Question 1 [14 marks]

(a) (2 marks) Variable `n` (of type `int`) has been initialized. Write the C code to declare a variable named `term` (of type `double`) and initialize it with the real number calculated using the formula:

$$\frac{\sqrt{3^n}-1}{n^2+5}$$

Don't write a complete C function or the code to initialize `n`. Just write the required statements.

(b) (3 marks) Here is the formula for a *harmonic series*:

$$\sum_{n=1}^{\infty}\frac{1}{n}=1+\frac{1}{2}+\frac{1}{3}+\ldots$$

Variable `k` (of type `int`) has been initialized with a positive integer. Write the C code to declare a variable named `sum` and initialize it with the sum of the first *k* terms of the harmonic series:

$$\sum_{n=1}^{k}\frac{1}{n}=1+\frac{1}{2}+\frac{1}{3}+\ldots\frac{1}{k}$$

Don't write a complete C function or the code to initialize `k`. Just write the required statements.

(c) (5 marks) The Maclaurin series for the function $ln(1 + x)$ is given by the formula:

$$\ln(1+x)=\sum_{n=1}^{\infty}\frac{(-1)^{n+1}}{n}x^n,-1<x\le 1$$

The partial sum of this series is described by the formula:

$$\ln(1+x)=\sum_{n=1}^{k}\frac{(-1)^{n+1}}{n}x^n=x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\ldots+\frac{(-1)^{k+1}x^k}{k},-1<x\le 1$$

Write a function named `maclaurin` that calculates and returns the partial sum of this series for specified values of *x* and *k*. The function prototype is:

```
double maclaurin(double x, int k);
```

Your function should terminate via `assert` if `x` is less than or equal to -1 or greater than 1. Your function should terminate via `assert` if `k` is 0 or negative.

(d) (4 marks) Write a function named `test_maclaurin` that calls the `maclaurin` function you wrote for part (c). Here is the function prototype:

```
void test_maclaurin(double x, int k, double expected_result);
```

Parameters `x` and `k` are the arguments that will be passed to `maclaurin`. Parameter `expected_result` is the partial sum that a correct implementation of `maclaurin` should return when it is called with those values of `x` and `k`.

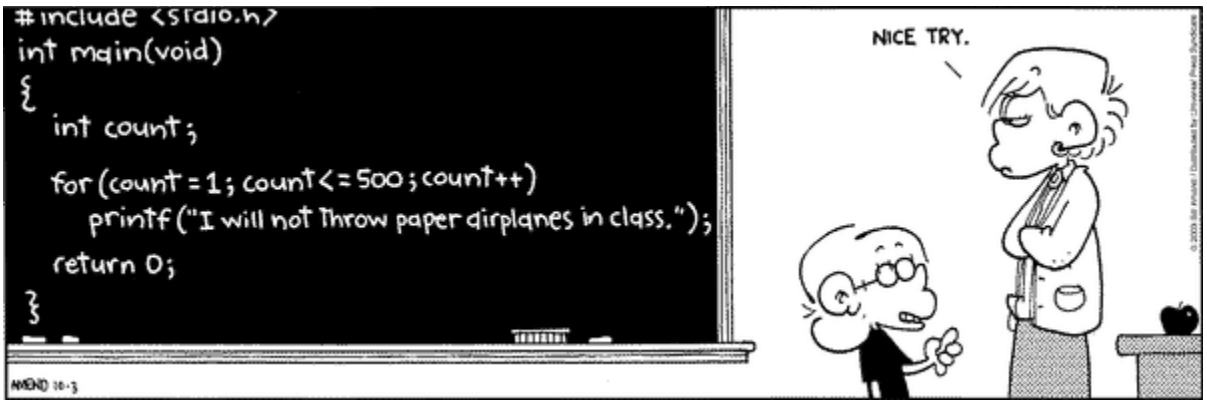
Suppose we call `test_maclaurin` this way, to verify that `maclaurin(0.5, 100)` returns (approximately) 0.4055.

```
maclaurin(0.5, 100, 0.4055);
```

When called this way, `test_maclaurin` should print:

```
Calling maclaurin, parameter x: 0.5, parameter k: 100
Expected sum: 0.4055
Actual sum: the actual sum calculated by maclaurin()
```

Don't worry about formatting the real numbers exactly as shown here. It doesn't matter how many digits your function prints after the decimal point.



