Question 1 [19 marks]

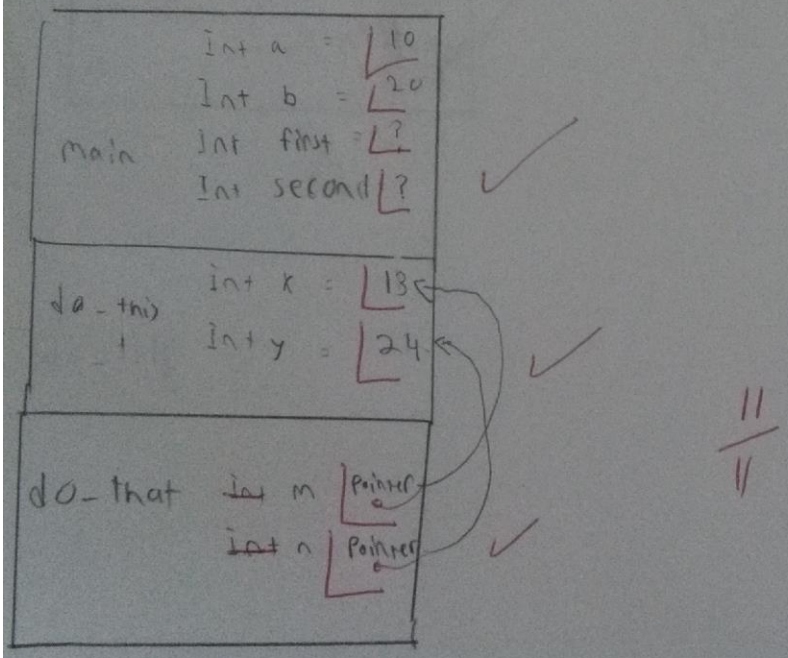
```
int do_that(int *m, int *n)
{
    *m = *m + 2;
    *n = *n + 2;   // Point A
    return *m + *n;
}

int do_this(int x, int y)
{
    x = x + 1;
    y = y + 2;
    return do_that(&x, &y);
}
```

```
int main(void)
{
    int a = 10;
    int b = 20;
    int first, second;
    first = do_this(a, b);
    second = do_that(&a, &b);
    return 0;
}
```

(a) Draw a memory diagram similar to one that would be produced by the C Tutor immediately after the statement at Point A has been executed for the the **first** time, that is, immediately after the statement, `*n = *n + 1;` is executed for the first time, but before the function returns.

Stack

Heap



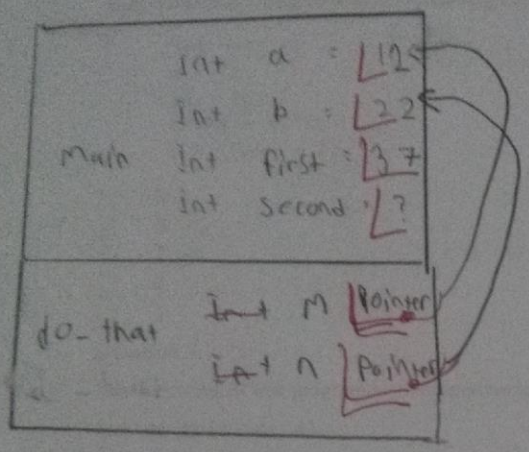
11/11

Page total 11 Running Total 11

(b) Draw a memory diagram similar to one that would be produced by the C Tutor immediately after the statement at Point A has been executed for the the first time, that is, immediately after the statement, `*n = *n + 1;` is executed for the first time, but before the function returns.

Stack

Heap



8 18

Multiple-choice Questions [32 marks] 28

For Questions 2 through 17, put a check mark, ✔, or an ✗, in the box, ☐, beside the correct answer. For each question, 2 marks will be awarded for selecting the correct answer and 0 marks will be awarded for selecting an incorrect answer. 0 marks will be awarded if you select more than one answer for a question, even if one of the answers is correct.

