Julius Ranoa
CSC 121 001 Computer Science I
Homework – Chapter 10 Pointers

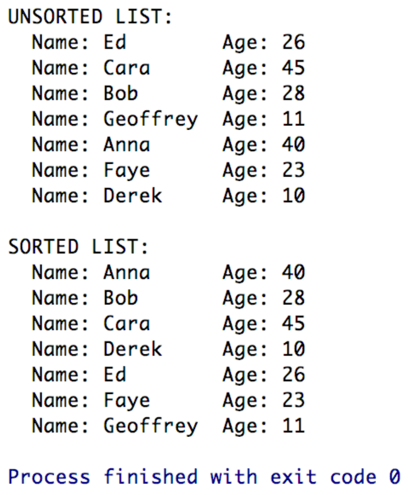
Part I. Review Questions @ Page 697.
Qn. 1 – 11, 13.

1. Each byte in memory is assigned a unique address.
2. The & operator can be used to determine a variable's address.
3. Pointer variables are designed to hold addresses.
4. The * operator can be used to work with the variable a pointer points to.
5. Array names can be used as pointers and vice versa.
6. Creating variables while a program is running is called dynamic memory allocation.
7. The new operator is used to dynamically allocate memory.
8. If the *new* operator cannot allocate the amount of memory requested, it throws exception.
9. A pointer that contains the address 0 is called a(n) null pointer.
10. When a program is finished with a chunk of dynamically allocated memory, it should free it with the delete operator.
11. You should only use the *delete* operator to deallocate memory that was dynamically acquired with the new operator.

13. Look at the following code.
```
int x = 7;
int *ptr = &x;
```
What will be displayed if you send the expression `*iptr` to `cout`? 7
What happens if you send the expression `ptr` to `cout`? The address of x in memory.

Part II. Programming Challenge @ Page 701.
Qn. 13 – Indirect Sorting Through Pointers #1.

Screenshot of Runtime:



```
UNSORTED LIST:
  Name: Ed        Age: 26
  Name: Cara      Age: 45
  Name: Bob       Age: 28
  Name: Geoffrey  Age: 11
  Name: Anna      Age: 40
  Name: Faye      Age: 23
  Name: Derek     Age: 10

SORTED LIST:
  Name: Anna      Age: 40
  Name: Bob       Age: 28
  Name: Cara      Age: 45
  Name: Derek     Age: 10
  Name: Ed        Age: 26
  Name: Faye      Age: 23
  Name: Geoffrey  Age: 11

Process finished with exit code 0
```

Source Code:

1. *PersonList.h*
2. *PersonList.cpp*
3. *main.cpp*

The source code is also stored at Github.
Link below:
https://github.com/TheLoneWoof1102/FA17_CSC121001/tree/master/Source%20Code/

main.cpp

```
#include "PersonList.h"
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    PersonList pl;

    // TEST DATA.
    srand(time(NULL));
    string testNames[] = {
        "Ed", "Cara", "Bob", "Geoffrey", "Anna", "Faye", "Derek"
    };
    int size = sizeof(testNames) / sizeof(testNames[0]);
    pl.setSize(size);
    for (int i = 0; i < pl.getSize(); i++) {
        pl.setPersonByIdx(i, Person(testNames[i], (rand() % 40) + 10));
    }
    // End Test Data

    cout << "UNSORTED LIST: " << endl;
    for (Person p : pl.getEasyList()) {
        cout << "  ";
        cout << "Name: " << setw(10) << left << p.getName();
        cout << "Age: " << p.getAge() << endl;
    }
    cout << endl;

    pl.sortListByName();

    // Another way of accessing members.
    cout << "SORTED LIST: "<< endl;
    for (int i = 0; i < pl.getSize(); i++) {
        cout << "  ";
        cout << "Name: " << setw(10) << left << pl.getPersonByIdx(i).getName();
        cout << "Age: " << pl.getPersonByIdx(i).getAge() << endl;
    }


    return 0;
}
```

// END of main.cpp.

PersonList.h

```
#ifndef HOMEWORK_CH10_QN3_PERSONLIST_H
#define HOMEWORK_CH10_QN3_PERSONLIST_H

#include <string>
#include <vector>
#include "Person.h"

class PersonList {

private:
    Person* people;
    int size = 0;

public:
    PersonList();
    PersonList(int);
    ~PersonList();

    int getSize();
    std::vector<Person> getEasyList();
    Person getPersonByIdx(int);

    void setSize(int);
    void setPersonByIdx(int, Person);

    void sortListByName();
};

#endif //HOMEWORK_CH10_QN3_PERSONLIST_H
```

// END of PersonList.h.

PersonList.cpp

```
//
// Created by TheLoneWoof on 10/20/17.
//

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

PersonList::PersonList() {
    size = 0;
    people = nullptr;
}

PersonList::PersonList(int listLength) {
    size = listLength;
    people = new Person[size];
}

PersonList::~PersonList() {
    delete [] people;
    people = nullptr;
    size = 0;
}

int PersonList::getSize() {
    return size;
}

vector<Person> PersonList::getEasyList() {
    vector<Person> l;
    for (int i = 0; i < size; i++) {
        l.push_back(people[i]);
    }
    return l;
}

Person PersonList::getPersonByIdx(int idx) {
    if (idx >= size || idx < 0) return Person();
    else return people[idx];
}

void PersonList::setSize(int listLength) {
    Person* temp_people = new Person[listLength];
    for (int i = 0; i < listLength && i < size; i++) {
        temp_people[i] = people[i];
    }
    delete [] people;
    people = temp_people;
    size = listLength;
    temp_people = nullptr;
}
```

// PersonList.cpp continued on next page.

PersonList.cpp

```cpp
void PersonList::setPersonByIdx(int idx, Person p) {
    if (idx < size && idx >= 0) {
        people[idx] = p;
    } else; // throw a fit.
}

void PersonList::sortListByName() {
    // Customized version of the Selection Sort
    Person** temp_PointerList;
    temp_PointerList = new Person*[size];
    for (int i = 0; i < size; i++) {
        temp_PointerList[i] = people + i;
    }

    Person* min_person;
    int min_index;

    for (int i = 0; i < size; i++) {
        min_index = i;
        min_person = temp_PointerList[i];
        for (int j = i + 1; j < size; j++) {
            if (temp_PointerList[j]->getUppercaseName() < min_person->getUppercaseName()) {
            // if ((*temp_PointerList[j]).name < (*min_person).name) {
                min_index = j;
                min_person = temp_PointerList[j];
            }
        }
        if (min_person != temp_PointerList[i]) {
            temp_PointerList[min_index] = temp_PointerList[i];
            temp_PointerList[i] = min_person;
        }
    }

    // Make a cache of the proper list of objects.
    // I can't directly assign temp_PointerList to people.
    // since they're referring to the same addresses.
    Person* temp_PersonList = new Person[size];
    for (int i = 0; i < size; i++) {
        temp_PersonList[i] = *(temp_PointerList[i]);
    }
    for (int i = 0; i < size; i++) {
        people[i] = temp_PersonList[i];
    }

    delete [] temp_PersonList;
    delete [] temp_PointerList;
    temp_PersonList = nullptr;
    temp_PointerList = nullptr;
}
```

// END of PersonList.cpp.