Final Examination of Introduction to Programming (CS1355-01)

January 9, 2011

1. (2%) What are the so-called automatic storage duration and static storage duration, respectively?

2. (2%) Find the reason why `print_all_rows` in the following program will print only one row of 10 asterisks, instead of 10 rows.

```
int i;
void print_one_row(void) {
  for (i = 1; i <= 10; i++)
    printf("*");
}
void print_all_rows(void) {
  for (i = 1; i <= 10; i++) {
    print_one_row();
    printf("\n");
  }
}
```

3. (2%) The following program outline shows only function definition and variable declarations.

```
int a;
void f(void) {
  int b
}
void g(int c) {
  int d;
  {
    int e;
  }
}
```

   For each of the following scopes, list all variable and parameter names visible in that scope. (a) The `f` function. (b) The `block` in which `e` is declared.

4. (2%) The following function supposedly computes the sum and average of the numbers in the array `a`, which has length `n`. `avg` and `sum` point to variables that the function should modify. Unfortunately, the function contains several errors. Please find and correct them.

```
void avg_sum(double a[], int n, double *avg, double *sum) {
  int i;
  sum = 0.0;
  for (i = 0; i < n; i++)
    sum += a[i];
  avg = sum / n;
}
```

5. (2%) If `i` is a variable and `p` points to `i`, which of the following expressions are aliases for `i`? (a) `&p` (b) `&*p` (c) `&i` (d) `&*i` (e) `*p` (f) `*&p` (g) `*i` (h) `*&i`

6. (2%) If `i` is an `int` variable and `p` and `q` are pointers to `int`, which of the following assignments are legal? (a) `p = q;` (b) `p = &q;` (c) `p = *q;` (d) `p = &i;` (e) `p = *&q;` (f) `*p = *q;`

7. (2%) Suppose that following declarations are in effect:

```
int a[] = {34, 54, 14, 2, 52, 72, 5, 15};
int *p = &a[1], *q = &a[5];
```

   (a) What is the value of `*(p+3)`? (b) Is the condition `*p < *q` true or false?

8. (2%) What will be the contents of the `a` array after the following statements are executed?

```
#define N 10
int a[N] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
int *p = &a[0], *q = &a[N-1], temp;
while (p < q) {
  temp = *p;
  *p++ = *q;
  *q-- = temp;
}
```

9. (2%) The following function supposedly creates an identical copy of a string. What's wrong with the function?

```
char *duplicate(const char *p) {
  char *q;
  strcpy(q, p);
  return q;
}
```

10. (2%) Let `f` be the following function:

```
int f(char *s, char *t) {
  char *p1, *p2;
  for (p1 = s; *p1; p1++) {
    for (p2 = t; *p2; p2++)
      if (*p1 == *p2) break;
    if (*p2 == '\0') break;
  }
  return p1 - s;
}
```

   What is the value of `f("abccbad", "babc")`?

11. (2%) The following function calls supposedly write a single new-line character, but some are incorrect. Which are incorrect? (a) `printf("%c", "\n");` (b) `printf("%s", '\n');` (c) `putchar("\n");` (d) `puts("\n");`

12. (2%) What are the differences between the following two declarations of `date`?

```
char date[] = "January 9";
char *date = "January 9";
```

13. (2%) Which ones of the following statements are true?

    (a) Directives always begin with the `#` symbol.
    (b) To continue a directive to the next line, we must end the current line with a `/` character.
    (c) Directives can appear anywhere in a program.
    (d) Comments may appear on the same line as a directive.

14. (2%) Find the error in the following statements and fix it.

```
#define N = 100
int a[N];
```

15. (2%) Let the `PRINT_INT` macro is defined as follows:

```
#define PRINT_INT(n) printf(#n " = %d\n", n)
```

    What is the result of the invocation `PRINT_INT(i/j)` after preprocessing?

16. (2%) Find the error in the following statements and fix it.

```
#define ECHO(s) { gets(s); puts(s); }
if (echo_flag)
  ECHO(str);
else
  gets(str);
```

17. (4%) Let `DOUBLE` be the following macro:

```
#define DOUBLE(x) 2*x
```

    (a) What is the value of `DOUBLE(2 + 3)`? (2%) (b) Fix the definition of `DOUBLE`. (2%)

18. (2%) Suppose that the macro `M` has been defined as follows:

```
#define M 10
```

    Which of the following tests will fail? (a) `#if M` (b) `#ifndef M` (c) `#ifdef M` (d) `#if defined(M)`

19. (2%) Describe two properties of structure that are different from those of an array.

