Southern Luzon State University
COLLEGE OF ENGINEERING
Computer Engineering Department
Lucban, Quezon

CPE 13 – Compilation Project

Submitted by:
PADUADA, Dave Jhared G.

Engr. Dean Adrian Nombrefia
INSTRUCTOR

Southern Luzon State University
COLLEGE OF ENGINEERING
Computer Engineering Department
Lucban, Quezon

CPE13 Data Structure and Algorithms

Compilation Project

Name: Dave Jhared G. Paduada **Date:** 1 – 17 - 25

Section: IF

```
#include <iostream>
#include <string> // For activity 2
#include <sstream> // For activity 2
#include <cmath> // For activity 4,Include cmath for math functions
using namespace std;

// Activity 1 Function
void activity1() {
    // Activity #1
    // September 18, 2024
    // Creating a program that computes for the sum, difference, product, and
quotient of 2 integers

    cout << "Activity 1: Introduction to C++ Programming\n\n";
    int first_Num;
    int sec_Num;

    cout << "--- Basic Calculator ---\n";

    // Input
    cout << "Enter the 1st number: ";
    cin >> first_Num;

    cout << "Enter the 2nd number: ";
    cin >> sec_Num;

    // Process & Output
    cout << "\nHere is the computed answers: \n";
    cout << " Sum: " << first_Num + sec_Num << "\n";
    cout << " Difference: " << first_Num - sec_Num << "\n";
    cout << " Product: " << first_Num * sec_Num << "\n";
    cout << " Quotient: " << first_Num/sec_Num;
}

// Activity 2 Function
void activity2() {
    cout << "Activity 2: Introduction to C++ Programming (Week 2)\n\n";
    // Activity #2
    // September 20, 2024
    //Create a multiplication table of 10 col and 10 rows

    // Input
    stringstream row; // Create a stringstream object to store the table

    int sixT = 16;
    int sevenT = 17;
    int eightT = 18;
    int nineT = 19;
    int twT = 20;
    int twOne = 21;
    int twTwo = 22;
    int twThree = 23;
    int twFour = 24;
    int twFive = 25;

    // Process
    row << sixT             << "   |  " << sevenT                << " | " <<
eightT              << "  | " << nineT          << "  | " <<
twT                << "  |  " << twOne         << " |  " << twTwo          << "
|  " << twThree                << " |  " << twFour             << " | " << twFive
<< "\n";
    row << sixT * sixT       << " | " << sixT * sevenT        << " | " << sixT *
eightT       << " | " << sixT * nineT    << " | " << sixT * twT       << " | " <<
sixT * twOne    << " | " << sixT * twTwo    << " | " << sixT * twThree       << "
| " << sixT * twFour        << " | " << sixT * twFive << "\n";
    row << sevenT * sixT    << " | " << sevenT * sevenT      << " | " << sevenT *
eightT     << " | " << sevenT * nineT   << " | " << sevenT * twT     << " | " <<
sevenT * twOne   << " | " << sevenT * twTwo   << " | " << sevenT * twThree     << "
| " << sevenT * twFour      << " | " << sevenT * twFive << "\n";
    row << eightT * sixT     << " | " << eightT * sevenT      << " | " << eightT *
eightT     << " | " << eightT * nineT   << " | " << eightT * twT     << " | " <<
eightT * twOne   << " | " << eightT * twTwo   << " | " << eightT * twThree     << "
| " << eightT * twFour      << " | " << eightT * twFive << "\n";
    row << nineT * sixT      << " | " << nineT * sevenT       << " | " << nineT *
eightT      << " | " << nineT * nineT    << " | " << nineT * twT      << " | " <<
nineT * twOne    << " | " << nineT * twTwo    << " | " << nineT * twThree      << "
| " << nineT * twFour       << " | " << nineT * twFive << "\n";
    row << twT * sixT        << " | " << twT * sevenT         << " | " << twT *
eightT        << " | " << twT * nineT      << " | " << twT * twT        << " | " <<
twT * twOne      << " | " << twT * twTwo      << " | " << twT * twThree        << "
| " << twT * twFour         << " | " << twT * twFive << "\n";
