CS 150 **Assignment 11** **Spring 2016**

Description:

In this assignment, you will be creating and maintaining a linked list (LL) of objects

All work must be submitted by the end of the lab period.

Hand in: Source Code. Remember to **include the necessary documentation in the source code.**
There is a link under the Assignments tab on Blackboard, for you to submit your work.
Name your documents correctly: Your Source Code and documents should include your first initial and last name. For example: Stewie Griffin would name his source code file – **SGriffin_Lab10.cpp**

Congratulations, you have just been hired by **the FIRM.** They want you to write a program that processes an input file containing information on their employees. The program creates a dynamic list of employees, and provides management with some tools that include:

- Printing the list of all employees
- Calculating the average number of employee dependents
- Termination an employee (remove employee from the list), based on a user specified ID number.

The input for the program contains information on the employees of the firm. There are sample versions of the input file provided for you to test your program solution.

Fortunately, most of the coding design has been completed and requires only your efforts at copying the source code found here to assemble the program. Once you have coded all of the parts detailed below, build and compile your program. You may need to correct any errors that you made in coding up your solution. Add the following code and function definitions to the source code template file. Make sure to include your name and lab CRN in the appropriate places. To receive full credit, you must **write a function for each of the following tasks, and your program MUST perform all the required functionality.**

Task 1: Define the following data structures in your source code, to accommodate our employee information and linked list of nodes. Create these exactly as you see here, beginning with the employee type struct, and then the typedef statement followed by the nodeType struct definition.

```
struct employeeType
{
        int idNum;
        string firstName;
        string lastName;
        char gender;
        double payrate;
        int dependents;
};
```

```
typedef  employeeType infoType ;
struct nodeType
{
     infoType info;
     nodeType  *link;
};
```

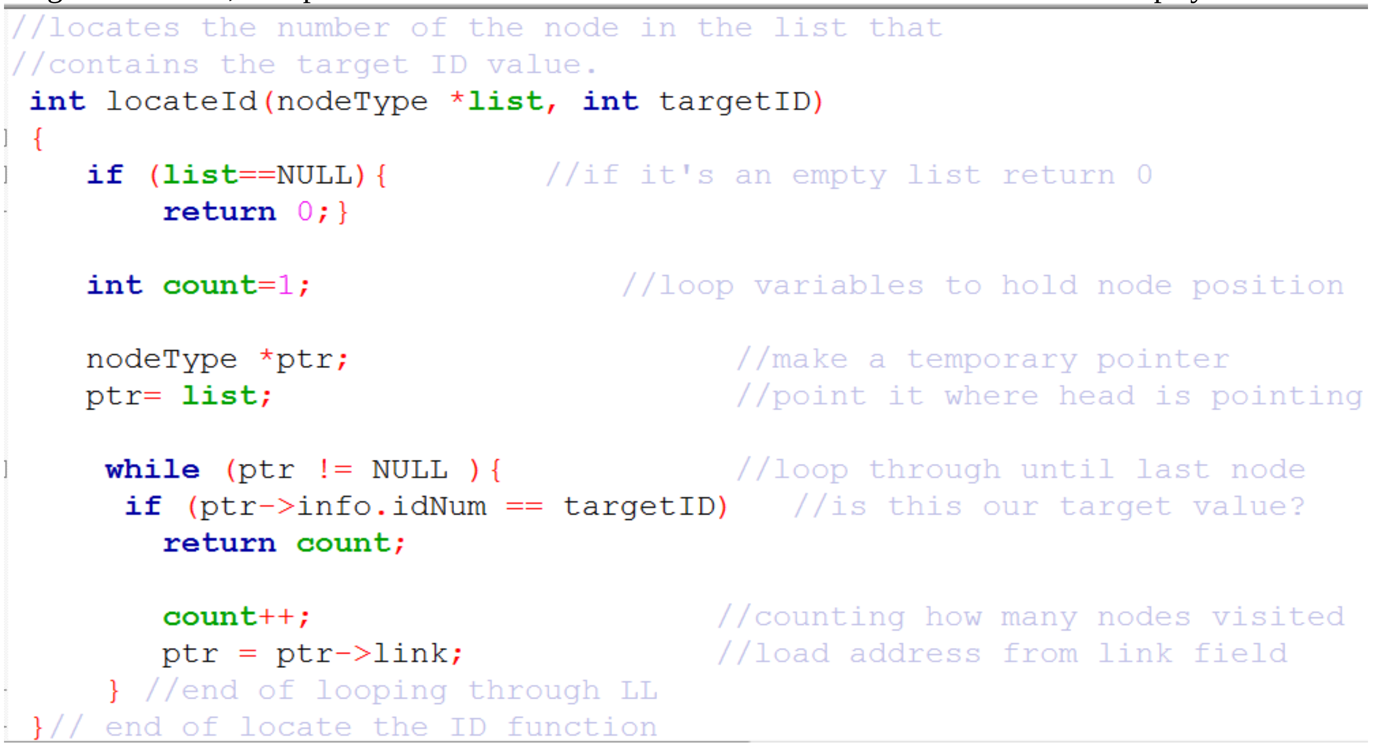
Task 2: Use the following code to fill in the definition of the print function. This function prints the contents of our LL of nodes, by traversing each node and calling the printOne function.

```
//print makes a temp pointer to head of LL, and moves node
//to node displaying the values of the members of each element
 void print(nodeType *first){
     nodeType *current;          //make a temporary pointer
     current = first;            //aim it at the first node in the list
    while (current != NULL)      //while not at end of list
     {
      printOne ( current->info) ;  //call the printOne function.
      cout <<"\n";
      current = current->link;     //move to the next node
      }
   }
```