20. (2%) Find the error in the following statements and fix it.

```
struct { int number; } part1;
struct { int number; } part2;
part1 = part2;
```

21. (2%) What are the values of `BLACK` and `DK_GRAY` in the following statement?

```
enum EGA_colors {BLACK, LT_GRAY = 6, DK_GRAY, WHITE = 15};
```

22. (2%) What are the so-called memory leak (2%) and dangling pointers (2%), respectively?

23. (2%) Suppose that `s` is the following structure:

```
struct {
  double a;
  union {
    char b[4];
    double c;
    int d;
  } e;
  int f;
} s;
```

If `char` values occupy one byte, `int` values occupy four bytes, and `double` values occupy eight byte, how much space will a C compiler allocate for `s`? Assume that the compiler leaves no holes (i.e., no extra space) between members.

24. (2%) Find the error in the following statements.

```
char *p = malloc(4);
free(p);
strcpy(p, "abc");
```

25. (2%) Write the following function:

```
int *create_array(int n, int initial_value);
```

The function should return a pointer to a dynamically allocated `int` array with `n` members, each of which is initialized to `initial_value`. The return values should be `NULL` if the array can't be allocated.

26. (50%) Determine whether the following statements are correct or not (i.e., true or false). If not, please explain your reason (no reason, no point).

    (a) (2%) Both local and external variables can be used to transmit information to a function.

    (b) (2%) Parameters have automatic storage duration and block scope.

    (c) (2%) The following ordering is a legal way to organize a C program.

        ❶ `#include` directives
        ❷ `#define` directives
        ❸ Type definitions
        ❹ Declarations of external variables
        ❺ Prototypes for functions other than `main`
        ❻ Definition of `main`
        ❼ Definitions of other functions

    (d) (2%) The following statements are legal:

```
int *p;
*p = 1;
```

    (e) (2%) A function can return a pointer to an automatic local variable.

    (f) (2%) Suppose that `a` is the name of an array. Then both `a + i` and `&a[i]` represent a pointer to element `i` of `a`.

(g) (2%) Suppose that `a`, `b` and `c` are all pointer variables of the same type, and that `a` and `b` point to elements of an array. Then the following statement is legal.

```
c = (a + b) / 2;
```

(h) (2%) The statement "`*p++ = j;`" is equal to "`*p = j; p = p + 1;`".

(i) (2%) When passed to a function, an array name is treated as a pointer.

(j) (2%) "`char str[10]="abc";`" is equal to "`char str[10]; str = "abc";`".

(k) (2%) The following statements are legal:

```
char *p = "abc";
*p = 'd';
```

(l) (2%) An array of `n` characters can hold strings with length between 0 and `n`.

(m) (2%) If the user enters the command line "`mycc -o remind.exe remind.c`", then `argc` will be 3 and `argv[3]` will be `remind.c`.

(n) (2%) The input of the preprocessor is a C program and its output is an object code.

(o) (2%) A macro defined inside the body of a function is local to that function, so that this macro becomes undefined outside the function.

(p) (2%) The `#if` directive treats undefined identifiers as macros that have the value `1`.

(q) (2%) The following statements are legal:

```
struct { int a; } b;
b->a = 8;
```

(r) (2%) The following statements are legal:

```
struct part { int number; };
part part1, part2;
```

(s) (2%) The following statement is to declare a node structure that contain an integer and a pointer to the next node.

```
typedef struct {
  int value;
  Node *next;
} Node;
```

(t) (2%) Enumeration constants have exactly the same properties as constant created using `#define`.

(u) (2%) If a memory allocation function can't locate a memory block of the requested size, it returns a null pointer.

(v) (2%) The following loop is correct to delete all nodes from a linked list and release the memory that they occupy.

```
for (p = first; p != NULL; p = p->next)
  free(p);
```

(w) (2%) The following statements will locate memory for a string of `n` characters.

```
char *p;
p = malloc(n);
```

(x) (2%) The argument to `free` can be a pointer to any object, such as a variable or array element.

(y) (2%) If `x` is a structure and `a` is a member of that structure, then `(&x)->a` is not the same as `x.a`.