    row << twOne * sixT      << " | " << twOne * sevenT       << " | " << twOne *
eightT      << " | " << twOne * nineT    << " | " << twOne * twT      << " | " <<
twOne * twOne    << " | " << twOne * twTwo    << " | " << twOne * twThree      << "
| " << twOne * twFour       << " | " << twOne * twFive << "\n";
    row << twTwo * sixT      << " | " << twTwo * sevenT       << " | " << twTwo *
eightT      << " | " << twTwo * nineT    << " | " << twTwo * twT      << " | " <<
twTwo * twOne    << " | " << twTwo * twTwo    << " | " << twTwo * twThree      << "
| " << twTwo * twFour       << " | " << twTwo * twFive << "\n";
    row << twThree * sixT    << " | " << twThree * sevenT     << " | " << twThree *
eightT    << " | " << twThree * nineT << " | " << twThree * twT    << " | " <<
twThree * twOne << " | " << twThree * twTwo << " | " << twThree * twThree    << "
| " << twThree * twFour     << " | " << twThree * twFive << "\n";
    row << twFour * sixT     << " | " << twFour * sevenT      << " | " << twFour *
eightT     << " | " << twFour * nineT   << " | " << twFour * twT     << " | " <<
twFour * twOne   << " | " << twFour * twTwo   << " | " << twFour * twThree     << "
| " << twFour * twFour      << " | " << twFour * twFive << "\n";
```

```
    row << twFive * sixT    << " | " << twFive * sevenT     << " | " << twFive *
eightT     << " | " << twFive * nineT  << " | " << twFive * twT    << " | " <<
twFive * twOne  << " | " << twFive * twTwo  << " | " << twFive * twThree    << "
| " << twFive * twFour     << " | " << twFive * twFive << "\n";
    string multiplication_Table = row.str();

    // Output
    cout << "    --- Multiplication Table for 16 to 25 ---\n\n";
    cout << multiplication_Table; // Convert stringstream to string
}

// Activity 3 Function
void activity3() {
    cout << "Activity 3: Control Statements\n\n";
    // September 25, 2024
    // B. Prints a square pattern using numbers in descending order.

    int n;
    cout << "Enter the size of the square (minimum 7): ";
    cin >> n;

    if (n < 7) {
        cout << "Please enter a number greater than or equal to 7." << endl;
        return;
    }

    int matrix[n][n];

    // Fill the matrix with numbers in descending order
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            matrix[i][j] = (n - i) * (n - j);
        }
    }

    // Print the matrix
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cout << matrix[i][j] << "\t";
        }
        cout << endl;
    }
}

    // Activity 4 Function
    const double pi = 3.14; // Constant value of p

    // Function to calculate the area and circumference of a circle
    void circleProperties(double radius) {
        double area = pi * pow(radius, 2);
        double perimeter = 2 * pi * radius;
        cout << "Area of the Circle: " << area << endl;
        cout << "Perimeter (Circumference) of the Circle: " << perimeter << endl;
    }
    // Function to calculate the area and circumference of an ellipse
    void ellipseProperties(double majorAxis, double minorAxis) {
        if (majorAxis <= 0 || minorAxis <= 0) {
            cout << "Invalid input for ellipse axes." << endl;
            return;
        }
        double area = pi * majorAxis * minorAxis;
        double perimeter = pi * (3 * (majorAxis + minorAxis) - sqrt((3 *
majorAxis + minorAxis) * (majorAxis + 3 * minorAxis)));
        cout << "Area of the Ellipse: " << area << endl;
        cout << "Perimeter (Circumference) of the Ellipse: " << perimeter <<
endl;
    }

void activity4() {
    cout << "Activity 4: Pointers, References, and Dynamic Memory
Allocation\n\n";
    // October 4, 2024
    //Accepts input for computing the area and circumference of circle and
ellipse.
    // Use cont function for pi and use RADIUS of circle to ellipse and vice
versa.

