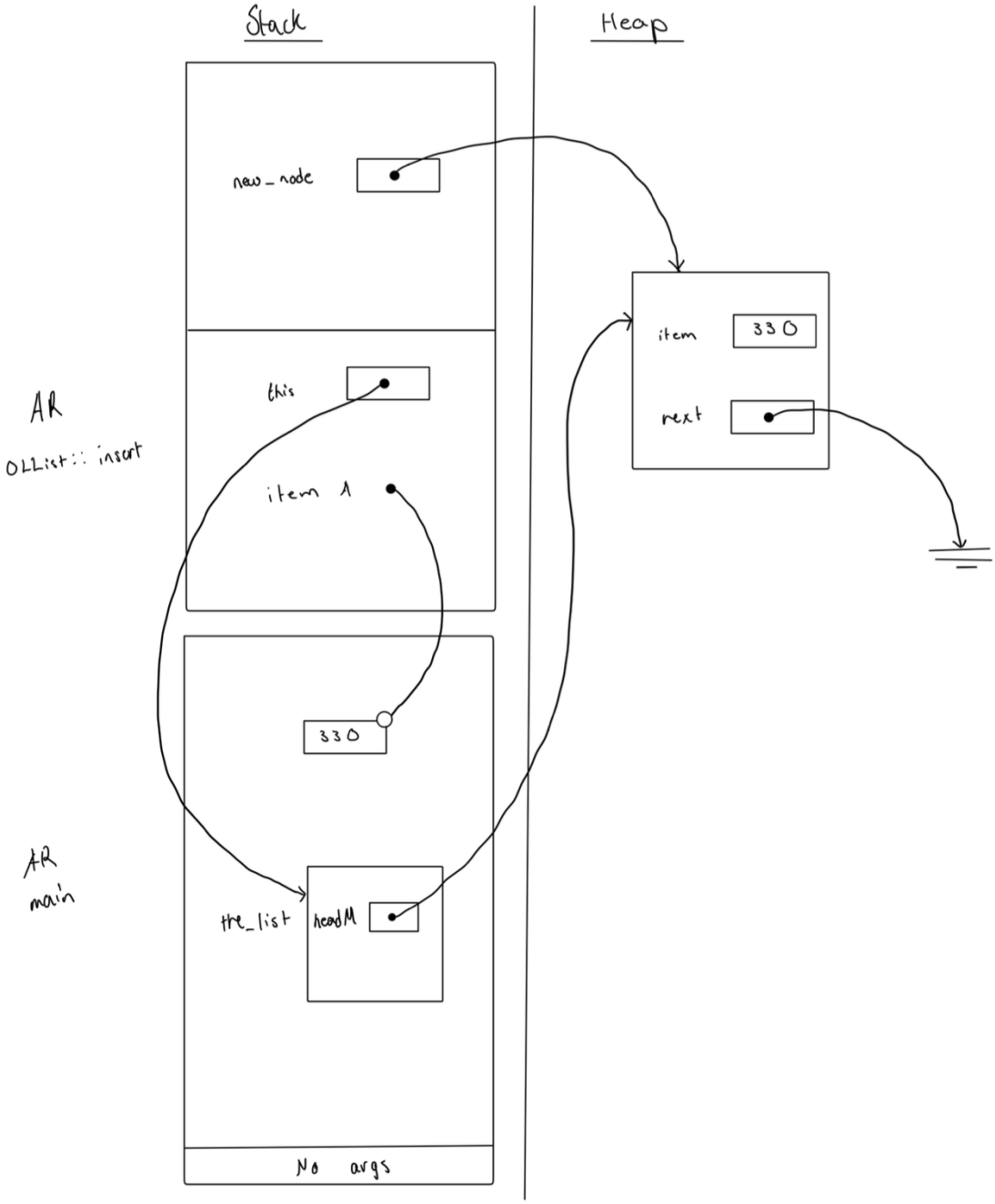
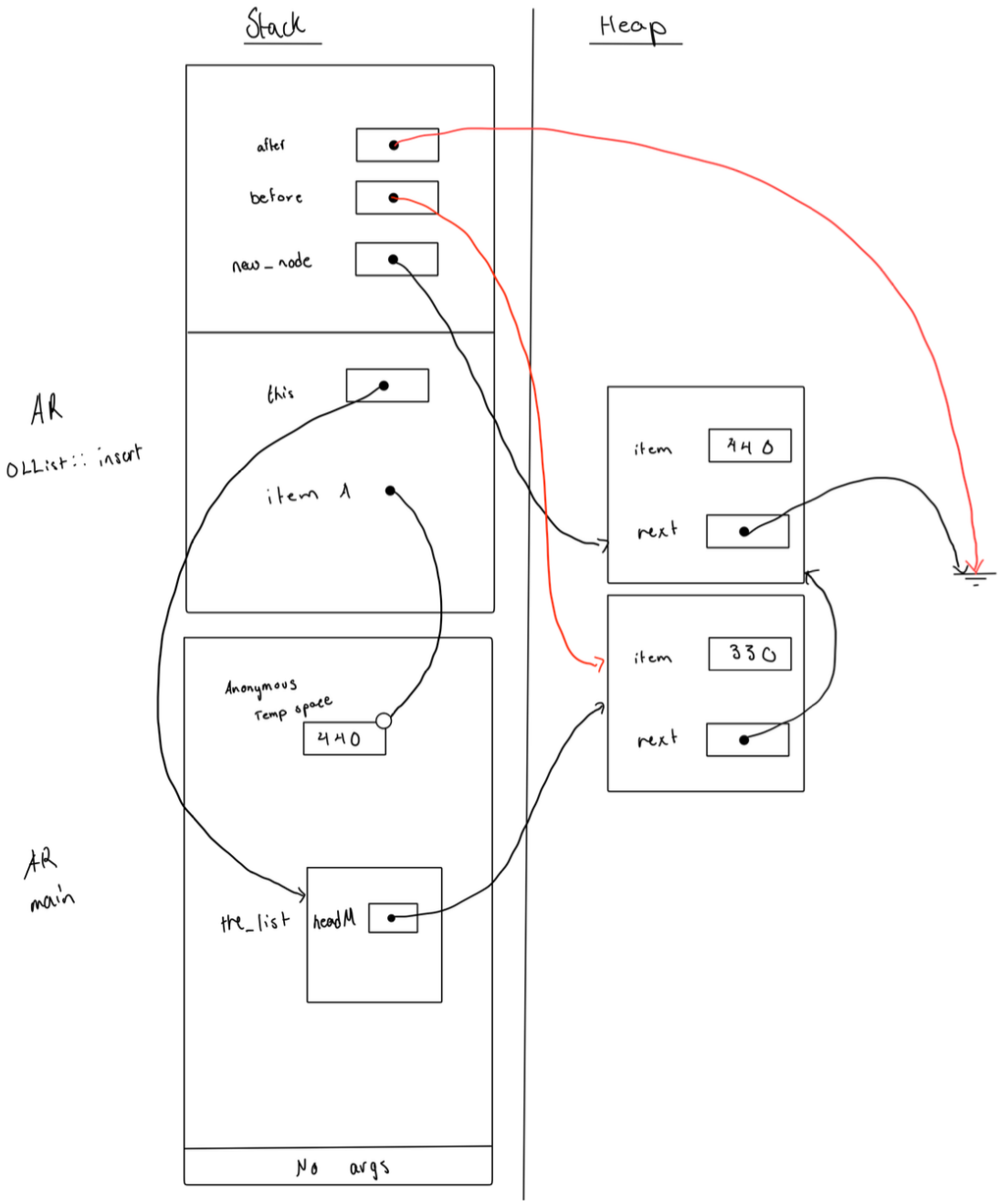
Course: Programming Fundamental – ENSF 337
Lab #: Lab 8
Instructor: M. Moussavi
Student Name: Carl Soriano
Lab Section: B01
Date submitted: November 25, 2022

