

NAME Solutions S.S.# \_\_\_\_\_

- #1. (Functions) Write a function that receives 3 integers, and returns the value of the largest one through the function name.

```
int largest( int a, int b, int c)
{
    int max;

    max = a;
    if (b > max)
        max = b;
    if (c > max)
        max = c;
    return max;
}
```

- #2. (Functions) What is the output of this segment of code.

```
int z = 0;
```

Ans.

```
void bill (int c)
{
    int x = 0;
    static int y = 0;
    x = x + c;
    y = y + 2 * c;
    z = z + 3 * c;
    printf (" x = %d, y = %d, z = %d \n",x,y,z);
}
```

X = 5, Y = 10, Z = 15

X = 7, Y = 24, Z = 36

X = 36, Y = 96, Z = 144

```
main()
{
    int x = 5;

    bill (x);    <-- x = 5
    x = x + 2;
    bill (x);    <-- x = 7
    bill (z);    <-- z = 36
}
```

- #3. (Recursion) Write a recursive function to compute  $f(x) = f(x - 1) + f(x - 2)$  with  $f(0) = 0$  and  $f(1) = 1$ .

```
int f( int x)
{
    if (x < 2)
        return x;
    else
        return f(x - 1) + f(x - 2);
}
```

- #4. (Arrays) Write a loop to double the value of each component of an array of integers called A. Stop at the component containing 0. Ex. before:  $A = \{2,5,7,0,3,5\}$  after  $A = \{4,10,14,0,3,5\}$ . Declare the array A and any variable needed.

```
i = 0;
while (A[i] != 0)
{
    a[i] = A[i] * 2;
    i++;
}
```

- #5. (Structures and functions) Declare a structure called point that stores 2 integers, x and y. Declare another structure called segment that stores 2 points, p1 and p2. Finally write a function that given a segment and a point determines if the point is on the segment. The function returns true if the point is on the segment and false otherwise. A point is on a segment if the distance from one end of the segment, p1, to the other end, p2, is the same as the distance from p1 to the point plus the distance from the point to p2.

```
struct point
{
    int    x;
    int    y;
}

struct segment
{
    point    p1;
    point    p2;
}

double dist(    point    p1, // returns the distance
               point    p2) // between two points
{
    return sqrt(    (p2.x - p1.x)*(p2.x - p1.x) +
                   (p2.y - p1.y)*(p2.y - p1.y) )
}

int point_on_segment(    segment    s,
                       point        p)
{
    return dist(s.p1,s.p2) == dist(s.p1,p) + dist(p,s.p2);
}
```

#6. (Pointers) What is the output of the following code segment. Assume the address of x is 4000 and y is 4002.

```
void bill(int *x,*y)           Ans.
{
  *x = *x + 10;                10  25
  *y = *y + 20;                30  35
  y = 8000;    <-- has no effect  60  35
}
```

```
void main()
{
  int x,y,*p;

  x = 0;
  y = 5;
  bill(&x,&y);
  printf(“%d %d\n”,x,y);

  bill(&y,&x);    <-- swapped x and y
  printf(“%d %d \n”,x,y);

  p = 4000;
  bill(p,p);    since &x = 4000,
                <-- same as bill (&x, &x)
  printf(“%d %d\n”,x,y);
}
```

#7. ( Scope ). On the right side show what is the output of this program.

```
#include "stdio.h"

int b[5] = {1,2,1,2,5},
    c[5] = {2,1,2,1,0};

void bill(int x, int *p)
{
    int I;

    x--;
    for (I = 0; I < x; I++)
        p[I] = p[I] + b[I] + c[I];
}

void main()
{
    int a[5] = {1,2,3,4,5},
        c[5] = {5,4,3,2,1};

    int I;

    bill(a[4],a);
    for (I = 0; I < 5; I++)
        printf(" %d ",a[I]);
    printf("\n");

    bill(c[0],c);
    for (I = 0; I < 5; I++)
        printf(" %d ",c[I]);
    printf("\n");

    bill(b[4],b);
    for (I = 0; I < 5; I++)
        printf(" %d ",b[I]);
    printf("\n");
}
```

output:

```
4 5 6 7 5
8 7 6 5 1
4 5 4 5 5

a = 1 2 3 4 5
b = 1 2 1 2
c = 2 1 2 1
    4 5 6 7 5

5 4 3 2 1
1 2 1 2
2 1 2 1
8 7 6 5 1

1 2 1 2 5
1 2 1 2
2 1 2 1
4 5 4 5 5
```

#8. (Scope) What is the output if you enter 1,2 and 3 for k:

```
void main()
{
  int    *p,x;           (k == 1)         x = 3  y = 6      
  char   c;             (k == 2)         x = 7  y = 9      

  p = &x;
  x = 3;                (k == 3)         x = 10 y = 11      

  y = 4;

  scanf("%d", &k);

  if (k == 1)
    {
      int x;

      x = 5;           <-- modifies only the local x not main's x
      y = 6;
    }
  else if (k == 2)
    {
      int x;

      *p = 7;         <-- p points to main's x
      x = 8;         <-- modifies the local x
      y = 9;
    }
  else
    {
      x = 10;
      y = 11;
    }

  printf ("x = %d, y = %d \n",x,y); <-- prints main's x and y
}
```

#9. (Functions and Scope) What is the output of the following code segment. Assume the address of x is 4000 and y is 4002.

