120 MINUTES **METU CEng 230 FINAL EXAM** **17.01.2009**

There are 18 questions for a total of 100 points.
Questions 1-15 are multiple choice (5 points each); 16–18 are short answers (8, 8, 9 points respectively).
In multiple choice questions you will not be penalized for your wrong answers.
For Questions 16–17, a partial credit up to 2 points may be granted.
For Question 18, partial credit may be granted up to 6 points.
Please write the answer that you think is most appropriate. Assume all variables are properly declared.

Name: **Student ID**: **Grade**: + =

1- (5 points) Which of the following *best describes* the behavior of the function "*int myFun (int x[], int y[], int k)*"? Assume that the sizes of arrays x and y are both greater than k.

```
int helperFun(int a, int b){
   if (a > b){
      return a;
   } else {
      return b;
   }
}

int myFun (int x[], int y[], int k){
   int i;
   int total = 0;
   for (i=0; i<k; i++){
      total += helperFun(x[i], y[i]);
   }
   return total;
}
```

(a) It returns the sum of all elements in array x.
(b) It returns the sum of all elements in array y.
(c) It calculates a sum based on the largest of the ith element of arrays x and y, where $1 \leq i \leq k$.
(d) It returns the sum of all elements in array x and y, altogether.

2- (5 points) Assume that you want to ***calculate*** the average lab grade for each Ceng230 section. Furthermore, you would like to see how each section performed on a specific lab assignment (e.g. Switch). In this case, what would be the *most suitable* data structure for storing the grades? (There are a total of 8 sections and 10 lab assignments. Each section contains exactly 50 students. Grades are recorded as integers.)

(a) int [80][50];
(b) char [80][50];
(c) float [8][50];
(d) int [8][10][50];

3- (5 points) How many of the following statements are true of C's program structure?
(i) Global variables ***are not*** allowed.
(ii) Nested loops ***can be*** defined.
(iii) Functions have to take ***at least*** one input argument.
(iv) Only the ***main*** function can have ***int*** as its return type.

(a) 0
(b) 1
(c) 2
(d) 3

4- (5 points) Which of the following is an incorrect initialization of the C string "CENG"?

(a) char a[5] = {'C','E','N','G','\0'};
(b) char a[] = "CENG";
(c) char a[4] = "CENG";
(d) char a[5] = "CENG";

5- (5 points) What gets printed out by the following code fragment?

```
int myFun (int x){
   x = x * 2;
   printf("%d ", x);
}

int main(int argc, char *argv){
   int x = 3;
   printf("%d ", x);
   myFun(x);
   printf("%d ", x);
}
```

(a) 3 3 3
(b) 3 6 6
(c) 3 6 3
(d) 3 3

6- (5 points) Which of the following functions would correctly return the value passed to it as argument incremented by 1? (For example inc(3) should return 4, inc(4) should return 5, inc(5) should return 6, etc.)

(a) int inc (int x){return x++;}
(b) int inc (int x){return ++x;}
(c) void inc (int x) {x++;}
(d) int inc () {int x; return x; }

7- (5 points) What would be displayed by the following program segment?

```
int total = 0;
int a[5][5];
int i, j, k;

for (i=0; i<5; i++){
   for (j=0; j<5; j++){
      a[i][j] = i*5+j+1;
   }
}
for (k=0; k<5; k++){
   total += a[k][k];
}
printf("%d\n",total);
```

(a) 15
(b) 63
(c) 65
(d) 115

8- (5 points) Which of the calls to the function defined below returns a ***positive*** integer?

```
int isP (char string[], int size){
   int result = 1;
   int i;
   for (i=0; i<(size/2); i++){
      result = result &&
           (string[i] ==
                    string[(size-1-i)]);
   }
   return result;
}
```

(a) isP("john",4)
(b) isP("anna",4)
(c) isP("mary",4)
(d) isP("jack",4)

9- (5 points) Which of the following is an ***invalid*** declaration of an array as a formal parameter of a function?
(a) int x[1][2][3]
(b) int x[]
(c) int x[][12]
(d) int x[2][]

10- (5 points) How many of the following function calls returns a ***positive*** number? (*Hint: strcmp("q", "Q") returns 1.*)

(i) strcmp ("qwerty","Qwerty")
(ii) strcmp ("qwerty","qwerty")
(iii) strcmp ("qwerty","qwertY")
(iv) strncmp ("qwertY","qwerty",5)

(a) 0
(b) 1
(c) 2
(d) 3

11- (5 points) What would be the value returned by the following function call? (Assume <math.h> has already been included.)

myFun(1.5, 1.0, -1.5, 5.0);

```
double myFun (double a, double b, double
c, double d) {
     double t1 = pow(fabs(a - c),2);
     double t2 = pow(fabs(b - d),2);
     return pow( t1 + t2,0.5);
}
```

(a) 4.0
(b) 5.0
(c) 6.0
(d) 7.0

12- (5 points) Which of the following best describes what the declaration ***int a[3][2][5]***; produces?