    int choice;
    double radius, majorAxis, minorAxis;

    cout << "Choose type of shape:" << endl;
    cout << "1. Circle" << endl;
    cout << "2. Ellipse" << endl;
    cin >> choice;

    if (choice == 1) {
        cout << "Enter the radius of the circle: ";
        cin >> radius;
        circleProperties(radius);
    } else if (choice == 2) {
        cout << "Enter the major axis of the ellipse: ";
        cin >> majorAxis;
        minorAxis = majorAxis / 2.0; // Assuming minor axis is half of the major
axis for this example
        ellipseProperties(majorAxis, minorAxis);
    } else {
        cout << "Invalid choice." << endl;
    }
}

    // Activity 5 Function
    int inputAndOutputArray(int arr[][500], int rows, int columns) {
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < columns; ++j) {
                cout << "Enter element " << i + 1 << ", " << j + 1 << ": ";
                cin >> arr[i][j];
            }
        }
        // Displaying the inputted arrays
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < columns; ++j) {
                if (arr[i][j] >= 100) {
                    cout << arr[i][j] << " ";
                } else if (arr[i][j] >= 10) {
                    cout << " " << arr[i][j] << " ";
                } else {
                    cout << "  " << arr[i][j] << " ";
                }
            }
            cout << endl;
        }

        int sum = 0;
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < columns; ++j) {
                sum += arr[i][j];
            }
        }

        return sum;
    }
void activity5() {
    cout << "Activity 5: Array and Multi-Dimensional Array\n\n";
    // October 9, 2024
    // Asks the user the number of row and column of 2-Dimensional array, put 
values in the array, prints the content of the array,
    // and dtermines the sum of all integers.

    int rows, columns;
    cout << "Enter the number of rows: ";
    cin >> rows;
    cout << "Enter the number of columns: ";
    cin >> columns;

    int arr[500][500];
    int sum = inputAndOutputArray(arr, rows, columns);
    cout << "The sum of all elements in the array is: " << sum << endl;
}


// Activity 6 Function
void activity6() {
    // October 30, 2024
    // Search Algorithm
    // Create a 4x4 that asks the user the number of row and column of a 2-
dimensional array,
    // initialize value in the array, prints the content of the array, and search 
for inputted number using
    // linear or sequential search. Must input 3-digit integer

    cout << "Activity 6: Search Algorithm \n\n";
    // fixed 4x4 matrix w/ unique 3-digit values
    int matrix[4][4] = {
        {889, 644, 198, 344},
        {940, 329, 525, 701},
        {777, 435, 542, 796},
        {628, 261, 445, 840}
    };

    // Display matrix content
    cout << "--- 4x4 Matrix Finder ---\n";
    for (int row = 0; row < 4; row++) {
        for (int col = 0; col < 4; col++) {
            cout << matrix[row][col] << " ";
        }
        cout << endl;
    }

    int searchValue;
    bool found = false;

    // Search for a number in the matrix
    while (!found) {
        cout << "\nEnter a 3-digit number to search for: ";
        cin >> searchValue;

        // Linear search
        for (int row = 0; row < 4; row++) {
            for (int col = 0; col < 4; col++) {
                if (matrix[row][col] == searchValue) {
                    cout << "Number " << searchValue << " found at index (" <<
row << ", " << col << ").\n";
                    found = true;
                    break;
                }
            }
            if (found) break;
        }

        // If not found, prompt the user to try again
        if (!found) {
            cout << "Number not found. Try again.\n";
        }
    }
}

    // Activity 7 Function
    // Function to perform Selection Sort
    void selectionSort(int array[], int size) {
        // Loop to iterate through each element except the last
        for (int i = 0; i < size - 1; i++) {
            int minIndex = i; // Assume the current index is the minimum
            for (int j = i + 1; j < size; j++) {
