Name:- Chad Huntebrinker

Answer the following Questions:

**Q1: [4 points]** For the following T(n) find values of n0 and c such that cn^3 is larger than T(n) for all n larger than n0.

T(n) = n^3 – 5n^2 + 20n – 10

|  |
| --- |
| Answer the question using this space.  Cn^3 = n^3 – 5n^2 + 20n – 10  C = 1 – 5/n + 20/n^2 – 10/n^3  n0 = 1; c = 1 – 5/1 + 20/1^2 – 10/1^3 = 6  6n^3 >= n^3 – 5n^2 + 20n – 10 when n >=1 |

**Q2: [6 points]** For the C++ function shown below, find the T(n) and find values of n0 and c.

**int selectkth(int a[], int k, int n) {**

**int i, j, mini, tmp;**

**for (i = 0; i < k; i++) { -----> 1 + 6n**

**mini = i;**

**for (j = i+1; j < n; j++) -> (n(n-1))/2**

**if (a[j]<a[mini])**

**mini = j;**

**tmp = a[i];**

**a[i] = a[mini];**

**a[mini] = tmp;**

**}**

**return a[k-1];**

**}**

|  |
| --- |
| Answer the question using this space.  (1+6n)((n^2)/2 – (n/2))  3n^3 – 2.5n^2 – 0.5n  C = 3 – 2.5/n – 0.5/n^2  n0 = 2; 3 – 2.5/2 – 0.5/4 = 1.625  1.625n^3 >= 3n^3 – 2.5n^2 – 0.5n; when n >=2 |

**Q3: [9 points]** Develop an OOP program to maintain a list of homework assignments. When an assignment is assigned, add it to the list, and when it is completed, remove it. You should keep track of the due date. Your program should provide the following services:

• Add a new assignment.

• Remove an assignment.

• Provide a list of the assignments in the order they were assigned.

• Find the assignment(s) with the earliest due date.

create a class called assignments to complete the task and use STL to implement the linked list. compile and run your project and test all the methods.

|  |
| --- |
| Provide your code and the results of running the project using this space  //Chad Huntebrinker  //CS 303  #include <iostream>  #include <iomanip>  #include <list>  #include <iterator>  #include <string>  using namespace std;  //A class called assignments is created. It holds the year (example: 2021 or 1873), the month (example: 8 or 11),  //the day (example: 5 or 18), and the assignment name (example: CS 303 or MATH 300). It has an addAssignments function  //along with 4 functions that each return either the year, month, day, or assignmentName.  class assignments {  public:  void addAssignments(int userYear, int userMonth, int userDay, string name);  int getYear() { return year; }  int getMonth() { return month; }  int getDay() { return day; }  string getName() { return assignmentName; }  private:  int year;  int month;  int day;  string assignmentName;  };  //This function is from the assignment class and allows the user to add a homework assignment.  void assignments::addAssignments(int userYear, int userMonth, int userDay, string name) {  year = userYear;  month = userMonth;  day = userDay;  assignmentName = name;  }  //This function removes an assignment from the assignment list. If the user enters in a year, month, day, and  //assignment name that matches with one in the list, then that assignment is removed. Otherwise, if the data entered  //in doesn't match an assignment in the list, then an error message is outputted to the user.  list<assignments> removeAssignments(list<assignments> userList, int userYear, int userMonth, int userDay, string name) {  list<assignments>::iterator be = userList.begin();  list<assignments>::iterator end = userList.end();  assignments temp;  while (be != end) {  temp = \*be;  if (userYear == temp.getYear() && userMonth == temp.getMonth() &&  userDay == temp.getDay() && name == temp.getName()) {  userList.erase(be);  break;  }  else {  cout << "Could not find assignment because wrong information entered. " << endl << endl;  }  \*be++;  }  return userList;  }  //This function outputs the list in the order that the assignments were assigned. They are outputted in the following  //way: Month || Day || Year || Assignment Name.  void outputList(list<assignments> userList){  list<assignments>::iterator be = userList.begin();  list<assignments>::iterator end = userList.end();  assignments temp;  cout << "Month || Day || Year || Assignment Name" << endl;  cout << setw(39) << setfill('=') << "" << endl;  cout << setfill(' ') << "";  while (be != end) {  temp = \*be;  cout << setw(6) << temp.getMonth() << "||" << setw(5) << temp.getDay() << "||" << setw(6) << temp.getYear()  << "||" << temp.getName() << endl;  \*be++;  }  cout << endl << endl;  }  //This function finds the assignment with the earliest due date. The assignment variable earliest is assigned  //the first element in the assignment list. Then, the assignment variable temp is assigned whatever element  //the iterator variable be is at. There are 3 situations where earliest is assigned to the element temp is assigned to:  //1) If temp's year is < earliest's year.  //2) If temp's year == earliest's year and temp's month < earliest's month.  //3) If temp's year == earliest's year, temp's month == earliest's month and temp's day < earliest's day.  //If none of the situations occur, then the loop goes on to the next element. At the very end of the function,  //the assignment with the earliest's due date is outputted with it's information (year, month, etc.)  void findEarliestDate(list<assignments> userList) {  list<assignments>::iterator be = userList.begin();  list<assignments>::iterator end = userList.end();  assignments temp, earliest;  earliest = \*be;  while (be != end) {  temp = \*be;  if ((earliest.getYear() > temp.getYear()) || (earliest.getYear() == temp.getYear() &&  earliest.getMonth() > temp.getMonth()) || (earliest.getYear() == temp.getYear() &&  earliest.getMonth() == temp.getMonth() && earliest.getDay() > temp.getDay())) {  earliest = temp;  }  \*be++;  }  cout << "Month/Day/Year: " << earliest.getMonth() << '/' << earliest.getDay() << '/' << earliest.getYear() << endl;  cout << "Assignment Name: " << earliest.getName() << endl << endl << endl;  }  int main() {  list<assignments> myList;  assignments temp;  char userInput;  int userYear, userMonth, userDay;  string name;  do {  //The menu that the user sees and uses the enter and remove assignments.  cout << "Please enter what command you would like to initiate: " << endl;  cout << "To add an assignment to the list, enter A: " << endl;  cout << "To remove an assignemnt from the list, enter R: " << endl;  cout << "To see a list of the assignments in the order they were assigned, enter L: " << endl;  cout << "To see the assignment with the earliest due date, enter D: " << endl;  cout << "To quit, enter Q: " << endl;  cin >> userInput;  userInput = toupper(userInput);  cout << endl;  if (userInput == 'A' || userInput == 'R') {  cout << "Please enter the year the assignment is due (example: 2020): ";  cin >> userYear;  cout << "Please enter the month the assignment is due (01 - 12): ";  cin >> userMonth;  cout << "Please enter the day the assignment is due (01 - 31): ";  cin >> userDay;  cin.ignore();  cout << "Please enter what class this assignment is in: ";  getline(cin, name);  cout << endl << endl;  if (userInput == 'A') {  temp.addAssignments(userYear, userMonth, userDay, name);  myList.push\_back(temp);  }  else {  myList = removeAssignments(myList, userYear, userMonth, userDay, name);  }  }  else if (userInput == 'L') {  outputList(myList);  }  else if (userInput == 'D') {  findEarliestDate(myList);  }  } while (userInput != 'Q');  return 0;  } |

