# CSE142 Exam with answers

Problem numbering may differ from the test as given.

All multiple choice questions are equally weighted. You can generally assume that code shown in the questions is intended to be syntactically correct, unless something in the question or one of the answers suggests otherwise.

CSE142 Final exam 6/6/01, with answers. MC questions only. For original formatting, see the unsolved version.

**1. What is output by the following program?**

```
#include <stdio.h>
#define N 10

int main( )
{
  int i, j;
  int A[N];

  for(i=0; i<N; i=i+1) {
    A[i] = 2;
    for(j=0; j<i; j=j+1) {
      A[i] = A[i] + 1;
    }
  }
  printf("%d\n", A[3]);
  return 0;
}
```

**A.** 2
**B.** 3
**C.** 5
**D.** 9
**E.** 11

ANSWER: **C**

**2. Which of the following is guaranteed to be true about EOF in C?**

**A. It is a special data value written at the end of each file.**

**B. It is a status indicating that the end of a file has been reached.**

**C. It is defined in the stdio.h header file.**

**A.** A only
**B.** B only
**C.** C only
**D.** A and B
**E.** B and C

ANSWER: **E**

**3. Suppose @ and # are operators (in some imaginary programming language that has precedence and associativity rules like C), and suppose you know that the expression**
**@ @ a # @ b # c**

**is evaluated as if it had parentheses as follows:**

**(@ (@ a)) # ((@ b) # c)**

**Study this information to see if you can tell whether the operators are unary or binary, and something about their associativity.**
**What must be true?**

**A.** @ is binary, left associative
**B.** @ is unary, left associative
**C.** # is unary
**D.** # is binary, left associative
**E.** # is binary, right associative
ANSWER: **E**
The grouping (@b) shows that @ is unary. This is consistent with the other uses of @. Likewise, # always has an operator on both its left and right, so it is binary.
From (@(@a)) we see that the rightmost @ is evaluated first, so @ is right associative.
# is also right associative, since ((@b)#c) is done first, before the other # is applied.

**4. What's the output of the following code fragment:**

```
#include <stdio.h>

void foo(int d)
{
  int a;

  a = d;
  d = 2 * a;
  printf("%d ", d);
}

void main(void)
{
  int a, d;

  d = 3;
  a = 2;
  foo(3);
  printf("%d %d", a, d);
}
```

**A.** 6 3 6
**B.** 6 3 3
**C.** 6 2 6
**D.** 3 2 6
**E.** 6 2 3
ANSWER: **E**

**5. A palindrome is a word which is spelled the same forwards as it is backwards. Eamples:**
**ABBA**
**nun**

**Below is a function is_palindrome() which determines if a string is a palindrome. This function relies on the recursive helper function recursive_palindrome().**

**The code for recursive_palindrome is missing a line at the indicated position. Choose the line which correctly completes the function.**

```
#define TRUE  1
#define FALSE 0

int is_palindrome(char word[])
{
 int len = strlen(word);

 return recursive_palindrome(word, 0, len - 1);
}


int recursive_palindrome(char word[], int start, int end)
{
 if (start >= end)
   return TRUE;
 else {
   /* MISSING RETURN STATEMENT: CHOOSE ONE THAT WORKS */
 }
}
```

**A.** return ((word[start] == word[end]) &&
recursive_palindrome(word, start + 1, end - 1));
**B.** return ((word[start] == word[end]) &&
recursive_palindrome(word, start, end));
**C.** return (word[start] == word[end]);
**D.** return ((word[start] == word[end]) ||
recursive_palindrome(word, start + 1, end - 1));
**E.** return (recursive_palindrome(word, start + 1, end - 1));
ANSWER: **A**

**6. What is the value of k at Point A?**

```
int i, j, k;

k = 0;

for (i = 0; i < 3; i++) {
  for (j = (3-i); j > 0; j--) {
    k = k + j;
  }
}
/* Point A */
...
```

**A.** 9
**B.** 10
**C.** 12
**D.** 14
**E.** 16
ANSWER: **B**

**7. Here are three consecutive lines from a C program:**

```
...
double income;
printf("Enter new income: ");
scanf("%lf", &income);          /* line A */
...
```

**Line A is which of the following?**
**I. An assignment statement**
**II. Initialization of income**
**III Declaration of income**