Question 2

What value will x contain after this code fragment is executed?

```
int *px, x;
x = 5;
px = &x;
x = x + 1;
x = (*px) + 2;
```

- ☐ 9
- ☑ 8
- ☐ 7
- ☐ 6
- ☐ 5

Question 3

The execution of this program is visualized using C Tutor:

```
int do_this(int a)
{
    return 2 * a;
}

int do_that(int x, int y)
{
    int a = do_this(x) + y;  // Line A
    return a;
}

int main(void)
{
    int a, b;
    a = 3;
    b = 5;
    printf("%d\n", do_this(a));
    printf("%d\n", do_that(a, b));
    return 0;
}
```

Immediately after `int a = do_this(x) + y;` (see Line A) is executed, but before `return a;` is executed, how many variables named a appear in the memory diagram produced by C Tutor?

- ☐ 0
- ☐ 1
- ☑ 2
- ☐ 3
- ☐ 4

Question 4

Assuming that `timer_t` is the `typedef`'d name of a `struct` "type", how many instances of the `struct` will we have when the following code is executed?

```
timer_t *first_timer = malloc(sizeof(timer_t));
timer_t *second_timer = malloc(sizeof(timer_t));
timer_t third_timer = *second_timer;
timer_t *fourth_timer = first_timer;
```

- ☐ 0
- ☐ 1
- ☐ 2
- ☑ 3
- ☐ 4

Question 5

In this program, function `f` has a single parameter, `x`, of type `double`. Its argument, `k`, has type `int`.

```
double f(double x)
{
    return 3 * x;
}

int main(void)
{
    int k = 7;
    double y;

    y = f(k);
    return 0;
}
```

Select the correct statement:

- ☑ This program causes a compilation error, because the function parameter and argument have different types.
- ☐ This program terminates with a run-time error, because the function parameter and argument have different types.
- ☐ Function `f` returns `21`.
- ☐ Function `f` returns `21.0`.
- ☐ Function `f` returns another value.

Question 6

This function is intended to return a new array containing the first n integers in `arr`, but some code has been deleted.

```
int *copy_array(int arr[], int n)
{
    // Deleted code goes here.

    for (int i = 0; i < n; i += 1) {
        copy[i] = arr[i];
    }
    return copy;
}
```

Identify the deleted code:

- [ ] `int copy[n];`
- [ ] `int *copy = malloc(n);`
- [ ] `int *copy = malloc(sizeof(arr));`
- [x] `int *copy = malloc(n * sizeof(arr[n-1]));`
- [ ] `int *copy = malloc(sizeof(arr) / sizeof(arr[0]) * sizeof(int));`
- [ ]
```
int *copy;
for (int i = 0; i < n; i += 1) {
    copy[i] = malloc(sizeof(int));
}
```

Question 7

The statement:

```
double *x;
```

- [ ] declares a variable named `*x` with type `double`.
- [x] declares a variable named `x` with type "pointer-to-`double`".
- [ ] declares a variable that can be multiplied by a `double`.
- [ ] declares an array that can store values of type `double`.

Question 8

What does this program output?

```
int f(int *x)
{
    *x = *x + 1;
    return *x;
}

int g(int x)
{
    x = x + 1;
    return x;
}

int main(void)
{
    int x = 10;
    int y = g(f(&x));
    printf("%d %d\n", x, y);
    return 0;
}
```

- [ ] 10 11
- [ ] 10 12
- [ ] 11 11
- [x] 11 12
- [ ] 12 12
- [ ] None of the above.

Question 9

What does this program output?

```
int *f(int *x)
{
    *x = *x + 1;
    return x;
}

int main(void)
{
    int x = 10;
    int y = *f(f(f(&x)));
    printf("%d %d\n", x, y);
    return 0;
}
```

- [x] 14 14
- [ ] 11 11
- [ ] 10 14
- [ ] 10 11
- [ ] None of the above.