                if (array[j] < array[minIndex]) { // Find the minimum element
                    minIndex = j; // Update the minimum index
                }
            }
            // Swap the found minimum element with the first element
            if (minIndex != i) {
                int temp = array[i];
                array[i] = array[minIndex];
                array[minIndex] = temp;
            }
        }
    }

    // Function to perform Bubble Sort
    void bubbleSort(int array[], int size) {
        // Loop to control the number of passes
        for (int i = 0; i < size - 1; i++) {
            // Loop to compare adjacent elements
            for (int j = 0; j < size - i - 1; j++) {
                if (array[j] > array[j + 1]) { // Swap if elements are out of
order
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
    }

    // Function to perform Insertion Sort
    void insertionSort(int array[], int size) {
        // Loop through each element starting from the second
        for (int i = 1; i < size; i++) {
            int currentElement = array[i];
            int j = i - 1;
            // Move elements that are greater than currentElement one position
ahead
            while (j >= 0 && array[j] > currentElement) {
                array[j + 1] = array[j];
                j--;
            }
            array[j + 1] = currentElement; // Place currentElement in the correct
position
        }
    }
void activity7() {
    //
    // November 6, 2024
    // Reformatting codes

    cout << "Activity 7: Sorting Algorithm\n\n";
    int arraySize;
    int array[50]; // Array to store up to 50 elements

    // Input section
    cout << "Enter the size of the array (max 50): ";
    cin >> arraySize;

    // Ensure the size is within the valid range
    if (arraySize > 50 || arraySize <= 0) {
        cout << "Invalid size! Please enter a size between 1 and 50." << endl;
        return;
    }

    cout << "Enter " << arraySize << " elements: \n";
    for (int i = 0; i < arraySize; i++) {
        cin >> array[i]; // Read each element
    }

    // Perform Selection Sort and display the sorted array
    int selectionSortedArray[50];
    copy(array, array + arraySize, selectionSortedArray);
    selectionSort(selectionSortedArray, arraySize);

    cout << "\nArray sorted using Selection Sort: \n";

    for (int i = 0; i < arraySize; i++) {
        cout << selectionSortedArray[i] << " ";
    }
    cout << endl;

    // Perform Bubble Sort and display the sorted array
    int bubbleSortedArray[50];
    copy(array, array + arraySize, bubbleSortedArray);
    bubbleSort(bubbleSortedArray, arraySize);

    cout << "\nArray sorted using Bubble Sort: \n";

    for (int i = 0; i < arraySize; i++) {
        cout << bubbleSortedArray[i] << " ";
    }
    cout << endl;

    // Perform Insertion Sort and display the sorted array
    int insertionSortedArray[50];
    copy(array, array + arraySize, insertionSortedArray);
    insertionSort(insertionSortedArray, arraySize);

    cout << "\nArray sorted using Insertion Sort: \n";

    for (int i = 0; i < arraySize; i++) {
        cout << insertionSortedArray[i] << " ";
    }
    cout << endl;
}

    // Activity 8 Other operations
    const int ROWS = 4;
    const int COLS = 3;
    const int MAX_SIZE = ROWS * COLS; // 12
    const int MAX_ARRAYS = 10;        // Can store up to 10 arrays

    // Function to print a single array in a 4x3 forma
    void print2DArray(const int array[], int current_size, int arrayNumber) {
        cout << "Array " << arrayNumber << " (Active Elements = " << current_size
<< "):" << endl;
        for (int i = 0; i < ROWS; i++) {
            for (int j = 0; j < COLS; j++) {
                int index = i * COLS + j;
                if (index < current_size) {
                    cout << array[index] << " ";
                } else {
                    cout << "- ";
                }
            }
            cout << endl;
        }
        cout << endl;
    }
void activity8() {
    cout << "Activity 8: Other operations\n\n";
    /*Create a program that prints a declared 4x3 bidimensional array and a meny
for inserting, deleting, and copying of array.
