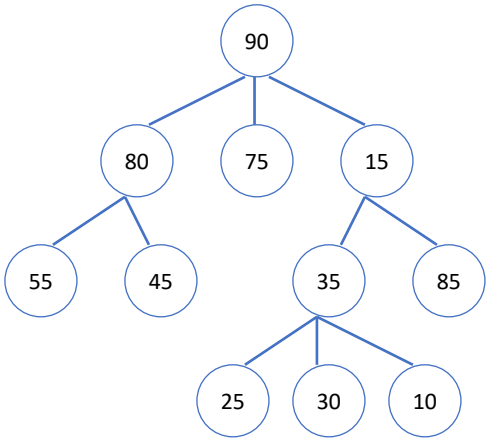
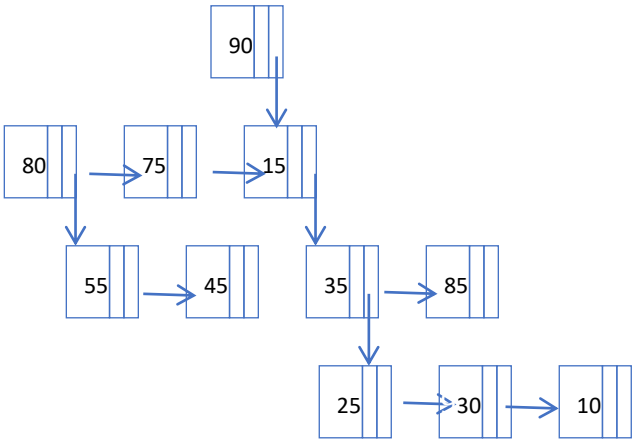
CS 2413 – Data Structures – Spring 2025 – Project Two
Due: 11:59 PM, February 22, 2025

Description: In this project, you will implement a generalized list data structure using an array. A generalized list can be used to represent a tree data structure. Consider the following tree and the corresponding generalized list.





The array representation of the above generalized list structure is given below:

	_Info	_Next	_Down
0	999	12	-1
1	75	9	-1
2	90	-1	4
3	30	13	-1
4	80	1	6
5	85	-1	-1
6	55	7	-1
7	45	-1	-1
8	999	0	-1
9	15	-1	10
10	35	5	11
11	25	3	-1
12	999	-1	-1
13	10	-1	-1

firstElement = 2; firstFree = 8;

Where firstElement is the first location in the array that contains the first element. In this case it is the root of the tree that has a value 90.

The free locations in the array are linked by the index positions in the array as a linked list. Follow the linked list starting at index position starting at location 8 and using the _Next index position. For **ease** I have put the value 999 in the _Info position where the index position is empty.

You will implement the following classes along with the methods mentioned each of the classes – You can add other helper methods as you deem useful:

```
template <class DT>
class GLRow; //class prototype
template <class DT>
ostream& operator <<(ostream& s, GLRow<DT>& oneGLRow);

class GLRow {
        friend ostream& operator<< <DT>(ostream& s, GLRow<DT>& oneGLRow);
        Protected:
                DT*     _Info;
                int     _Next;
                int     _Down;
        Public:
                GLRow ();
                GLRow (DT& newInfo);
                GLRow (GLRow<DT>& anotherOne);
                GLRow<DT>& operator= (GLRow<DT>& anotherOne); //Make sure you do
                        //a deep copy
                int getNext();
                int getDown();
                DT& getInfo();
                int setNext(int n);
                int setDown(int d);
                int setInfo (DT& x);
                ~GLRow(); //destructor
};
template <class DT>
class ArrayGLL; //class prototype
template <class DT>
ostream& operator <<(ostream& s, ArrayGLL<DT>& oneGLL);

class ArrayGLL {
        friend ostream& operator<< <DT>(ostream& s, ArrayGLL<DT>& OneGLL);
        Protected:
                GLRow<DT>* myGLL;
                int     maxSize; //Maximum size of the array of GLRows
                int     firstElement;
                int     firstFree;
        Public:
                ArrayGLL ();
                ArrayGLL (int size);
                ArrayGLL (ArrayGLL<DT>& anotherOne);
                ArrayGLL<DT>& operator= (ArrayGLL<DT>& anotherOne);
                void display (); //display in parenthesis format 10% BONUS
                int find (DT& key); //return the index position where you find
                        //the element key; -1 if not found; use recursive search
                void findDisplayPath (DT& Key); // as you travel through the tree
                        //print the values of nodes encountered. If the element is
                        //you will print the all the values
                int noFree (); //return the number of free locations; you need
                        //to follow the _Next and get the free locations
                int size (); //return the number of elements stored
                int parentPos(DT& Key); // provide the location of the parent of
                        //the element Key in the array 10% BONUS
                GLRow<DT>& operator [] (int pos); //return the GLRow that is in
                        //in the position pos in the array
                int getFirstFree();
                int getFirstElement();
                void setFirstFree (int pos);
                void setFirstElement (int pos);
                ~ArrayGLL (); //destructor
};
```

