BICOL UNIVERSITY, POLANGUI
POLANGUI, ALBAY
AY : 2024 - 2025

MEMBERS:
PESINO, MARK SELWYN G.
SAYSON, KYLA ISABELLE O.
SECRETO, JAN BENNY S.

Project Overview

This project demonstrates the implementation of a **Heap Builder** program in C++. It supports building and maintaining **Min-Heaps** and **Max-Heaps**. A heap is a specialized tree-based data structure that satisfies the **heap property**:

- **Min-Heap**: The smallest element is at the root, and every parent node is smaller than its children.
- **Max-Heap**: The largest element is at the root, and every parent node is larger than its children.

Features of our code

1. **Heap Creation**: Users can create a heap from a list of numbers and choose between Min-Heap and Max-Heap configurations.
2. **Heap Insertion**: Users can insert new numbers into the heap while maintaining the heap property.
3. **Dynamic Conversion**: The heap can be converted between Min-Heap and Max-Heap at any time.

HOW TO RUN THE CODE

1. CHOOSE HOW MANY NUMBERS YOU WANT TO ADD IN THE HEAP
2. YOU WILL CHOOSE IF YOU WANT TO BUILD MIN-HEAP, MAX-HEAP, INSERT NUMBER OR EXIT THE PROGRAM
3. AFTER CHOOSING IT WILL EXECUTE WHAT YOU CHOOSE FROM THE PROGRAM.

FUNCTIONALITY OF THE CODE

```
// Function to display the heap
void displayHeap(const vector<int>& heap) {
    for (int num : heap) {
        cout << num << " ";
    }
    cout << endl;
}
```

The displayHeap function is used to show all the numbers in a heap (a list of numbers) by printing them to the screen. It takes the heap as input, which is a collection of numbers, and loops through each number in the list. For every number, it prints it followed by a space to keep the output organized. Once all the numbers are printed, it moves to the next line to ensure the output doesn't overlap with other text. For example, if the heap contains {5, 10, 15, 20}, calling this function will display 5 10 15 20 on the screen. This function is helpful because it gives you a clear view of the current contents of the heap at any point in your program.

```
// Function to maintain the heap property (Min-Heap or Max-Heap)
void heapify(vector<int>& heap, int n, int i, bool isMinHeap) {
    int chosen = i;  // Initialize as root
    int left = 2 * i + 1;  // Left child index
    int right = 2 * i + 2;  // Right child index

    if (left < n && ((isMinHeap && heap[left] < heap[chosen]) || (!isMinHeap && heap[left] > heap[chosen]))) {
        chosen = left;
    }

    if (right < n && ((isMinHeap && heap[right] < heap[chosen]) || (!isMinHeap && heap[right] > heap[chosen]))) {
        chosen = right;
    }

    if (chosen != i) {
        swap(heap[i], heap[chosen]);
        heapify(heap, n, chosen, isMinHeap);
    }
}
```

The heapify function ensures that a specific part of a heap satisfies the heap property, either as a Min-Heap (smallest number at the top) or a Max-Heap (largest number at the top). It works by comparing the current element (starting at index i) with its left and right child nodes. If one of the children is smaller (for a Min-Heap) or larger (for a Max-Heap) than the current element, the function swaps the current element with the appropriate child. After the swap, it recursively applies the same process to the affected child to fix any further violations. This function is important for maintaining the heap structure when building or modifying a heap.

```
void insertNumber(vector<int>& heap, int number, bool isMinHeap) {
    heap.push_back(number);
    int i = heap.size() - 1;
    while (i > 0) {
        int parent = (i - 1) / 2;
        if ((isMinHeap && heap[parent] > heap[i]) || (!isMinHeap && heap[parent] < heap[i])) {
            swap(heap[i], heap[parent]);
            i = parent;
        } else {
            break;
        }
    }
}
```

The insertNumber function adds a new number to the heap while maintaining its structure as either a Min-Heap or a Max-Heap. It starts by placing the new number at the end of the heap. Then, it compares the newly added number with its parent node. If the heap property is violated (e.g., the parent's value is greater for a Min-Heap or smaller for a Max-Heap), the function swaps the new number with its parent. This process continues upward until the number is in the correct position or reaches the root. This ensures that the heap remains valid after the insertion.

```
int main() {
    vector<int> heap;
    int n, number, option;
    bool isMinHeap = true;

    cout << "** Heap Builder **\n";
    cout << "How many numbers to add? ";
    cin >> n;

    cout << "Enter the numbers:\n";
    for (int i = 0; i < n; ++i) {
        cin >> number;
        heap.push_back(number);
    }
```

The main function begins by initializing a vector called heap to store the numbers, along with some variables to track user input. It starts by asking the user how many numbers they want to add to the heap. After the user specifies the count (n), it prompts them to enter the numbers one by one. Each entered number is added to the heap vector using the push_back function. This setup lays the foundation for building a Min-Heap or Max-Heap with the input numbers. The program also

prepares for additional operations, such as switching between heap types or inserting more numbers, based on user choices later in the menu.

```
    for (int i = (heap.size() / 2) - 1; i >= 0; --i) {
        heapify(heap, heap.size(), i, isMinHeap);
    }

    do {
        cout << "\nMenu:\n1. Build Min-Heap\n2. Build Max-Heap\n3. Insert Number\n4. Exit\nChoice: ";
        cin >> option;

        switch (option) {
            case 1:
                isMinHeap = true;
                for (int i = (heap.size() / 2) - 1; i >= 0; --i) {
                    heapify(heap, heap.size(), i, isMinHeap);
                }
                cout << "Min-Heap: ";
                displayHeap(heap);
                break;

            case 2:
                isMinHeap = false;
                for (int i = (heap.size() / 2) - 1; i >= 0; --i) {
                    heapify(heap, heap.size(), i, isMinHeap);
                }
                cout << "Max-Heap: ";
                displayHeap(heap);
                break;
```

```
            case 3:
                cout << "Enter number to insert: ";
                cin >> number;
                insertNumber(heap, number, isMinHeap);
                cout << "Number inserted.\n";
                break;

            case 4:
                cout << "Exiting program. Goodbye!\n";
                return 0;

            default:
                cout << "Invalid choice. Please try again.\n";
        }
    } while (true);

    return 0;
}
```

This part of the code performs two main functions: building a heap and presenting a menu for user actions. Initially, a loop starts from the middle of the heap array and moves toward the root, calling the heapify function to organize the elements into a heap (either Min-Heap or Max-Heap). The heapify function ensures the correct heap structure based on the selected type.

After the heap is built, the program enters a menu loop where the user can choose from four options. If the user selects "Build Min-Heap" or "Build Max-Heap," the heap is restructured using the heapify function with the appropriate settings. The program displays the heap after the transformation. If the user chooses to insert a new number,

they are prompted to input the value, which is added to the heap while maintaining the correct structure. Finally, the "Exit" option ends the program, while invalid inputs prompt an error message. This design allows dynamic interaction with the heap and accommodates various operations based on the user's preferences.