Question 11

This function is supposed to return the location (index) of `target` in the first `n` elements in array `arr`, or `-1` if `target` is not found in the array. One line of code has an error.

```
1    int sequential_search(int arr[], int n, int target)
2    {
3        int location = -1;
4        for (int i = 0; arr[i] != target; i = i + 1) {
5            location = i + 1;
6            if (i == n - 1)
7                return i - n;
8        }
9        return location;
10   }
```

Which line has the bug?

- [x] Line 3
- [ ] Line 4
- [ ] Line 5
- [ ] Line 6
- [ ] Line 7

Questions 12 through 15: these questions deal with four implementations of functions that are intended to return the sum of the first `n` integers in an array.

Question 12

```
int sum_array_A(int arr[], int n)
{
    int sum = 0;
    int *p = arr + n;
    while (arr != p) {
        sum = sum + *arr;
        arr = arr + 1;
    }
    return sum;
}
```

- [x] This function causes a compilation error.
- [ ] This function causes the program to terminate when it executes.
- [ ] This function returns the correct result.
- [ ] This function returns an incorrect result.

Question 13

```
int sum_array_B(int arr[], int n)
{
    int sum = 0;
    int i = n;
    while (i >= 0) {
        sum = sum + arr[n - i];
        i = i - 1;
    }
    return sum;
}
```

Will crash on execution ← probably not
if n is equal to the length
of the array

will produce an incorrect result
if n < the length of the array

- ☐ This function causes a compilation error.
- ☐ This function causes the program to terminate when it executes.
- ☐ This function returns the correct result.
- ☑ This function returns an incorrect result. ✓

Question 14

```
int sum_array_C(int arr[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i = i + 1) {
        sum = sum + *(&arr[0] + i);
    }
    return sum;
}
```

- ☐ This function causes a compilation error.
- ☐ This function causes the program to terminate when it executes.
- ☑ This function returns the correct result. ✓
- ☐ This function returns an incorrect result.

Question 15

```
int sum_array_D(int arr[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i = i + 1) {
        sum = sum + arr[i];
        i = i + 1;
    }
    return sum;
}
```

- ☐ This function causes a compilation error.
- ☐ This function causes the program to terminate when it executes.
- ☐ This function returns the correct result.
- ☑ This function returns an incorrect result. ✓

Question 16

Consider this code fragment:

```
typedef struct {
    int x;
    int y;
} point_t;
...
point_t *pt1 = malloc(sizeof(point_t));
*pt1 = (point_t) {10, 20};
point_t *pt2 = pt1;
```

What does the statement `point_t *pt2 = pt1;` do?

- ☐ It stores a copy of the struct pointed to by `pt1` in `pt2`.
- ☐ It creates, on the heap, a copy of the struct pointed to by `pt1` and stores a pointer to the copy in `pt2`.
- ☐ It stores the address of `pt1` in `pt2`.
- ☑ It causes `pt2` and `pt1` to point to the same struct.
- ☐ It probably damages the program's heap.

Question 17

A program calls `malloc` to allocate an array of `n` integers from the heap, and stores the pointer returned by `malloc` in a variable named `p`. After the program has finished using the array, it should deallocate the memory by calling `free`; for example, `free(p);`

Select the statement that is functionally equivalent to `free(p);`

- ☐ `free(*p);`
- ☐ `free(&p);`
- ☐ `free(p[n]);`
- ☐ `free(*p[n]);`
- ☐ `free(&p[n]);`
- ☐ `free(p[0]);`
- ☐ `free(*p[0]);`
- ☑ `free(&p[0]);`

Crib Sheet

`void *malloc(int size);`

Allocates *size* bytes from the heap and returns a pointer to the memory. The memory is not initialized. On error, the function returns `NULL`.

`void free(void *ptr);`

Frees the memory pointed to by `ptr`, which must have been returned by a previous call to `malloc`. Otherwise, or if `free(ptr)` has already been called, undefined behaviour occurs. If *ptr* is `NULL`, no operation is performed.