Question 2 [12 marks]

(a) (5 marks) Write a function named `either_24` that is passed an array containing `n` integers. The function prototype is:

```
_Bool either_24(int arr[], int n);
```

Assume that parameter `n` will always be positive. (Your function does not have to check this). The function returns `true` if array `arr` contains either a 2 next to a 2 or a 4 next to a 4, but not both. For example,

- if `arr` contains `{1, 2, 2}` the function returns `true`;
- if `arr` contains `{3, 2, 4, 4, 1}` the function returns `true`;
- if `arr` contains `{4, 4, 1, 2, 2}` the function returns `false`;
- if `arr` contains `{1, 2, 1, 2}` the function returns `false`;
- if `arr` contains `{1, 3, 5}` the function returns `false`.

(b) (7 marks) Write a function named `series_up` that is passed an integer `n`. The function prototype is:

```
int *series_up(int n);
```

Assume that parameter `n` will always be positive. (Your function does not have to check this).

The function returns a pointer to an array that has been dynamically allocated from the heap and is initialized with integers that follow this pattern:

```
1,   1,2,   1,2,3,   ...,   1,2,3,4,...,n
```

(The extra spaces shown here are to help you see the pattern, and are not stored in the array.)

For example, if `n` is 5, the function returns a pointer to this array:

```
[1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5]
```

Note: the required capacity of the array is `n(n+1)/2`.

Question 3 [6 marks]

Answer this question on the question paper.

Part 1 (3 marks)

Consider this program:

```
void mystery(int a[], int n)
{
    int *p;
    p = a;
    *p = a[0] + a[n-1];
}

int main(void)
{
    int arr[] = {3, 7, 2, 9, 1, 6};
    mystery(arr, 6);
    exit(0);
}
```

Which of the memory diagrams A, B, C,and D on the next page best depicts the program's state immediately after the statement:

```
*p = a[0] + a[n-1];
```

in function `mystery` is is executed, but before the function returns to `main`?