**Q4: [6 points]** For the double-linked list in the figure below, explain the effect of each statement in the following fragments. **MAKE SURE TO STATE THE FINALE RESULT OF EACH FRAGMENT.** Each fragment should be treated independently.

a. DNode\* node\_ptr = tail->prev;

node\_ptr->prev->next = tail;

tail->prev = node\_ptr->prev;

|  |
| --- |
| Answer Q4: a using this space.  Line 1: A new variable of the object DNode\* named node\_ptr is created and is now assigned the data where the tail’s prev pointer is pointing to.  Line 2: node\_ptr’s prev and next pointer is pointing to the tail.  Line 3: the tail’s prev pointer is pointing to where the node\_ptr’s prev pointer is pointing. So, the tail’s pointer is pointing to itself (it is pointing to the tail). |

b. DNode\* node\_ptr = head;

head = new DNode("Tamika");

head->next = node\_ptr;

node\_ptr->prev = head;

|  |
| --- |
| Answer Q4: b using this space.  Line 1: a new DNode (called node\_ptr) is created and it is assigned the head’s data.  Line 2: head has a new data assigned to it (the name “Tamika” is now the head’s data).  Line 3: head’s next pointer is pointing to the node\_ptr variable.  Line 4: node\_ptr’s prev pointer is now pointing to the head variable. |

c. DNode\* node\_ptr= new DNode(“Adam”);

tail->prev->next=node\_ptr;

node\_ptr->prev=tail->prev;

tail=node\_ptr;

|  |
| --- |
| Answer Q4: c using this space.  Line 1: a new DNode (called node\_ptr) is created and is assigned a new data string to hold (the name “Adam”).  Line 2: the tail’s prev and next pointer is now pointing to the node\_ptr variable.  Line 3: the node\_ptr’s prev pointer is pointing to where the tail’s prev pointer is pointing to (so, node\_ptr’s prev pointer is pointing to itself).  Line 4: the tail is assigned the data that node\_ptr is holding (so, the tail is now holding the name “Adam”). |