    Ask the user for the operation, perform the intended operation, and prints
the result. (you need to copy the whole array to another array. */

    static int arrays[MAX_ARRAYS][MAX_SIZE];
    static int array_sizes[MAX_ARRAYS];

    int array_count = 1;
    int currentArrayIndex = 0; // Start with Array 1 as the active array

    // Initialize Array 1 with two-digit integers: 55 to 66
    for (int i = 0; i < MAX_SIZE; i++) {
        arrays[0][i] = 55 + i;
    }
    array_sizes[0] = MAX_SIZE; // Fully occupied

    cout << "Original Array (Array 1):" << endl;
    print2DArray(arrays[0], array_sizes[0], 1);

    bool exitProgram = false;
    while (!exitProgram) {
        cout << "=== MENU ===" << endl;
        cout << "1. Insert an element" << endl;
        cout << "2. Delete an element" << endl;
        cout << "3. Copy the array" << endl;
        cout << "4. Exit" << endl;

        cout << "\n\nCurrently operating on Array " << (currentArrayIndex + 1) <<
endl;

        int choice;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                // INSERT into the current active array
                if (array_sizes[currentArrayIndex] == MAX_SIZE) {
                    cout << "\n--Current array is full. Cannot insert a new
element.--" << endl;
                } else {
                    int r, c, num;
                    cout << "Enter the value to insert (two-digit integer): ";
                    cin >> num;

                    cout << "Enter the row (0 to " << ROWS - 1 << "): ";
                    cin >> r;
                    cout << "Enter the column (0 to " << COLS - 1 << "): ";
                    cin >> c;

                    if (r < 0 || r >= ROWS || c < 0 || c >= COLS) {
                        cout << "\n--Invalid row or column!--" << endl;
                        break;
                    }

                    int pos = r * COLS + c;

                    if (pos < 0 || pos > array_sizes[currentArrayIndex]) {
                        cout << "\n--Invalid position!--" << endl;
                        break;
                    }

                    // Shift right
                    for (int i = array_sizes[currentArrayIndex] - 1; i >= pos; i-
-) {
                        arrays[currentArrayIndex][i + 1] =
arrays[currentArrayIndex][i];
                    }

                    arrays[currentArrayIndex][pos] = num;
                    array_sizes[currentArrayIndex]++;

                    cout << "\n--Value inserted into Array " <<
(currentArrayIndex + 1) << ".--" << endl;
                }
                break;
            }
            case 2: {
                // DELETE from the current active array
                if (array_sizes[currentArrayIndex] == 0) {
                    cout << "\n--Current array is empty. Cannot delete an
element.--" << endl;
                } else {
                    int r, c;
                    cout << "Enter the row (0 to " << ROWS - 1 << "): ";
                    cin >> r;
                    cout << "Enter the column (0 to " << COLS - 1 << "): ";
                    cin >> c;

                    if (r < 0 || r >= ROWS || c < 0 || c >= COLS) {
                        cout << "\n--Invalid row or column!--" << endl;
                        break;
                    }

                    int pos = r * COLS + c;
                    if (pos < 0 || pos >= array_sizes[currentArrayIndex]) {
                        cout << "\n--No element at that position to delete!--" <<
endl;
                        break;
                    }

                    // Shift left
                    for (int i = pos; i < array_sizes[currentArrayIndex] - 1;
i++) {
                        arrays[currentArrayIndex][i] =
arrays[currentArrayIndex][i + 1];
                    }
                    array_sizes[currentArrayIndex]--;

                    cout << "\n--Value deleted from Array " << (currentArrayIndex
+ 1) << ".--" << endl;
                }
                break;
            }
            case 3: {
                if (array_count == MAX_ARRAYS) {
                    cout << "-- Maximum array limit reached. Cannot create more
arrays. --" << endl;
                } else {
                    int sourceIndex = array_count - 1; // last created array
index
                    int newIndex = array_count;        // new array index

                    int srcSize = array_sizes[sourceIndex];
                    // Copy elements
                    for (int i = 0; i < srcSize; i++) {
                        arrays[newIndex][i] = arrays[sourceIndex][i];
                    }
                    array_sizes[newIndex] = srcSize;

                    array_count++;
                    currentArrayIndex = newIndex; // Switch active array to the
newly created one