Circle the correct answer: **A** **B** **C** **D**

Diagram A

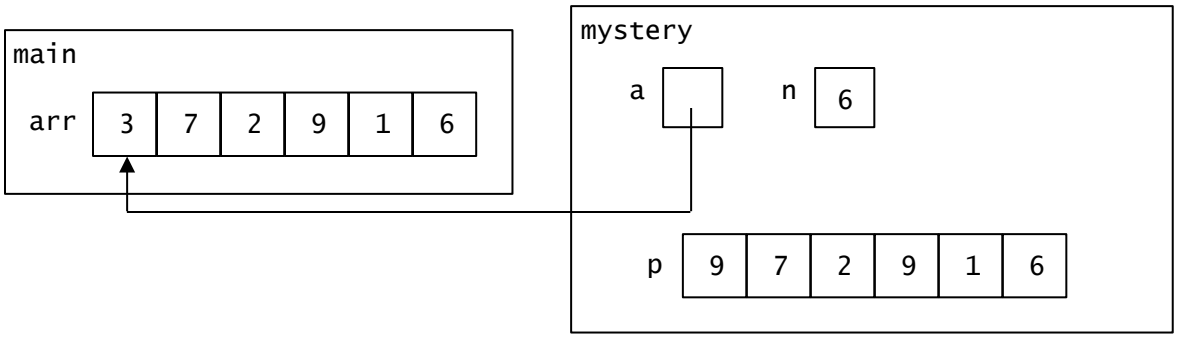


Diagram B

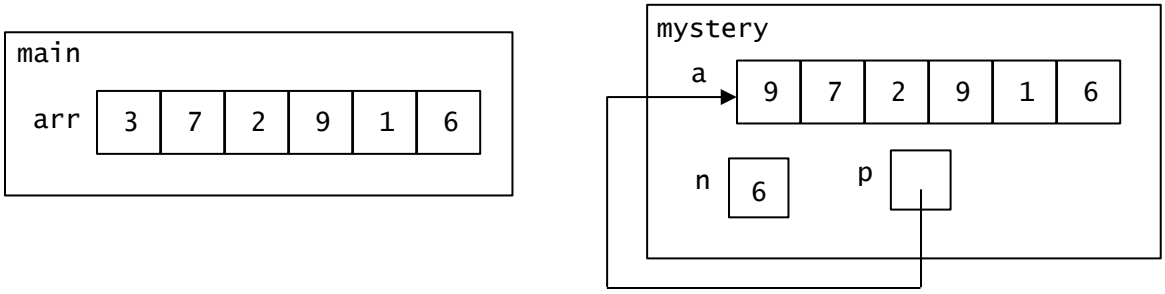


Diagram C

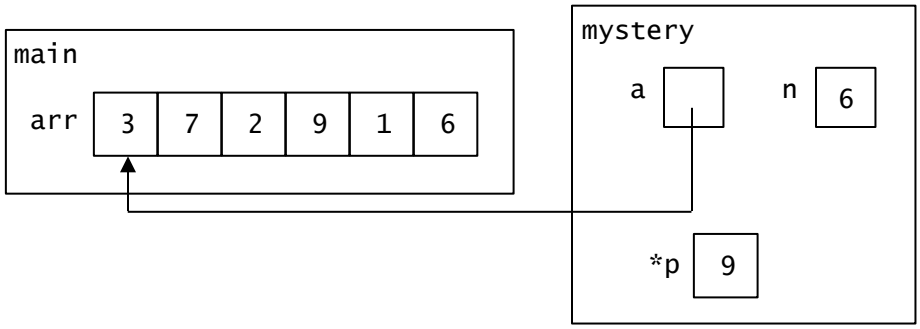
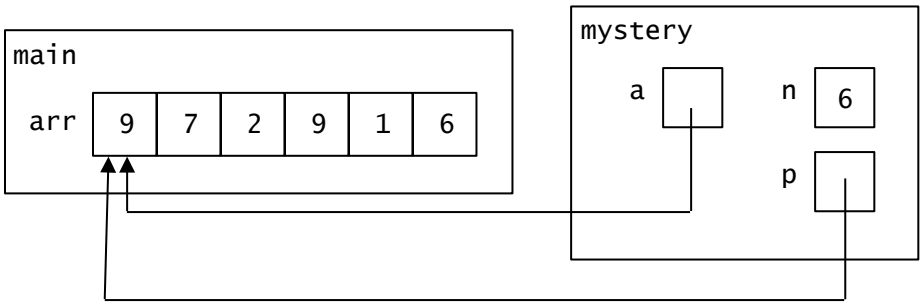


Diagram D



Part 2 (3 marks)

Consider this program:

```
#include <math.h>

/* Coordinates of a two-dimensional point. */
struct point {
    int x;
    int y;
};

typedef struct point point_t;

double distance(point_t *start, point_t *end)
{
    double dist = sqrt(pow(start->x - end->x, 2) +
                       pow(start->y - end->y, 2));
    return dist;
}

int main(void)
{
    point_t pt1, pt2;
    pt1.x = 1;
    pt1.y = 2;

    pt2.x = 4;
    pt2.y = 6;

    double length = distance(&pt1, &pt2);
    exit(0);
}
```

Which of the memory diagrams A, B, C, D and E on the following pages best depicts the program's state immediately after the statement:

```
    double dist = sqrt(pow(start->x - end->x, 2) +
                       pow(start->y - end->y, 2));
```

in function `distance` is executed, but before the `return` statement is executed?