Your main program will have the following structure (changes may be required).

```
#include <iostream>
using namespace std;

int main() {
    ArrayGLL<int> firstGLL(20);
    int noElements, firstFree, firstElement;
    int value, next, down, parentPos;
    int pos = -1;
    int keyValue;

    // Read the number of elements
    cout << "Enter number of elements: ";
    cin >> noElements >> firstFree >> firstElement;


     // Input validation and initialization of GLRows
     for (int i = 0; i < noElements; i++) {
            cin >> value >> next >> down;

    // Dynamically create a new GLRow<int> for each entry
           GLRow<int> newRow(value);
           newRow.setNext(next);
           newRow.setDown(down);

           firstGLL[i] = newRow;  // Ensure deep copy is handled correctly
     }

     // Setting up the structure
     firstGLL.setFirstFree(firstFree);
     firstGLL.setFirstElement(firstElement);
     cout << "\n=== First GLL Structure ===\n";
     firstGLL.display();

    // Testing copy constructor
    ArrayGLL<int>* secondGLL = new ArrayGLL<int>(firstGLL);

    if (!secondGLL) {
        cerr << "Error: Memory allocation failed for secondGLL." << endl;
        return 1;
    }

    (*secondGLL)[1].setInfo(600);
    (*secondGLL)[2].setInfo(700);

    cout << "\n=== Second GLL (After Modifications) ===\n";
    (*secondGLL).display();

    // Testing find function
    keyValue = 700;
    pos = (*secondGLL).find(keyValue);
    if (pos != -1) {
        cout << "Element " << keyValue << " found at position: " << pos << endl;
        cout << "Element details: " << (*secondGLL)[pos] << endl;
        (*secondGLL).findDisplayPath(keyValue);
    } else {
        cout << "Element " << keyValue << " not found." << endl;
    }

    // Testing parentPos function
    parentPos = (*secondGLL).parentPos(keyValue);
    if (parentPos != -1) {
        cout << "Parent of " << keyValue << " is at position: " << parentPos << endl;
        cout << "Parent details: " << (*secondGLL)[parentPos] << endl;
    } else {
        cout << "Parent of " << keyValue << " not found." << endl;
    }

    // Testing size and free locations
    cout << "\nSize of secondGLL: " << (*secondGLL).size() << endl;
    cout << "Number of free locations: " << (*secondGLL).noFree() << endl;

    // Test getFirstElement and getFirstFree
    cout << "\nFirst element position: " << firstGLL.getFirstElement() << endl;
    cout << "First free position: " << firstGLL.getFirstFree() << endl;

    // Test setFirstElement and setFirstFree
```

```
    firstGLL.setFirstElement(2);
    firstGLL.setFirstFree(5);
    cout << "Updated first element position: " << firstGLL.getFirstElement() << endl;
    cout << "Updated first free position: " << firstGLL.getFirstFree() << endl;

    // Test GLRow class methods
    cout << "\n=== Testing GLRow Methods ===\n";
    GLRow<int> testRow(50);  // Creating a GLRow object
    cout << "Initial Row: " << testRow << endl;

    // Set and Get methods
    testRow.setNext(8);
    testRow.setDown(4);
    testRow.setInfo(75);
    cout << "Updated Row: " << testRow << endl;
    cout << "Next Pointer: " << testRow.getNext() << endl;
    cout << "Down Pointer: " << testRow.getDown() << endl;
    cout << "Info: " << testRow.getInfo() << endl;

    // Test copy constructor
    GLRow<int> copiedRow(testRow);
    cout << "Copied Row: " << copiedRow << endl;

    // Test assignment operator
    GLRow<int> assignedRow(0);
    assignedRow = copiedRow;
    cout << "Assigned Row: " << assignedRow << endl;

    // Ensure different memory locations
    if (&testRow != &copiedRow && &copiedRow != &assignedRow) {
        cout << "Deep copy successful: Objects have separate memory allocations." << endl;
    } else {
        cout << "Error: Deep copy failed!" << endl;
    }

    // Proper memory cleanup
    delete secondGLL;
    secondGLL = nullptr;

    cout << "\n=== All Tests Completed Successfully! ===\n";
    return 0;
}
```

Sample Input

14 2 8
999 12 -1
75 9 -1
90 -1 4
30 13 -1
80 1 6
85 -1 -1
55 7 -1
45 -1 -1
999 0 -1
15 -1 10
35 5 11
25 3 -1
999 -1 -1
10 -1 -1

