Exam 3 Answers

1. What does the following program print?

```
#include <stdio.h>

int main()
{
   char s[] = "Hsjodi";
   int i = 0;
   for (i = 0; s[i] != '\0'; ++i) {
      --s[i];
   }
   print("%s\n",s);
   return 0;
}
```

Answer: `Grinch`

Explanation: char is an integer type, so `--s[i]` is the character whose ascii code is one less than that of `s[i]`. Moreover, the ascii codes of lower-case characters and of upper case characters are consecutive. Thus `--'H'` is the character `'G'`, `--'s'` is the character `'r'`, etc.

What does the following program print?

```
#include <stdio.h>

int main()
{
   char s[] = "Grinch";
   int i = 0;
   for (i = 0; s[i] != '\0'; ++i) {
      ++s[i];
   }
   print("%s\n",s);
   return 0;
}
```

Answer: `Hsjodi`

Explanation: See above

What does the following code print?

```
char s[] = "Grinch";
printf("%d ",(int) sizeof(s));
printf("%d ",(int) strlen(s));
s[4] = '\0';
printf("%s\n",s);
```

Answer: `7 6 Grin`

Explanation: The array s requires 6 positions for the characters in the string plus one for the null character, so its size must be 7. The length of the string is the number of characters up to, but not including the null character, so it is 6. When we replace `s[4]` with the null character, the string in the array consists of the characters in the array up to, but not including the first null character in the array. Since we have replaced the character `s[4]`, w, which was `'c'`, with the null character, `s` now contains the string "`Grin`".

Note: Question 2 was graded as an extra credit problem with a possible score of 10.

2. (i) if i is a variable and p points to i, which of the following expressions are aliases for i?

(a) *p (b) &p (c) *&p (d) &*p (e) *i (f) &i (g) *&i (h) &*i

Answer: (a) and (g)

Explanation: That (a) is an alias of i is obvious from the definition of the dereferencing operator *. As for *&i, we are dereferencing the address of i, which is like dereferencing p, since p contains the address of i. But &p is the address of p; *&p is an alias for p; &*p is the address of i; *i is illegal, since i is not a pointer; &i is the address of i; and &*i is illegal, since i is not a pointer.

(ii) if I is an int variable and p and q are pointers to int, which of the following assignments are legal?

(a)	p = i;	illegal; p is a pointer and i is not
(b)	*p = &i;	illegal; *p is an int and &i is an address
(c)	&p = q;	illegal; &p is a constant
(d)	p = &q;	illegal; p is a pointer to int and &q is the address of a pointer
(e)	p = *&q;	legal; *&q is an alias for q, so this is equivalent to p = q;
(f)	p = q;	legal
(g)	p = *q;	illegal; p is a pointer and *q is an int
(h)	*p = q;	illegal; *p is an int, q is a pointer
(i)	*p = *q;	legal; *p and *q are both ints

(a) Fill in the blanks.

```
int d = 7;
int*  p = &d;
int** q = &p;

  *p is an alias for   __d__

  *q is an alias for   __p__

  **q is an alias for  __d__
```

Explanation: view the declaration of `q` as `(int*)* q = &p;` thus `q` is a pointer to something of type `int*`, which is consistent with assigning the address of `p` to `q`. Moreover, it then follows that *q is a pointer to int which is an alias for p. Thus, `**q = *(*q) = *p = d`.

(b) Assuming the statements from part (a), what is printed by the following code segment?

```
*p  += 3;    // equivalent to d += 3; d now has value 10
**q -= 4;    // equivalent to d -= 4; d now has value 6
printf("%d %d %d\n",*p, **q, d);
// equivalent to printf(("%d %d %d\n",d, d, d);
```

Answer: **6 6 6**

3. (a) `Max` is a function that accepts two `int` parameters and returns the value of the larger one. Two `int` variables, `population1` and `population2`, have already been declared and initialized. Write an expression (not a statement) whose value is the larger of `population1` and `population2`.

```
Max(population1,population2)
```

(b) Write a statement that declares a prototype for a function `powerTo` which has two parameters. The first is a `double` and the second an `int`. The function returns a `double`.

```
double powerTo(double,int);
```

Explanation: a functions prototype consists of the header for the function terminated by a semi-colon. The parameter names are optional in a prototype.