Circle the correct answer: **A** **B** **C** **D** **E**

Diagram A

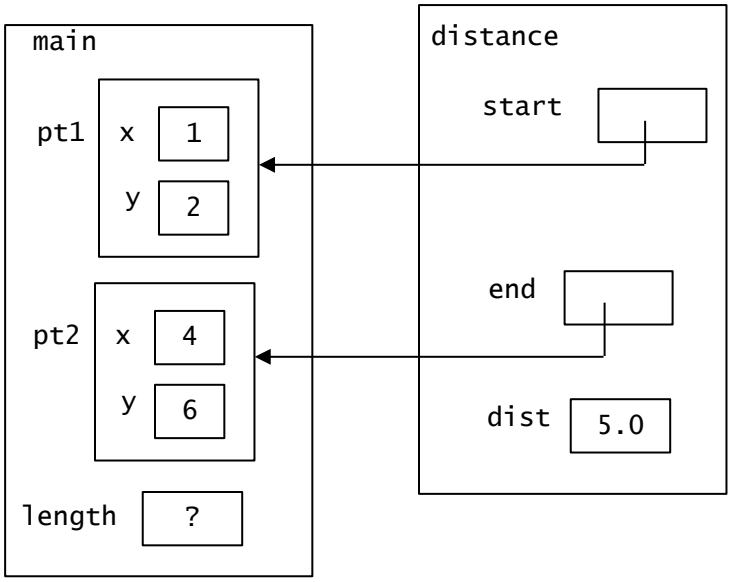
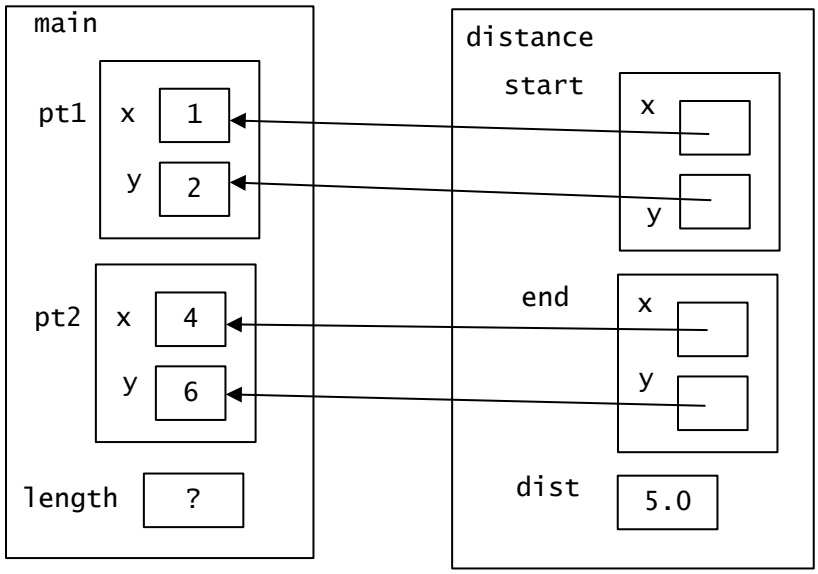


Diagram B



Diagrams C, D and E are on the following pages.