Task 3: Write up the following function that calculates the average number of dependents that the Firms' employees are on record having. The function receives the head pointer to a list, and then traverses through the list counting the nodes. The average is calculated from the count and sum, and returned.

```
int avgDependents (nodeType *list)
{
   if (list==NULL){         //if it's an empty list return 0
       return 0;}

   int sum=0,avg=0,count=0;
   nodeType *ptr;                           //make a temporary pointer
   ptr= list;                               //point it where head is pointing

    while (ptr != NULL ){                   //loop through until last node
       sum = sum + ptr->info.dependents;    //sum up each nodes dependents
       count++;                             //count how many nodes visited
       ptr = ptr->link;                     //load the address in the current
    }                                       //  nodes link field
     //end of looping thru LL
    if (count != 0){
       avg = sum / count;           //divide nodes by dependents
       return avg ;                 //return average number of dependents
    }
}// end of average function
```

Task 4: Code the following function named locateId, whose parameters are the head pointer to a linked list of nodes, and the integer id number of the employee we want to find. The function returns the number of the node that the target id number is found in. We will use this function to locate the user defined target ID number, and pass this return value into deleteNode function to terminate the employee.



```
//locates the number of the node in the list that
//contains the target ID value.
 int locateId(nodeType *list, int targetID)
 {
    if (list==NULL){          //if it's an empty list return 0
        return 0;}

    int count=1;                      //loop variables to hold node position

    nodeType *ptr;                        //make a temporary pointer
    ptr= list;                            //point it where head is pointing

     while (ptr != NULL ){                //loop through until last node
      if (ptr->info.idNum == targetID)    //is this our target value?
        return count;

        count++;                         //counting how many nodes visited
        ptr = ptr->link;                 //load address from link field
     } //end of looping through LL
 }// end of locate the ID function
```

Task 5: This task involves two parts: coding up the two methods of creating a list, as shown below. There are two functions in this program that can be used to create the Linked List (LL) of employee nodes, either in a 'forwards' manner, or a 'backwards' manner – meaning that the list adds new nodes to the front(last one added is now the first on the list), or that the list adds new nodes to the back (last one is added to end of the list). **Task 5(a)** This function creates a linked list by adding a new last node after each call to the getOne function, inserting the nodes into the list in the same order as the input file - returns pointer to first node in list.

```
nodeType* linkdListCreate1 (ifstream& inFile){

   nodeType *first = NULL;   //will be our permanent head pointer
   nodeType *last = NULL;    //will point at the last node
   nodeType *temp = NULL;    //creates dynamic instances of node to add to list

   employeeType person;      //local instance to store input data & add to list

   person =getOne(inFile);   //the getOne function reads one input file line,
                             //& returns an employee object filled with data
    while (inFile)
    {
     temp = new nodeType;        //make a new node
     temp->info = person;        //store the employee data in it
     temp->link = NULL;          //null, because this will be new last node


        if(first == NULL){    //IF this is true, the list is initially empty
             first = temp;   //so point both first & last at the new node
             last = temp;
           }
         else {              //OTHERWISE, the list is not empty, so
             last->link = temp; //point last node link at new node
             last = temp;       //make last pointer point at this new last node.
           }

        person=getOne(inFile);  //get the next populated employee object
                                // so we can loop back up and put it in a new node

        }//end of while input loop
         return first;  //return the address of head of our new linked list
      }
```

