problem.

```
int a[] = { 6, 7, 8, 1, 2, 3, 9, 4, 5, 0 };
while (*a != 0) {
   printf("%d", (*a)*(*a));
   ++a;
}
```

14. Define a function `my_isalpha` to return `1` if the parameter is a Latin character, otherwise `0`. A Latin character belongs to either the set `[a - z]` or `[A - Z]`.

15. Define function `my_isupper` to return `1` if the parameter is an upper-case Latin character, otherwise `0`.

16. Define function `my_tolower` that converts a parameter `c` to its lowercase equivalent if `c` is an uppercase character and has a lowercase equivalent. If no such conversion is possible, the value returned is `c` unchanged.

17. Complete the following functions:

a)

```
/* add appropriate header file(s) ...


int* allocate(size_t n) { /* allocate memory here */


}
```

b)

```
void foo(void) {
   int *parr = allocate(15);

   /* ... doing something with the memory ... */

   /* release memory here */


}
```

18. Explain two problems that can arise when a program uses dynamically allocated memory.

19. Write a function that expands a null-terminated string by replacing all the tab characters with spaces. In other words, if parameter `tab_size` is 4, a single tab character (`'\t'`) should be expanded to 4 spaces. Note: You can assume that the output buffer has enough space to accommodate the expanded string.

```
void ExpandString(char *dst, const char *src, size_t tab_size) {









}
```

20. As shown below, in C (and in C++), the `typedef` storage specifier allows the declaration of a new name (or alias) for a built-in or user-defined type. This can also be achieved by using the `define` pre-processor directive. Explain which of these alternatives is a better choice.

For example: `typedef unsigned char BYTE; /* vs. */ #define unsigned char BYTE`

21. Write a function that counts the amount of whitespace (space, tab, and newline) in a text file:

```
void CountWhitespace(const char *file, int *space, int *tab, int *newline) {









}
```

22. Identify whether each of the following function declaration is legal or illegal. If illegal, provide a short explanation about the illegal nature of the declaration.

```
typedef struct TIME {
   int msec, mmin, mhrs;
} Time;
```

a) `TIME foo1(int, int, int);` _____

_____

b) `int foo2(Time*);` _____

_____

c) `bool foo3(struct TIME, struct TIME);` _____

_____

d) `void foo4(int, struct TIME*);` _____

_____

23. Given the following code fragment, write the value printed: _____

```
struct STRUCT_SAMPLE {
   int ma, mb;
   char mc;
   float md;
} s;

printf("%i", sizeof(s));
```

24. Given the following code fragment, write the value printed: _____

```
union UNION_SAMPLE {
   int ma;
   char mb[2];
   float mc;
} u;

printf("%i", sizeof(u));
```

25. Given the legal code below, write the output. Answer: _____

```
typedef union {
   struct { float mx, my; };
   float mv[2];
} Vector2D;

Vector2D pos = { 10.f, 15.f };
printf("%f %f", pos.mv[0], pos.mv[1]);
```

26. Given the following definitions:

```
int c[] = { 5, 7, 10, 3, 1 }, *pc = c;
/* assume pc contains rvalue 1000 and of sizeof(int) evaluates to value 4 */
```

determine whether each of the following expressions are legal or illegal. For legal expressions, compute the value resulting from the evaluation of the expression. Compare your answer with the table on the left hand side and choose the corresponding letter as your final answer. For example, you would write `A` as the answer if the expression evaluates to a value of `4`. **Assume non-cumulative execution, i.e., all modifications from previous lines ARE LOST.**

A) 4 B) 5 C) 8 D) 1000 E) 1001 F) 1004 G) illegal H) 10 I) 1012 J) 1002 K) 3 L) 6 M) 7 N) 1003	1. _____ `pc` 2. _____ `*pc` 3. _____ `*(pc+2)` 4. _____ `*pc++` 5. _____ `(*pc)++` 6. _____ `*++pc` 7. _____ `++*pc++` 8. _____ `(*++pc)++` 9. _____ `*(pc+*(pc+4))` 10. _____ `pc+c[3]` 11. _____ `*(pc+c[3])` 12. _____ `*pc+c[3]`

27. What is the output of the following code fragment:

```
char *strs[] = { "how", "stunt", "ball", "dalq", 0 }, **pps = strs;
int i = 0;
while (*pps) {
   printf("%c", *(*pps++ + i) + i);
   ++i;
}
```