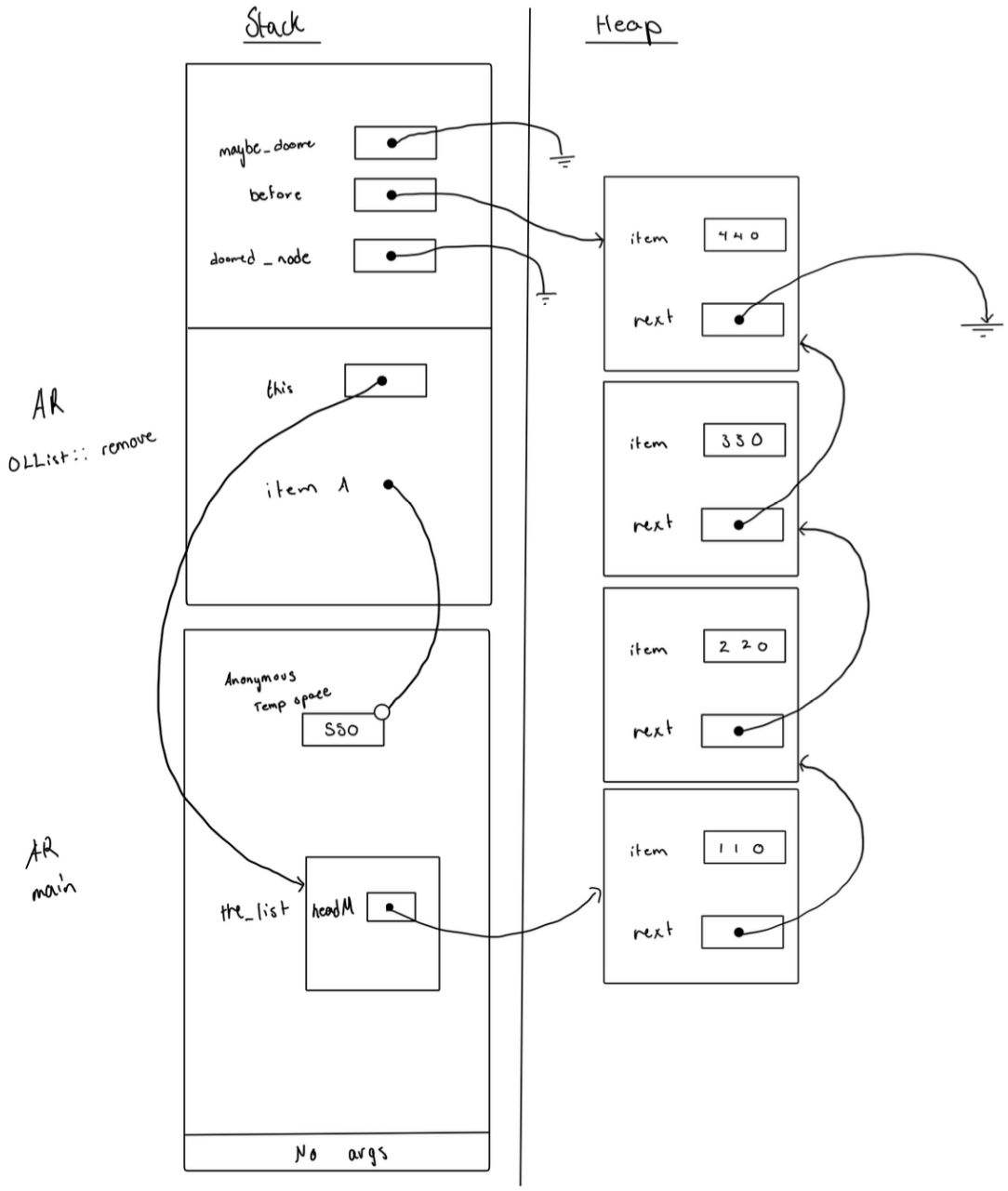
Exercise A



Point 1


Stack
Heap
after
before
new_node
this
item 1
AR
OLList:: insert
Anonymous
Temp space
440
item
440
next
item
330
next
AR
main
the_list
headM
No args

Point 2


Stack
Heap
maybe_doome
before
doomed_node
this
item 1
Anonymous
Temp space
550
the_list
headM
No args
item
440
next
item
330
next
item
220
next
item
110
next
AR
OLList:: remove
AR
main