                    cout << "Array " << (sourceIndex + 1) << " copied
successfully to Array " << array_count
                         << ". Now operating on Array " << (currentArrayIndex +
1) << "." << endl;
                }
                break;
            }
            case 4:
                // EXIT
                exitProgram = true;
                cout << "Exiting the program." << endl;
                break;
            default:
                cout << "Invalid choice! Please try again." << endl;
        }

        // After each operation (except exit), print all arrays
        if (!exitProgram) {
            cout << "\n--- CURRENT ARRAYS ---" << endl;
            for (int i = 0; i < array_count; i++) {
                print2DArray(arrays[i], array_sizes[i], i + 1);
            }
        }
    }
}

    // Activity 9 Function
    // Function to print the 4x4 array
    void printArray(int arr[], int rows, int cols) {
        cout << "Current array state:" << endl;
        for(int i = 0; i < rows; i++) {
            for(int j = 0; j < cols; j++) {
                if(arr[i * cols + j] == -1)
                    cout << "-" << "\t";
                else
                    cout << arr[i * cols + j] << "\t";
            }
            cout << endl;
        }
    }

void activity9() {
    // December 11, 2024
    /* Create a program that have an array of 4x4 bidimensional elemets that can
perform insertion (push and enqueue) and deletion (pop and (dequeue).
    You must ask the user which operation to use and print the array every after
selected operation. There should be a terminate option if the user wishes to end
    the program. */

    cout << "Activity 9: Lists, Stacks and Queues, array implementation\n";
    const int ROWS = 4;
    const int COLS = 4;
    const int SIZE = ROWS * COLS;
    int arr[SIZE];

    // Initialize array with user input
    cout << "Enter 16 numerical elements to initialize the 4x4 array:" << endl;
    for(int i = 0; i < SIZE; i++){
        cout << "Element " << i + 1 << ": ";
        cin >> arr[i];
    }

    // Print initial array
    cout << "\nInitial array:" << endl;
    printArray(arr, ROWS, COLS);

    while(true){
        // Display menu
        cout << "\nSelect an operation:" << endl;
        cout << "1. Push" << endl;
        cout << "2. Pop" << endl;
        cout << "3. Enqueue" << endl;
        cout << "4. Dequeue" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice (1-5): ";
        int choice;
        cin >> choice;

        if(choice == 1){ // Push
            // Check if array is full (no -1 present)
            bool full = true;
            for(int i = 0; i < SIZE; i++) {
                if(arr[i] == -1){
                    full = false;
                    break;
                }
            }
            if(full){
                cout << "Error: Array is full. Cannot perform push operation." <<
endl;
                // Print the current array state
                printArray(arr, ROWS, COLS);
            }
            else{
                // Shift elements to the right to make space at front
                for(int i = SIZE - 1; i > 0; i--){
                    arr[i] = arr[i - 1];
                }
                // Insert new element at front
                int elem;
                cout << "Enter number to push: ";
                cin >> elem;
                arr[0] = elem;
                cout << "\nArray after push:" << endl;
                printArray(arr, ROWS, COLS);
            }
        }
        else if(choice == 2){ // Pop
            // Check if array is empty (all -1)
            bool empty = true;
            for(int i = 0; i < SIZE; i++) {
                if(arr[i] != -1){
                    empty = false;
                    break;
                }
            }
            if(empty){
                cout << "Error: Array is empty. Cannot perform pop operation." <<
endl;
                // Print the current array state
                printArray(arr, ROWS, COLS);
            }
            else{
                // Remove first element and shift left
                for(int i = 0; i < SIZE - 1; i++){
                    arr[i] = arr[i + 1];
                }
                // Set last element to -1
                arr[SIZE - 1] = -1;
                cout << "\nArray after pop:" << endl;
                printArray(arr, ROWS, COLS);
            }
        }
        else if(choice == 3){ // Enqueue
            // Check if array is full (no -1 present)
            bool full = true;
            for(int i = 0; i < SIZE; i++) {
                if(arr[i] == -1){
                    full = false;
                    break;
                }
            }
            if(full){
                cout << "Error: Array is full. Cannot perform enqueue operation."
