# MTH 307/417/515 Midterm Solutions

1. Write the output for the following programs. Explain the reasoning behind your answer.

   (a)
```
#include<stdio.h>
void main()
{
    int i;
    switch(i += 5/2*6 + 3.0)
    {
        default:printf("Ronaldo");
        case 3:printf("David Beckham");
               break;
        case 15:printf("Ronaldinho");
               break;
        case 0:printf("Lionel Messi");
               break;
    }
}
```

   **Answer.** This yields the output: `Ronaldinho`.

   Reasoning: As the variable `i` is not initialized, all modern compilers will initialize its value to `0` by default. Hence, `switch` expression upon evaluation yields the value `15`. (Caution: Some

primitive compilers might initialize `i` with some junk value. So this output may vary.)

(b)
```
#include<stdio.h>
int main()
{
    int i = -10.99, j = -1.0, k = 1, l = 0.0, m;
    m += (++i - ++j * k-- / -l++);
    printf("%d, %d, %d, %d, %d \n", i, j, k, l, m);
    return 0;
}
```

**Answer.** This yields the output: `-9, 0, 0, 1, -9`.

Reasoning: This is a direct application of the operator precedence rules detailed in Page 12 of the lesson plan.

(c)
```
#include <stdio.h>
int main()
{
    int number;
    printf("%d \n %d \t %d \n %d %d",calc(-1),calc(0),calc(4),calc(2),calc(3));
    return 0;
}
int calc(int num)
{
    if (num >= 0)
```

```
        return num + calc(num-1); /* This is called recursion */
    else
        return num;
}
```

**Answer.** This yields the output:

```
-1
 -1      9
 2 5
```

Reasoning: The important thing to note here is that the function `calc` is called within itself. The process of calling a function within itself is called *recursion*, and the `C` compiler allows recursive calls. Each call of the `calc` function in `main` passes a value for `num` into `calc`, which returns a value based on whether `num` satisfies the `if` condition, or not. If the `if`-condition holds true (for example, in the case of `2`), then the function `calc` is called within itself with the value `num-1`. This process continues until the condition is violated, so this works almost like a `while` loop. If the value passed from `main` violates the condition (for example, in the case of `-1`), the same value is returned.

(d)
```
#include <stdio.h>
int main()
{
```

```
    int a[5]={1,2,3,4,5}, b[5]={10,20,30,40,50}, tally;
    for(tally = 0; tally <  5; ++tally)
        *(a + tally) = *(tally + a) + *(b + tally);
    for(tally = 0; tally <  5; tally++)
        printf("%d ",*(a + tally));
    return 0;
}
```

**Answer.** This gives the output: `11 22 33 44 55`

Reasoning: This simply uses the fact that an array name, by default, points to the first element of the array (see Page 28 of the lesson plan).

2. Write `C` programs for the following.

   (a) Computing the value of $e^x$ up to $d$ decimal places of accuracy, where the values of $x$ (of `double` type) and $d$ (of `int` type) are accepted as input.

   **Answer.** Below is a program that executes this task.

```
#include<stdio.h>
int factorial(int n)
 {
     int i, fact = 1;
     for(i = 1; i <= n; i++)
```

```
     {
         fact *= i;
      }
    return fact;
}
double power(double x, int n)
{
    int i;
    double prod = 1.0;
    for(i = 1; i <= n; i++)
        prod *= x;
    return prod;
}
int main()
 {
     int d, i;
     double x, sum = 1.0;
     printf("\n Enter the exponent of e : ");
     scanf("%lf", &x);
     printf("\n Enter the decimal accuracy : ");
     scanf("%d", &d);
     for(i = 1; i <= (2*d+1); i++)
     {
         sum += (power(x,i)/factorial(i));
```

```
    }
    printf("\n The value of e^(%f) up to %d decimal accuracy  = %f", x, d, sum);
    return 0;
}
```

(b) Declare a structure named `vector` whose members are three double variables `x, y, and z`, which store the values of the $x$-coordinate, the $y$-coordinate, and the $z$-coordinate (resp.) of a 3-dimensional vector.

(i) Define a function named `dotproduct` that takes two arguments `vec1` and `vec2`, of `vector` type, and returns a double value that is the dot product of the vectors `vec1` and `vec2`.

(ii) Define a function named `crossproduct` that takes two arguments, `vec1` and `vec2`, of `vector` type, and returns a `vector` type variable that is cross product of the vectors `vec1` and `vec2`.

(iii) Use the functions `dotproduct` and `crossproduct` to determine whether two given vectors are parallel, orthogonal, or skew vectors.

**Answer.** Below is a program that executes these tasks.

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

```
{
        double x, y, z;
}vector;
double dotproduct(vector vec1, vector vec2)
{
    double prod;
    prod = (vec1.x * vec2.x) + (vec1.y * vec2.y) + (vec1.z * vec2.z);
    return prod;
}
vector crossproduct(vector vec1, vector vec2)
{
    vector vec3;
    double x1, y1, z1;
    x1 = (vec1.y * vec2.z) - (vec1.z * vec2.y);
    y1 = (vec1.x * vec2.z) - (vec1.z * vec2.x);
    z1 = (vec1.x * vec2.y) - (vec1.y * vec2.x);
    vec3.x = x1;
    vec3.y = y1;
    vec3.z = z1;
    return vec3;
}
void printvec(vector vec)
{
    printf("(%lf, %lf, %lf)", vec.x, vec.y, vec.z);
```

```
}
void main()
{
    vector vec1, vec2, cprod;
    double dprod;
    printf("\n Enter coordinates of first vector in the form (x, y, z) : ");
    scanf("(%lf, %lf, %lf)", &vec1.x, &vec1.y, &vec1.z);
    printf("\n Enter coordinates of second vector in the form (x, y, z) : ");
    scanf("(%lf, %lf, %lf)", &vec2.x, &vec2.y, &vec2.z);
    dprod = dotproduct(vec1, vec2);
    cprod = crossproduct(vec1, vec2);
    printf("\n The dot product of the two vectors is : %lf", dprod);
    printf("\n The cross product of the two vectors is : "); printvec(cprod);
    if (dprod == 0)
        printf("\n The vectors entered are orthogonal");
    else if ((cprod.x == 0) && (cprod.y == 0) && (cprod.z == 0))
        printf("\n The vectors entered are parallel");
    else printf("\n The vectors entered are skew");
}
```

(c) Define a function `int *find_largest(int a[])` that sorts a given integer array `a` in ascending order, and returns the pointer to the largest element in the array. The program should accept an inte-

ger array and use `find_largest` to sort the array, and then print the sorted array and the largest element in the array.

**Answer.** Below is a program that executes these tasks.

```
#include<stdio.h>
#define N 10
int *find_largest(int a[])
{
    int i, j, temp, *p;
    for(i = 0; i < N; i++)
    {
        for(j = i+1; j < N; j++)
        {
            if(a[i] > a[j])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    p = &a[N-1];
    return p;
}
```

```
int main()
{
    int i, a[N], *p;
    printf("\n Enter an array of size %d: \n",N);
    for(i = 0; i < N; i++)
    {
        scanf("%d",&a[i]);
    }
    p = find_largest(a);
    printf("\n The sorted array is: \n");
    for(i = 0; i < N; i++)
    {
        printf("%d ",a[i]);
    }
    printf("\n The largest element is: %d", *p);
    return 0;
}
```