Point 3

Exercise B

```cpp
// OLList.cpp
// ENSF 337 Fall 2021 Lab 8 Exercise A and B

#include <iostream>
#include <stdlib.h>
using namespace std;
#include "OLList.h"

OLList::OLList(): headM(0){}

OLList::OLList(const OLList& source)
{
    copy(source);
}

OLList& OLList::operator =(const OLList& rhs)
{
    if (this != &rhs) {
        destroy();
        copy(rhs);
    }
    return *this;
}

OLList::~OLList()
{
    destroy();
}

void OLList::print() const
{
    cout << '[';
    if (headM != 0) {
        cout << ' ' << headM->item;
        for (const Node *p = headM->next; p != 0; p = p->next)
            cout << ", " << p->item;
    }
    cout << " ]\n";
}

void OLList::insert(const ListItem& itemA)
{
    Node *new_node = new Node;
    new_node->item = itemA;

    if (headM == 0 || itemA <= headM->item ) {
        new_node->next = headM;
        headM = new_node;
        // point one
    }
    else {
        Node *before = headM;
        Node *after = headM->next;
        while(after != 0 && itemA > after->item) {
            before = after;
            after = after->next;
        }
        new_node->next = after;
        before->next = new_node;
    }
}
```

```cpp
void OLList::copy(const OLList& source)
{
    if (source.headM == 0) {
        headM = 0;
        return;
    }
    headM = new Node;
    Node *newest_node = headM;

    const Node *source_node = source.headM;
    while (true) {
        newest_node->item = source_node->item;
        source_node = source_node->next;
        if (source_node == 0)
            break;
        newest_node->next = new Node;
        newest_node = newest_node->next;
    }
    newest_node->next = 0;

}

void OLList::remove(const ListItem& itemA)
{
    if (headM == 0 || itemA < headM->item)
        return;

    Node *doomed_node = 0;

    if (itemA == headM->item) {
        doomed_node = headM;
        headM = headM->next;
    }
    else {
        Node *before = headM;
        Node *maybe_doomed = headM->next;
        while(maybe_doomed != 0 && itemA > maybe_doomed->item)
{
            before = maybe_doomed;
            maybe_doomed = maybe_doomed->next;
        }
        if(maybe_doomed == 0 || maybe_doomed->item != itemA)
            return;
        else {
            before->next = maybe_doomed->next;
            doomed_node = maybe_doomed;
        }
    }

    delete[] doomed_node;
    }


void OLList::destroy()
{
    Node *b = headM;
    Node *before;

    while(b != 0){
        before = b;
        b = b->next;
        delete before;
    }
    headM = 0;
}
```

```
List just after creation. expected to be [ ]
[ ]
the_list after some insertions. Expected to be: [ 99, 110, 120,
    220, 330, 440, 550 ]
[ 99, 110, 120, 220, 330, 440, 550 ]
testing for copying lists ...
other_list as a copy of the_list: expected to be [ 99, 110, 120,
    220, 330, 440, 550 ]
[ 99, 110, 120, 220, 330, 440, 550 ]
third_list as a copy of the_list: expected to be: [ 99, 110, 120,
    220, 330, 440, 550 ]
[ 99, 110, 120, 220, 330, 440, 550 ]
testing for removing and chaining assignment operator...
the_ist after some removals: expected to be: [ 99, 110, 120, 220,
    440 ]
[ 99, 110, 120, 220, 440 ]
printing other_list one more time: expected to be: [ 99, 110, 120,
    220, 330, 440, 550 ]
[ 99, 110, 120, 220, 330, 440, 550 ]
printing third_list one more time: expected to be: [ 99, 110, 120,
    220, 330, 440, 550 ]
[ 99, 110, 120, 220, 330, 440, 550 ]
chaining assignment operator ...
the_list after chaining assignment operator: expected to be: [ 99,
    110, 120, 220, 440 ]
[ 99, 110, 120, 220, 440 ]
other_list after chaining: expected to be: [ 99, 110, 120, 220, 440
    ]
[ 99, 110, 120, 220, 440 ]
third_list after chaining: expected to be: [ 99, 110, 120, 220, 440
    ]
[ 99, 110, 120, 220, 440 ]
Program ended with exit code: 0
```

Exercise C

In ZIP folder