(a) An array of 30 characters all initialized to 'a'.
(b) An array of 30 integer numbers.
(c) Such a declaration cannot be compiled.
(d) An uninitialized array of 30 chararcters.

13- (5 points) What is the value of ***a[0][1]*** for the declaration

```
int a[2][2] = {{1,2},{3,4}};
```

(a) 1
(b) 2
(c) 3
(d) 4

14- (5 points) What is the content of *array* after the execution of the following code segment?

```
int array[8] = {1, 2, 3, 4, 5, 6, 7, 8};

int i;
for (i=0; i<8; i++){
        int temp = array[i];
        array[i] = array[7-i];
        array[7-i] = temp;
}
```

(a) {1, 2, 3, 4, 5, 6, 7, 8}
(b) {8, 7, 6, 5, 4, 3, 2, 1}
(c) {2, 1, 4, 3, 6, 5, 8, 7}
(d) {1, 1, 1, 1, 1, 1, 1, 1}

15- (5 points) What is the value of the variable x after the following program fragment is executed?

```
double x;
x = 21 / 6;
```

(a) 3.0
(b) 4.0
(c) 3.5
(d) 4.5

16- (8 points) ***locationOfMinElement*** is a function that returns the location of the minimum element in an integer array. You may assume that the array contains at least one element and that it does not contain any duplicate values.

As an example, consider the following initialization:

```
int array[5] = {5, 4, 2, -5, 7};
```

In this case, a call to the function:

```
locationOfMinElement (array, 5);
```

returns ***3***, representing the location of the element with the minimum value.

Please note that locations are defined starting from 0; that is for the given initialization, element 5 would be on location 0, 4 on location 1, 2 on location 2, -5 on location 3 and 7 on location 4.

Complete the missing statements in the following code segments.

```
// returns the location of the element with 'value'
// 'array' has size='arraySize'
// location starts from 0
// array does not contain any duplicates
// if element cannot be found, returns -1
int location(int array[], int arraySize, int value){
   for (int i=0; i<arraySize; i++){
      if (  ANSWER:                              ){
         return i;
      }
   }
   return -1;
}

int locationOfMinElement (int array[], int arraySize){
   int minValue = array[0];
   int i;
   for (i=1; i<arraySize; i++){
      if ( array[i] < minValue ){
         minValue = array[i];
      }
   }

   return  ANSWER:

}
```

17- (8 points) What is the content of ***array*** after execution of the following code fragment?

```
int i;
char array[] = {'C','E','N','G','2','3','0','\0'};
int size = strlen(array);
for (i=0; i<(size/2); i++){
   array[i] = array[size-i-1];
}
```

ANSWER: { ___, ___, ___, ___, ___, ___, ___, ___ }

18- (9 points) A function is called **recursive** if it can call itself. A recursive function must have stopping condition(s) to avoid infinitely many calls to itself. Furthermore, **mutual recursion** is a form of **recursion** where two mathematical or computational functions are defined in terms of each other. For example the function *isEven* (which returns 1 if the number is even and 0 otherwise) can be defined in terms of another function *isOdd* (which returns 1 if the number is odd and 0 otherwise), as follows. Since these two functions recursively call each other, they are called mutually recursive.

```
int isOdd(int number){
      if (number==0) {
        // number is not odd
        return 0;
      } else {
          // number is odd if (number-1)
          // is even
          return isEven(--number);
      }
}
```

```
int isEven(int number){
      if (number==0) {
        // number is even
        return 1;
      } else {
          // number is even if (number-1)
          // is odd
          return isOdd(--number);
      }
}
```

isEven(3) $\rightarrow$ 0
isEven(4) $\rightarrow$ 1

However, ***isEven*** goes into an infinite loop when called with negative numbers. Without changing the stopping conditions, modify the following places in functions *isOdd* and *isEven* so that ***isEven*** now works for negative numbers, too.

```
int isOdd(int number){
      if (number==0) {
        // number is not odd
        return 0;
      } else {
```

ANSWER:

```
      }
}
```

```
int isEven(int number){
      if (number==0) {
        // number is even
        return 1;
      } else {
```

ANSWER:

```
      }
}
```