Diagram C

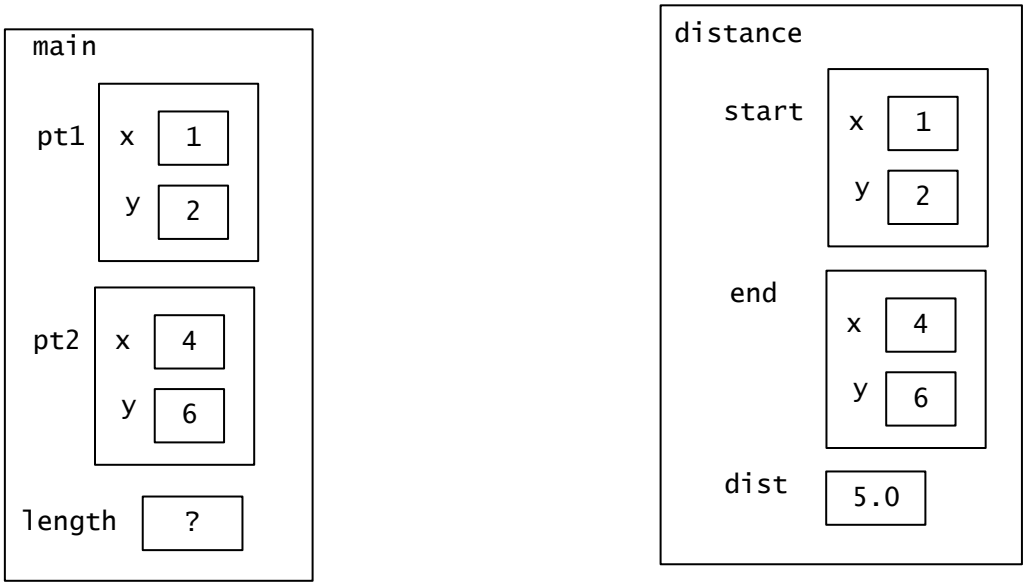


Diagram D

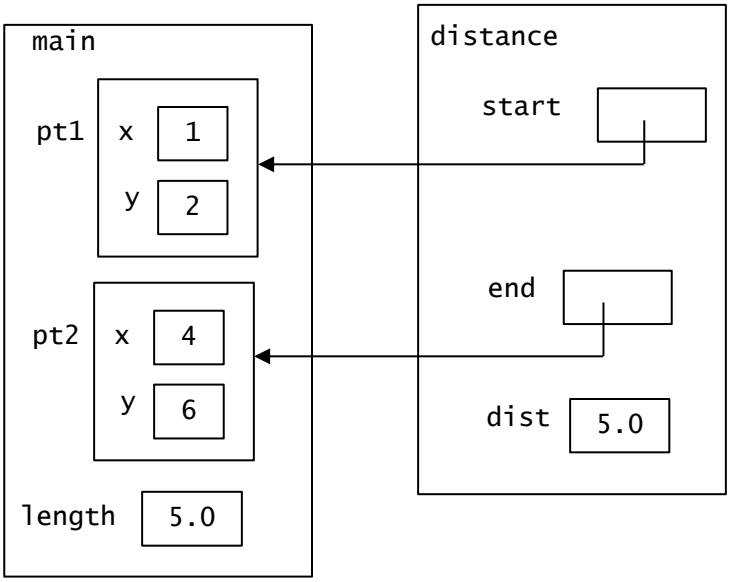
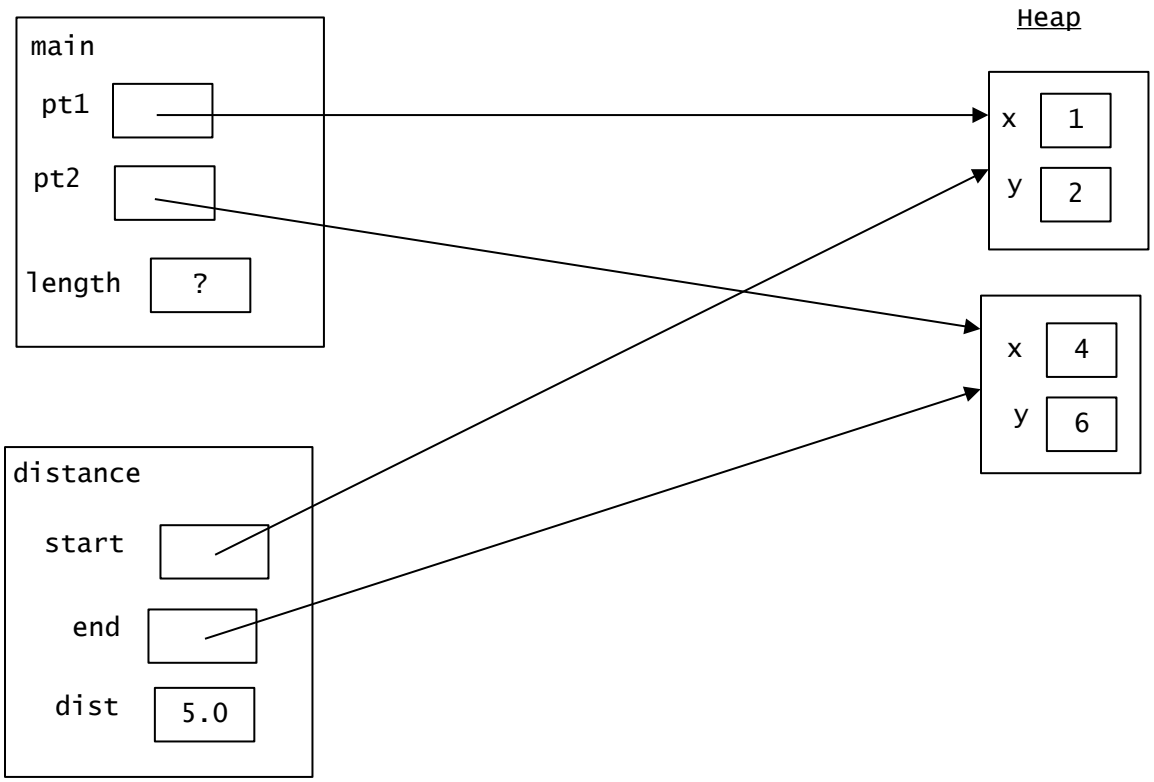


Diagram E is on the next page.