<< endl;
                // Print the current array state
                printArray(arr, ROWS, COLS);
            }
            else{
                // Shift elements to the right to make space at front
                for(int i = SIZE - 1; i > 0; i--){
                    arr[i] = arr[i - 1];
                }
                // Insert new element at front
                int elem;
                cout << "Enter number to enqueue: ";
                cin >> elem;
                arr[0] = elem;
                cout << "\nArray after enqueue:" << endl;
                printArray(arr, ROWS, COLS);
            }
        }
        else if(choice == 4){ // Dequeue
            // Check if array is empty (all -1)
            bool empty = true;
            for(int i = 0; i < SIZE; i++) {
                if(arr[i] != -1){
                    empty = false;
                    break;
                }
            }
            if(empty){
                cout << "Error: Array is empty. Cannot perform dequeue
operation." << endl;
                // Print the current array state
                printArray(arr, ROWS, COLS);
            }
            else{
                // Remove last element by setting it to -1
                int pos = -1;
                for(int i = SIZE - 1; i >=0; i--){
                    if(arr[i] != -1){
                        pos = i;
                        break;
                    }
                }
                if(pos != -1){
                    arr[pos] = -1;
                    cout << "\nArray after dequeue:" << endl;
                    printArray(arr, ROWS, COLS);
                }
            }
        }
        else if(choice == 5){
            cout << "Exiting the program. Goodbye!" << endl;
            break;
        }
        else{
            cout << "Invalid choice. Please select a valid operation (1-5)." <<
endl;
        }
    }
}

    // Activity 10 Function
    void banner() {
        cout << R"(
    +-----------------------------------------+
    | _____ ____   ____   ____                |
    || ____|  _ \ / ___| |  _ \ ___ _ __   ___ |
    ||  _| | | | | |     | |_) / _ \ '_ \/ __||
    || |___| |_| | |___  |  __/  __/ | | \__ \|
    ||_____|____/ \____| |_|   \___|_| |_|___/|
    +-----------------------------------------+
    )";
    }

    void displayMenu(string vendingMachine[4][4], int prices[4][4], int
quantities[4][4]) {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                if (quantities[i][j] > 0) {
                    cout << "[" << i + 1 << "," << j + 1 << "] [PHP" <<
prices[i][j] << "] " << vendingMachine[i][j]
                         << " (" << quantities[i][j] << "x) \t";
                } else {
                    cout << "[" << i + 1 << "," << j + 1 << "] -       \t\t\t\t";
                }
            }
            cout << endl;
        }
        cout << "----------------------------\n";
    }
```

```
void activity10() {
    cout << "Activity 10: Algorithm Development\n\n";
    banner();
    string vendingMachine[4][4] = {
        {"Parker Jotter", "P. Metropolitan", "Lamy Safari", "P. Vanishing
Point"},
        {"Kaweco Art Sport", "Sailor Pro", "Lamy LX Safari", "Kaweco Classic"},
        {"Pilot E95S", "Aurora Ipsilon", "Lamy Dialog CC", "Platinum 3776"},
        {"Parker Sonnet", "Pelikan M800", "Platinum Peppy", "Pelikan M200"}
    };

    int prices[4][4] = {
        {1500, 1200, 1600, 9900},
        {5900, 17560, 1799, 1499},
        {8960, 7560, 30240, 24199},
        {1200, 48000, 550, 14995}
    };

    int quantities[4][4] = {
        {3, 1, 3, 4},
        {9, 5, 5, 7},
        {5, 1, 6, 8},
        {3, 1, 5, 2}
    };

    int insertedAmount = 0, totalAmount = 0;
    int purchaseCount[4][4] = {0};
    char proceed;

    do {
        displayMenu(vendingMachine, prices, quantities);

        cout << "Press any key to continue: ";
        cin >> proceed;

        // Insert payment
        cout << "Enter your payment (PHP): ";
        cin >> insertedAmount;

        if (insertedAmount <= 0) {
            cout << "Invalid payment. Payment refunded: PHP" << insertedAmount <<
endl;
            continue;
        }

        while (true) {
            int row, column;
            cout << "\nEnter the row and column of the item you want to purchase
(e.g., 1 2): ";
            cin >> row >> column;

            row--; // Convert to array index
            column--;
            if (row < 0 || row >= 4 || column < 0 || column >= 4 ||
quantities[row][column] <= 0) {
                cout << "Invalid selection or item out of stock. Try again.\n";
                continue;
            }

            int itemPrice = prices[row][column];
            if (insertedAmount < itemPrice) {
                cout << "Insufficient funds. Item price: PHP" << itemPrice << ".