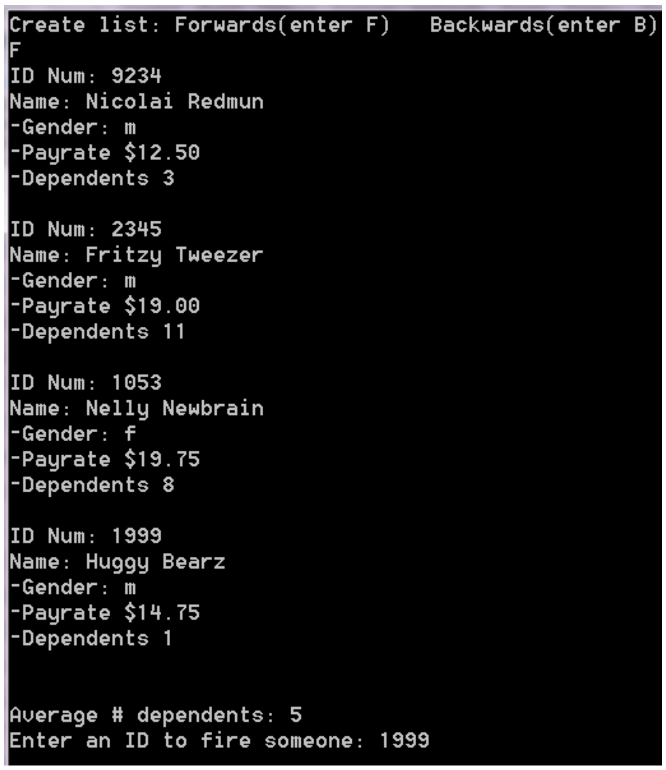
Task 5(b) This function creates a LL by setting the first pointer to the new node after each call to the getOne function, thus inserting nodes into list in reverse order. Returns pointer to first node in list.

```
nodeType* linkdListCreate2 (ifstream& inFile){

   // first declare & initialize some nodeType pointers to manage the list
   nodeType *first=NULL,   *temp=NULL;

   employeeType person;      //object to hold data, add to list
   person = getOne(inFile);  //getOne reads file data in, returns object
                             // that we next put into our list

while (inFile)      //while still data to read, loop and add new nodes
{
   temp = new nodeType;        //make a new node
   temp->info = person;        // copy our object into the new node
   temp->link = first;         // make it's link point to first node.
   first = temp;               // this becomes the new first node.
   person = getOne(inFile);    //get a new set of input data in our object
}
   return first;       //return the address of the first node.
}
```

Task : Complete the coding to make the program run as shown in the output below. The comments in the source code template will help you understand what you need to complete, and what is already completed for you. You will need to call the functions that you defined above to complete these tasks. In general, you will:

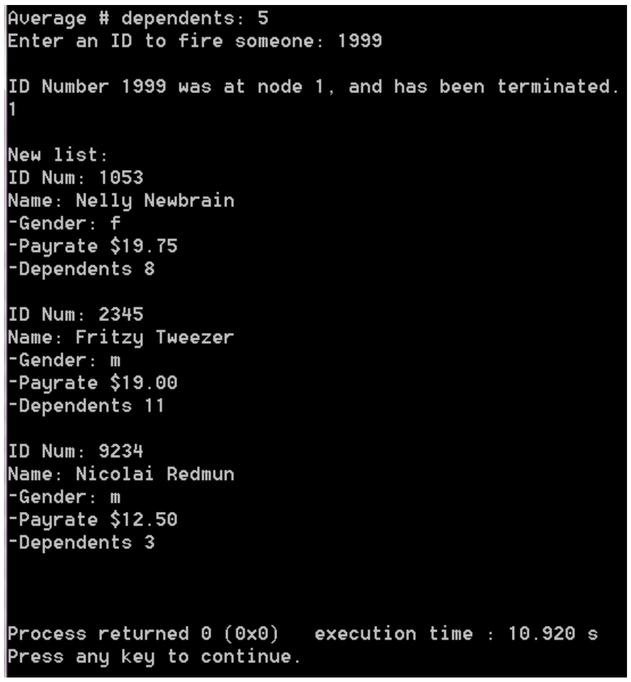
- Create a properly initialized nodeType pointer to maintain the list.
- Create the list either forwards or backwards, as the user determines.
- Print the list.
- Calculate and output the average number of employee dependents.
- Prompt the user for an ID number, and terminate that employee. Provide output as shown.
- Print the list.

Initial prompt for list creation, first print, and prompt for ID number to find.



```
Create list: Forwards(enter F)   Backwards(enter B)
F
ID Num: 9234
Name: Nicolai Redmun
-Gender: m
-Payrate $12.50
-Dependents 3

ID Num: 2345
Name: Fritzy Tweezer
-Gender: m
-Payrate $19.00
-Dependents 11

ID Num: 1053
Name: Nelly Newbrain
-Gender: f
-Payrate $19.75
-Dependents 8

ID Num: 1999
Name: Huggy Bearz
-Gender: m
-Payrate $14.75
-Dependents 1


Average # dependents: 5
Enter an ID to fire someone: 1999
```

This is the output after entering the target ID number and hitting the enter key.



```
Average # dependents: 5
Enter an ID to fire someone: 1999

ID Number 1999 was at node 1, and has been terminated.
1

New list:
ID Num: 1053
Name: Nelly Newbrain
-Gender: f
-Payrate $19.75
-Dependents 8

ID Num: 2345
Name: Fritzy Tweezer
-Gender: m
-Payrate $19.00
-Dependents 11

ID Num: 9234
Name: Nicolai Redmun
-Gender: m
-Payrate $12.50
-Dependents 3



Process returned 0 (0x0)   execution time : 10.920 s
Press any key to continue.
```