Diagram E


Heap
main
pt1
pt2
length
?
x
1
y
2
x
4
y
6
distance
start
end
dist
5.0

Question 4 [18 marks]

Here is the declaration for nodes in a singly-linked list of integers:

```
struct intnode {
    int    value;
    struct intnode *next;
};

typedef struct intnode IntNode;
```

Here is the `intnode_construct` function that was used in several lectures and labs. It returns a pointer to a dynamically allocated `IntNode`:

```
IntNode *intnode_construct(int value, IntNode *next)
{
    IntNode *p = malloc(sizeof(IntNode));
    assert (p != NULL);
    p->value = value;
    p->next = next;
    return p;
}
```

(a) (9 marks) Write a function named `interleave_lists` that is passed pointers to two singly-linked lists. The function prototype is:

```
IntNode *interleave_lists(IntNode *head_1, IntNode *head_2);
```

Parameters `head_1` and `head_2` point to the first nodes in the first and second linked lists, respectively. The function must not alter those linked lists. The function builds a new linked list that contains the integer from the first node in the first linked list, followed by the integer from the first node in the second linked list, followed by the integer from the second node in the first linked list, followed by the integer from the second node in the second linked list, and so on. The function returns a pointer to the first node in the new linked list.

The linked lists pointed to by `head_1` and `head_2` may have different lengths. One of the linked lists may be empty (contain no nodes), in which case the new linked list will be a copy of the non-empty list. The function will return a `NULL` pointer if it is passed pointers to two empty linked lists.

The nodes in the new linked list nodes must be dynamically allocated from the heap (call `intnode_construct` to do this).

Example: suppose `head_1` points to a linked list containing three integers: 1, 3, 5, and `head_2` points to a linked list containing 5 integers: 10, 11, 12, 13, 14. The new linked list will contain 8 integers, in this order: 1, 10, 3, 11, 5, 12, 13, 14. Notice how, in the new linked list, the integers copied from the first linked list have been interleaved with the first three integers copied from the second linked list.

(b) (9 marks) Write a function named `remove` that is passed a pointer to a singly-linked list, and removes all nodes containing a specified value from the list. The function prototype is:

```
IntNode *remove(IntNode *head, int value);
```

Parameter `head` points to the first node in the linked list. Memory for all of the nodes was allocated by `intnode_construct`. All nodes containing `value` must be removed from the linked list, and the memory used by those nodes must be deallocated.

The function returns a pointer to the first node in the modified linked list. The function will return a `NULL` pointer if it is passed a pointer to an empty linked list (one with no nodes).

Selected Functions from the C Standard Library

assert.h

```
assert(expression);
```

If *expression* evaluates to `true`, `assert` does nothing. If If *expression* evaluates to `false`, `assert` causes the program to display information and terminate. The information displayed includes the text of the expression, the name of the source file, the source line, and the name of the enclosing function.

math.h

```
float powf(float x, float y);
double pow(double x, double y);
```

Returns the value of `x` raised to the power of `y`.

```
float sqrtf(float x);
double sqrt(double x);
```

Returns the square root of `x`.

```
float fabsf(float x);
double fabs(double x);
```

Returns the absolute value of `x`. To calculate the absolute value of an integer, call `abs` (see `stdlib.h`).

stdlib.h

```
int abs(int x);
```

Returns the absolute value of `x`. To calculate the absolute value of a real number, call f`absf` or `fabs` (see `math.h`).

```
void *malloc(int size);
```

Allocates a block of memory with the specified size from the heap and returns a pointer to the block. Returns the `NULL` pointer if the block cannot be allocated.

```
void free(void *ptr);
```

Causes the memory pointed to by `ptr` to be deallocated (made available for subsequent allocation by `malloc`.) The function does nothing if `ptr` is `NULL`. The behaviour of this function is undefined if `ptr` is not a pointer that was previously returned by `malloc`. The behaviour of this function is undefined if the memory pointed to by `ptr` was previously deallocated by a call to `free`.