Your balance: PHP" << insertedAmount << endl;
                cout << "Do you want to add more money? (y/n): ";
                cin >> proceed;

                if (tolower(proceed) == 'y') {
                    int additionalAmount;
                    cout << "Enter additional payment (PHP): ";
                    cin >> additionalAmount;
                    insertedAmount += additionalAmount;
                } else {
                    break;
                }
            } else {
                // Process purchase
                quantities[row][column]--;
                insertedAmount -= itemPrice;
                totalAmount += itemPrice;
                purchaseCount[row][column]++;
                cout << "\nPurchased: " << vendingMachine[row][column] <<
"\nRemaining Balance: PHP" << insertedAmount << endl;

                displayMenu(vendingMachine, prices, quantities);

                cout << "Do you want to buy another item? (y/n): ";
                cin >> proceed;

                if (tolower(proceed) != 'y') {
                    break;
                }
            }
        }

        // Print receipt
        cout << "\n--- RECEIPT ---\n";
        cout << "Every Day Carry Pens" << "\nActivity 10\n\n";
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                if (purchaseCount[i][j] > 0) {
                    cout << purchaseCount[i][j] << "x " << vendingMachine[i][j]
<< " - PHP"
                         << prices[i][j] * purchaseCount[i][j] << endl;
                }
            }
        }

        cout << "----------------------------\n";
        cout << "Total Amount: PHP" << totalAmount << endl;
        cout << "Change: PHP" << insertedAmount << endl;
        cout << "----------------------------\n";

        // Reset for another transaction
        cout << "Do you want to make another transaction? (y/n): ";
        cin >> proceed;
        if (tolower(proceed) == 'y') {
            totalAmount = 0;
            insertedAmount = 0;
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                    purchaseCount[i][j] = 0;
                }
            }
        }
    } while (tolower(proceed) == 'y');

    cout << "Thank you for using Every Day Carry Pens!\n";
}

// Main Menu
void mainMenu() {
    int choice;
    do {
        cout << "\n\n\n--- Main Menu ---\n";
        cout << "1. Activity 1: Introduction to C++ Programming\n";
        cout << "2. Activity 2: Introduction to C++ Programming (Week 2)\n";
        cout << "3. Activity 3: Control Statements\n";
        cout << "4. Activity 4: Pointers, References, and Dynamic Memory
Allocation\n";
        cout << "5. Activity 5: Array and Multi-Dimensional Array\n";
        cout << "6. Activity 6: Search Algorithm\n";
        cout << "7. Activity 7: Sorting Algorithm\n";
        cout << "8. Activity 8: Other operations\n";
        cout << "9. Activity 9: Lists, Stacks and Queues, array
implementation\n";
        cout << "10. Activity 10: Alrogrithm Development\n";
        cout << "0. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: activity1(); break;
            case 2: activity2(); break;
            case 3: activity3(); break;
            case 4: activity4(); break;
            case 5: activity5(); break;
            case 6: activity6(); break;
            case 7: activity7(); break;
            case 8: activity8(); break;
            case 9: activity9(); break;
            case 10: activity10(); break;
            case 0: cout << "Exiting program. Thank you!\n"; break;
            default: cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 0);
}

int main() {
    mainMenu();
    return 0;
}
```