**A.** I only
**B.** II only
**C.** III only
**D.** All of I, II, and III
**E.** None of I, II, or III
ANSWER: **B**

**8.**
```
typedef struct {
  int x, y;
} Point;

typedef struct {
  int color;
  Point pos;
} ColoredPoint;
```

**ColoredPoint cp;**
**What is the correct code to move cp to the origin of the coordinate system?**

**A.**
```
cp->pos.x = 0;
cp->pos.y = 0;
```
**B.**
```
cp.pos->x = 0;
cp.pos->y = 0;
```
**C.**
```
cp->pos->x = 0;
cp->pos->y = 0;
```
**D.**
```
cp.pos.x = 0;
cp.pos.y = 0;
```
**E.**
```
cp->pos = {0, 0};
```
ANSWER: **D**
The -> notation is a shorthand which can only be used with pointers; there are no pointers here.
{ } can only be used with initializers, and not in assignment statements.

**9. We want code to set the diagonal entries of a 5 by 5 two dimensional array ("matrix") to 1; all other entries in the matrix are to be left unchanged. Which choice is correct?**

**Background: The "diagonal" entries of a matrix are those where the column number and row number are the same. The term is only used there are the same number of rows as columns. For example, in the following 3 by 3 matrix:**
**1 2 3**
**4 5 6**
**7 8 9**
**the diagonal entries have indices [0][0], [1][1], and [2][2], and the diagonal values are 1, 5, and 9.**

**A.**
```
int arr[5][5], i;
for (i = 0; i <= 5; i++)
   arr[i][i] = 1;
```
**B.**
```
int arr[5][5], i;
for (i = 0; i < 5; i++)
   arr[i][i] = 1;
```
**C.**
```
int arr[5][5], i;
for (i = 0; i < 5; i++)
   arr[i, i] = 1;
```
**D.**
```
int arr[5][5], i;
for (i = 0; i <= 5; i++)
   arr[i] = 1;
```
**E.**
```
int arr[5][5], i, j;
for (i = 0; i < 5; i++)
   for (j = 0; j < 5; j++)
      arr[i][j] = 1;
```
ANSWER: **B**

**10. Here's a truth table that is not completely filled in. Which of the choices gives the correct values for the last column?**

| A | B | !(A && !B) |
|---|---|---|
| T | T | |
| T | F | |
| F | T | |
| F | F | |

**A.** T
T
T
T

**B.** F
T
F
F

**C.** F
T
F
T

**D.** T
T
F
T

**E.** T
F
T
T

ANSWER: **E**

**11. What combination of values of a, b, and c make the following condition evaluate to 1 (or "true")? All variables are integers, although you can think of them as Booleans.**

**(!a || b) && (!b && c)**

**A.** a = 0, b = 0, c = 0
**B.** a = 0, b = 0, c = 1
**C.** a = 1, b = 1, c = 0
**D.** a = 0, b = 1, c = 1
**E.** a = 1, b = 0, c = 1

ANSWER: **B**

**12. Study the following function. Try to summarize what is does, and then choose the description which matches best.**

```
int followingFunction(int A[], int x, int y){
 int i;
 int z = 0;

 for (i=x; i<=y; i++) {
   z = z + A[i];
 }

 return z;
}
```

**A.** counts the number of elements between the xth and yth element (inclusive).
**B.** counts the number of elements between the xth and yth element (exclusive).
**C.** sums the elements from the xth to the yth (exclusive)
**D.** sums the elements from the xth to the yth (inclusive)
**E.** multiplies x times y

ANSWER: **D**

**13. Here are fragments illustrating three proposed solutions to the problem of swapping two integers. Which are valid?**

```
/************Proposal A */
void swap (int *a, int *b) {
   int temp;
   temp = *a;
   a = b;
   *b = temp;
}

int main (void) {
   int i, j;
...
   swap (&i, &j);
...
```

```
/************Proposal B */
 void swap (int a, int b) {
   int temp;
   temp = a;
   a = b;
   b = temp;
}

int main (void) {
   int i, j;
...
   swap (i, j);
...
```

```
/************Proposal C */
void swap (int *a, int *b) {
   int temp;
   temp = *a;
   *a = *b;
   *b = temp;
}

int main (void) {
   int i, j;
...
   swap (&i, &j);
...
```