<pre> void bill(int x,*y)     {         x = 56;         *y = 72;         y = 8000;     } </pre>	<pre> &lt;-- has no effect &lt;-- has no effect </pre>	<pre> Ans. 73 72 72 72 72 2 3 </pre>
<pre> void main()     {         int x,y,*p,a[3];          x = 37;         y = 42;         bill(x,&amp;y);         printf(“%d %d\n”,x,y);          p = 4000;         bill(x,p)         printf(“%d %d\n”,x,y);          a[0] = 1;a[1] = 2; a[2] = 3;         bill(a[0],a);         printf(“%d %d %d \n”,a[0],a[1],a[2]);     } </pre>	<pre> &lt;-- p points to x </pre>	<pre> &lt;-- a is the address of the first component </pre>

- #10. (Just a good exercise) Write a for loop to print the first 10 numbers in the sequence where the next number is the previous number plus the index that is being incremented by 1 each time it goes through the loop. ex 0 1 3 6 10 15 21 ... note:  $0+(0) = 0$ ,  $0+(1) = 1$ ,  $1+(2) = 3$ ,  $3+(3) = 6$ ,  $6+(4) = 10$ , ...

```
y = 0;
for (x = 0; x < 10; x++)
{
    y = y + x;
    printf(" %d ",y);
}
```

- #11. (Recursion) Write a recursive function to compute  $f(x) = f(x - 1) + x$  with  $f(0) = 0$ . This will generate the sequence in the problem above.

```
int f(int x)
{
    if ( x < 1)
        return 0;
    else
        return f(x - 1) + x;
}
```

- #12. (Files) Write a function to read the data from a file called data.dat into a 2 dimensional array of integers. The first line in the file contains 2 numbers, the row and column dimensions respectively. The remainder of the lines in the file contain the data for the array organized by its dimensions ( each row is on a new line).

Example data.dat file: **Do not assume the file is as below.**

```
2 3
1 5 2
4 6 1
```

You must dynamically allocate memory for this array since you do not know ahead of time how large this array is to be.

Hint: open the file, read the first line (row, column), dynamically allocate sufficient memory using malloc or new, then finally use a double “for” loop to initialize the array from the file.

```
void    bill ( )
{
    FILE *fp;
    int   row,col,r,c;

    fp = fopen("data.dat","r");
    if (fp == NULL)
        {
            printf("Unable to open file\n");
            return;
        }

    fscanf(fp,"%d %d",&row,&col);
    table = new int[row][col];

    for (r = 0; r < row; r++)
        for (c = 0; c < col; c++)
            fscanf(fp,"%d ", &(table[r][c]) );

    fclose(fp);
}
```

The following is technically incorrect because the array indexes are not known at compile time.

Must use : `fscanf(fp,"%d ", table + r * row + c );`

#13. (Short answers). Answer each of the following.

a. What does extern do.

extern tells the compiler that a variable is declared elsewhere and describes how that variable is declared.

b. What does static do. Note it does 2 different things depending on how you use it.

static makes a local variable permanently reside in memory and also limits the scope of variables having file scope to the current file.

c. What must one do to be able to modify a variable in a function and have the change be reflected in the main program that calls the function. Give example.

Use call by reference.  
declare the function as:

```
bill( int      *x)
    {
    *x = 5;
    }
```

Use as:

```
int  y;
bill(&y);
```

- d. What is the difference between using `fprintf` and `fwrite`. Give a typical application for using each.

`fprintf` is a formatted output function. All of the data is converted to ASCII before saving to the file. `fwrite` on the other hand is not formatted. The data is stored as it is in memory. Since the data size is predictable using `fwrite`, one can calculate where a certain piece of data is stored in the file and read that data directly without having to read the rest of the data in the file. Random access. Since `fprintf` saves in ASCII the file is readable by other applications.

- e. What is the advantage of using pointers and dynamic memory allocation when implementing a stack or a queue? Why not just use an array?

An array may run out of memory or may have a lot of unused memory. Using dynamic memory allows the program to allocate the memory as needed.

#14. (Declarations) Declare the following variables. Do not initialize them.

- a. Declare a structure called record that holds a person's name, age, social security number phone number and salary.

```
struct    record
{
    char name[80];
    int  age;
    char ss[12];
    char phone[20];
    float salary;
}
```

- b. Declare a structure called node to be used in a linked list that has pointers to the previous and next nodes and the data part is a pointer to a void.

```
struct node
{
    void      *data;
    node      *next;
    node      *prev;
}
```

- c. Declare an array with 5 elements where each element is a pointer to a double called apd.

```
double      *apd[5];
```

- d. Declare an array of 5 elements where each element is an array of 3 elements of doubles called aad.

```
double      aad[3][5];
```