Redirected Input: Redirected input provides you a way to send a file to the standard input of a program without typing it using the keyboard. To use redirected input in Visual Studio environment, follow these steps: After you have opened or created a new project, on the menu go to project, project properties, expand configuration properties until you see Debugging, on the right you will see a set of options, and in the command arguments type "< **input filename**". The < sign is for redirected input and the **input filename** is the name of the input file (including the path if not in the working directory). A simple program that reads a matrix can be found below.

```
#include <iostream>

using namespace std;

int main () {

      int r,c,cv,nsv;
      int val;

      cin >> r >> c >> cv >> nsv;
      for (int i=0; i < r; i++) {
         for (int j=0; j < c; j++) {
            cin >> value;
            cout << value << " ";
         }
         endl;
      }
      return 0;
}
```

Constraints

1. In this project, the only header you will use is #include <iostream> and using namespace std.
2. None of the projects is a group project. Consulting with other members of this class our seeking coding solutions from other sources including the web on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.

Project Submission Requirements: Generalized List Implementation

1. Code Development (75%)

Implement the provided **GLRow** and **ArrayGLL** class structures to construct and manipulate a **Generalized List** (GLL) using an array-based approach. Your implementation must be fully compatible with the **main program** provided and should correctly manage tree-like hierarchical data using _Info, _Next, and _Down pointers. Specifically, your code should include methods for:

- **Inserting elements** into the Generalized List.
- **Finding an element** and returning its position.
- **Finding and displaying the path** to a given element.
- **Determining the parent of an element** in the structure.
- **Retrieving the number of free locations** in the array.
- **Copying and assigning Generalized Lists**, ensuring **deep copies**.
- **Displaying the Generalized List** in a structured format.

Your code **must** successfully run with the provided **main program** and corresponding input, demonstrating the correct functionality of the GLRow and ArrayGLL classes. **All methods must integrate seamlessly** with the main program.

AI/LLM Tool Usage:

- You **may use** Large Language Models (LLMs) or AI tools, such as **GitHub Copilot**, to assist in writing and refining your classes and methods.
- If you **did not** use AI tools, you **must still document** how you would have used them to approach this project (see **Section 2**).
- **Failure to follow the provided class structure** will result in **zero points** for this project.

2. LLM and GitHub Copilot Usage Documentation (15%)

If you choose to use **LLMs or AI tools**, you must **document your usage** in a **PDF report** that includes:

- **Prompts and Suggestions:**
  - List the specific prompts or suggestions you used, such as:
    - *"Generate a method to find an element in a tree-like structure stored in an array."*
    - *"How can I implement deep copy for a class that contains dynamic memory in C++?"*
- **Rationale:**
  - Explain **why** you used each AI-generated suggestion and how it helped your implementation.
  - Example:
    - *"I used AI to help design a deep copy constructor for GLRow, ensuring that dynamically allocated memory was properly duplicated, preventing shallow copies."*
- **Incremental Development:**
  - Describe **how** you used AI tools in an **incremental manner**, refining your classes and methods step-by-step.
  - Example:
    - *"Initially, I generated the find() method for ArrayGLL. Then, I iteratively improved its recursion to traverse both _Next and _Down pointers correctly."*

Even if you **did not use AI**, your report should explain how **you would have used** these tools to complete the project.

3. Debugging and Testing Plan (10%)

Submit a **detailed debugging and testing plan** in **a separate document (PDF or text file)** that includes:

- **Specific Tests:**
  - Explain the **test cases** used to verify correctness, such as:
    - **Checking insertion order**: Ensuring elements are placed correctly in the array.
    - **Verifying find()**: Confirming correct index positions are returned.
    - **Testing findDisplayPath()**: Ensuring the correct path is printed.
    - **Checking deep copy operations**: Validating that changes to a copied list don't affect the original.
- **Issues and Resolutions:**
  - Document **problems you encountered** and how you **fixed them**.
  - Example:
    - *"Initially, find() returned incorrect results because I wasn't properly checking _Next and _Down. To fix this, I ensured recursive traversal through both."*
- **Verification:**
  - Explain **how you ensured your classes work correctly**:
    - Running test cases in the **provided main program**.
    - Creating **additional test cases** for **edge cases** (e.g., searching for non-existent elements, checking an empty list).

Example Submission Structure

1. Code Implementation:

Submit your .cpp file containing:

- **Implementation of the GLRow and ArrayGLL classes.**
- **Main program** for reading input, executing methods, and testing.

2. LLM Documentation (PDF):

Include a report with:

- AI-generated **prompts & responses**.
- **How you used/refined** AI suggestions.
- **Lessons learned** (even if you didn't use AI).

3. Testing Plan:

Provide a separate PDF or text file with:

- **Planned test cases** and expected outputs.
- **Bug fixes and debugging strategies.**