**A.** A only
**B.** B only
**C.** C only
**D.** A and B
**E.** None of A, B, or C

ANSWER: **C**

**14. Recall that a pre-condition is "a condition assumed to be true before a function call" and a postcondition is "a condition assumed to be true after a function executes." (Hanly & Koffman p.131).**

**Assuming the following function is correct, what would be a proper precondition for it?**

```
double recL (double a, double w) {
   double q;
   q = a / w;
   return q;
}
```

**A.** a is greater than or equal to 0
**B.** w is greater than 0
**C.** q has been properly initialized
**D.** a and w have different values
**E.** the return value of the function is different from a
ANSWER: **B**

**15. Your prof sez that in the following fragment of a function, line A is OK but B will not compile. He's right. What's true that explains this?**

```
char courseName[7] = "CSE142";   /*line A*/
...
courseName = "142";    /*line B*/
```

**A.** It's purely a matter of style.
**B.** The length of the string in line B is wrong
**C.** Line A is a declaration
**D.** Line B is a declaration
**E.** The contents of an array cannot be changed once it has been assigned a value
ANSWER: **C**

**16. Recall that \n symbolizes the linefeed character, and \0 symbolizes the null character. Both of these are single characters in memory.**

**StringA contains:**
**Hello\n\0**

**String B contains:**
**world\n\0**

**The two are concatenated with the string library call**
**strcat (StringA, StringB);**

**What does the resulting string contain?**

**A.** Hello\n\world\n
**B.** Helloworld\n
**C.** Hello\0world\0
**D.** Hello\nworld\n\0
**E.** Hello\n\0world\n\0
ANSWER: **D**

**17. Suppose an array contains the following eight values:**

**-50 30 99 6 12 15 1 14**

**What is true?**

**A. The array satisfies the precondition for Linear Search**
**B. The array satisfies the precondition for Binary Search**
**C. The array satisfies the postcondition for Selection Sort**

**A.** A only
**B.** B only
**C.** C only
**D.** B and C only
**E.** All of A, B, or C
ANSWER: **A**

**18. Let's say we are applying binary search to the array of 6 values**

**10 18 19 22 30 60**

**The target is 30. To start the search, L is -1 and R is 6. After the first probe, what happens?**

**A.** L increases
**B.** L decreases
**C.** R increases
**D.** R decreases
**E.** L and R do not change throughout the algorithm
ANSWER: **A**

**19. When binary search is applied to an array of size 16, about 4 probes (tests of array elements) are needed to determine if a target value is in the array. Suppose an array has 20000 elements. About how many probes are needed?**

**A.** 15
**B.** 50
**C.** 200
**D.** 500
**E.** 5000
ANSWER: **A**

**20. Consider the following declarations and function prototypes:**

```
typedef struct {
        char type[50];
        double height;
        int age;
} tree;

int seed( int count, char *kind_of, tree grove[6] );

tree forest[ 27 ];
int height, age;
char type[] = "maple";
tree elm = { "elm", 25.6, 42 };
```

**Given the above information, which of the following pairs correctly calls function seed?**

**A.** seed( 13, forest[ 6 ].type, *forest[ 4 ] );
**B.** seed( elm.age, type, &forest );
**C.** seed( elm.age, type, forest[0] );
**D.** seed( forest[ 13 ].age, type, forest );
**E.** seed( age, type[ 0 ], &forest[ 2 ] );

ANSWER: **D**
The types of the three parameters are:
int
char *
array of tree
Arrays, as parameters, are treated like pointers. So when the function is called, the second argument could be a char *, or it could be an array of char (it cannot be a simple char). Likewise, third argument could be passed an array of tree (as in D, the correct answer), or as a pointer to a tree.

This question was a bit trickier than we really meant it to be. Still, D is the best answer. The other choices would give errors or warnings based on type mismatches.

**21. What output would you get if you made the function call recursiveFunction(6)?**

```
int recursiveFunction(int num)
{
    if (num == 0)
    {
        return num;
    }
    else
    {
        num = -1 + recursiveFunction(num);
        printf("%d", num);
        return num;
    }
}
```

**A.** 6543210
**B.** 654321
**C.** 666666
**D.** 0123456
**E.** There will be an infinite recursion
ANSWER: **E**