(c) Write the include directive needed to allow use of functions like `printf` and `fgetc`.

```
#include <stdio.h>
```

(d) Write the definition of a function `printLarger`, which has two `int` parameters and returns nothing. The function prints the larger value of the two parameters to standard output on a single line by itself.

```
void printLarger(int x, int y)
{
   if (x > y) {
     printf("%d\n",x);
   } else {
     printf("%d\n",y);
   }
}
```

(a) Write a statement that declares a prototype for a function `printErrorDescription` which has one `int` parameter and returns nothing.

```
void printErrorDescription(int);
```

(b) Write a statement that declares a prototype for a function `add` which has two `int` parameter sand returns an `int`.

```
int add(int,int);
```

(c) Write the include directive needed to allow use of the functions `islower()` and `isupper()`.

```
#include <ctype.h>
```

(d) Write the definition of a function `printGrade` which has a `char` parameter and returns nothing. The function prints on a line by itself the message "`Grade:` " followed by the value of the `char` parameter.

```
void printGrade(char ch)
{
   printf("Grade: %c\n",ch);
}
```

4. (a) Write a single statement containing the declaration with initializer for a two-dimensional array that represents the following matrix:

$$\begin{pmatrix} 2 & 1 & 0 \\ 1 & 3 & 2 \end{pmatrix}$$

```
int A[2][3] = { {2,1,0}, {1,3,2} };
```

(b) You are to write the definition for the functionbelow that computes the sum of each row of a two-dimensional array `A`, storing the results in the one-dimensional array `rowsums`. Thus, the entry of `rowsums` at index `i` should be the sum of the entries in row `i` of `A`.

`A` has `COLMAX` columns, where `COLMAX` is a previously defined constant; the number of rows of `A` is passed in via the parameter `rowmax`.

```
void RowSums(int A[][COLMAX], int rowmax, int rowsums[])
{
   int i = 0, j = 0, sum = 0;
   for (i = 0; i < rowmax; ++i {
     sum = 0;
     for(j = 0; j < COLMAX; ++j) {
        sum += A[i][j];
     }
     rowsums[i] = sum;
   }
}
```

(b) You are to write the definition for the functionbelow that computes the sum of each column of a two-dimensional array `A`, storing the results in the one-dimensional array c`olsums`. Thus, the entry of `colsums` at index `j` should be the sum of the entries in column `j` of `A`.

`A` has `COLMAX` columns, where `COLMAX` is a previously defined constant; the number of rows of `A` is passed in via the parameter `rowmax`.

```
void ColSums(int A[][COLMAX], int rowmax, int colsums[])
{
   int i = 0, j = 0, sum = 0;
   for (j = 0; j < COLMAX; ++j {
     sum = 0;
     for(i = 0; j < rowmax; ++i) {
        sum += A[i][j];
     }
     rowsums[i] = sum;
   }
}
```

5. Consider the following program.

```
#include <stdio.h>
void add_doubles(double  x,double  y, double  sum)
{
    sum = x + y;
}

int main()
{
   double u = 2.0, v = 3.2, w = 8.2;
   add(  u,  v,  w);
   printf("the sum is %.1f\n",w);
   return 0;
}
```

If necessary, modify the above program so that the value printed is 5.2. If no modification is needed, write "No change needed".

Note: question should have stated that you may not change the function's return type. Also, make as few changes as possible.

Answer:

```
include <stdio.h>
void add_doubles(double  x,double  y, double  *sum)
{
    *sum = x + y;
}

int main()
{
   double u = 2.0, v = 3.2, w = 8.2;
   add(  u,  v,  &w);
   printf("the sum is %.1f\n",w);
   return 0;
}
```

Write a complete program according to the following specifications.

- The program declares and defines a void function that computes the area and perimenter length of a rectangle given its length and width (double values). The function should have four parameters: two that are used to pass in the length and width, and two used to return the area and perimeter length. The function should contain no printf statements.

- The main function of the program inputs the length and width and calls the function to compute the area and perimeter length. Do not do input validation.

- The main function then prints a statement reporting the area and perimeter length.

```
void Area_and_Perimeter(double len, double wid,
                        double *area, double *perimeter);

int main()
{
   int length = 0.0, width = 0.0,
       area = 0.0, boundary = 0.0;
   scanf("%lf %lf",&length,&width);
   Area_and_Perimeter(length,width,&area,&boundary);
   printf("Area: %.2f; Perimeter: %.2f\n",
           area, boundary);
   return 0;
}

void Area_and_Perimeter(double len, double wid,
                        double *area, double *perimeter)
{
   *area = len * wid;
   *perimeter = 2*len + 2*wid;
}
```