East West University
Department of Computer Science and Engineering
Course: CSE246 Algorithm
Section : (2,11)

Basic Concepts

Store a fixed-size sequential collection of elements in an array.

C Language	C++ Language

#include <stdio.h>
 	
int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
	#include <iostream>
	using namespace std;
	
	int main() {
	    int arr[5] = {1, 2, 3, 4, 5};
	    for (int i = 0; i < 5; i++) {
	        cout << arr[i] << " ";
	    }
	    return 0;
	}

Dynamic array stores a resizable, sequential collection of elements in contiguous memory.

C Language	C++ Language

#include <stdio.h>
#include <stdlib.h>
 	
int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
 	
    // Dynamically allocate memory for n integers
    int* arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
	#include <iostream>
	using namespace std;
	
	int main() {
	    int n;
	    cout << "Enter the number of elements: ";
	    cin >> n;
	
	    // Dynamically allocate memory for n integers
	    int* arr = new int[n];

printf("Memory allocation failed\n");
return 1;
}
 	
// Initialize and print the array
for (int i = 0; i < n; i++) {
arr[i] = i + 1;
printf("Element %d: %d\n", i, arr[i]);
}
 	
// Free the allocated memory
free(arr);
return 0;
}
	// Initialize and print the array
	for (int i = 0; i < n; i++) {
	arr[i] = i + 1;
	cout << "Element " << i << ": " << arr[i] << endl;
	}
	
	// Free the allocated memory
	Delete[ ] arr;
	return 0;
	}

Pointers with Arrays

C Language	C++ Language

#include <stdio.h>
 	
int main() {
int arr[3] = {10, 20, 30};
int* ptr = arr; // arr is treated as a pointer to the first element
 	
printf("Accessing array elements via pointer:\n");
for (int i = 0; i < 3; i++) {
printf("Element %d: %d\n", i, *(ptr + i)); // Using pointer arithmetic
}
 	
return 0;
}
	#include <iostream>
	using namespace std;
	
	int main() {
	int arr[3] = {10, 20, 30};
	int* ptr = arr; // arr is treated as a pointer to the first element
	
	cout << "Accessing array elements via pointer:" << endl;
	for (int i = 0; i < 3; i++) {
	cout << "Element " << i << ": " << *(ptr + i) << endl; // Using pointer arithmetic
	}
	
	return 0;
	}

Pointer to Pointer

<table>
<tr><th>C Language</th><th>C++ Language</th></tr>
<tr><td>

```
#include <stdio.h>

int main() {
   int num = 20;
   int* ptr = &num;      // Pointer to an integer
   int** ptr2 = &ptr;    // Pointer to pointer

   printf("Value of num: %d\n", num);
   printf("Value at ptr (dereferencing ptr2): %d\n", **ptr2);

   return 0;
}
```

</td><td>

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

int main() {
   int num = 20;
   int* ptr = &num;      // Pointer to an integer
   int** ptr2 = &ptr;    // Pointer to pointer

   cout << "Value of num: " << num << endl;
   cout << "Value at ptr (dereferencing ptr2): " << **ptr2 << endl;

   return 0;
}
```

</td></tr>
</table>

Dynamic Memory Allocation of pointers

C Language	C++ Language

#include <stdio.h>
#include <stdlib.h>
 	
int main() {
    int* ptr = (int*)malloc(5 * sizeof(int)); // Allocates memory for 5 integers
 	
    if (ptr == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }
 	
    for (int i = 0; i < 5; i++) {
        ptr[i] = i + 1; // Assign values
        printf("%d ", ptr[i]);
    }
 	
    free(ptr); // Free the allocated memory
    return 0;
}
	#include <iostream>
	using namespace std;
	
	int main() {
	    int* ptr = new int[5]; // Allocates memory for 5 integers
	
	    for (int i = 0; i < 5; i++) {
	        ptr[i] = i + 1; // Assign values
	        cout << ptr[i] << " ";
	    }
	
	    Delete[ ] ptr; // Free the allocated memory
	    return 0;
	}

Linked Lists

C Language	C++ Language

#include <stdio.h>
#include <stdlib.h>
 	
struct Node {
    int data;
    struct Node* next;
};
 	
// Function to print linked list
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}
 	
int main() {
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;
 	
    // Allocate nodes in the heap
    head = (struct
	#include <iostream>
	using namespace std;
	
	class Node {
	public:
	    int data;
	    Node* next;
	};
	
	// Function to print linked list
	void printList(Node* node) {
	    while (node != NULL) {
	        cout << node->data << " -> ";
	        node = node->next;
	    }
	    cout << "NULL" << endl;
	}
	
	int main() {
	    Node* head = new Node();
	    Node* second = new Node();
	    Node* third = new Node();
	
	    // Assign data and link nodes

Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));
 	
    // Assign data and link nodes
    head->data = 1;
    head->next = second;
 	
    second->data = 2;
    second->next = third;
 	
    third->data = 3;
    third->next = NULL;
 	
    printList(head);
    return 0;
}
	head->data = 1;
	    head->next = second;
	
	    second->data = 2;
	    second->next = third;
	
	    third->data = 3;
	    third->next = NULL;
	
	    printList(head);
	    return 0;
	}

Stack Implementation

C Language	C++ Language

#include <stdio.h>
#define MAX 5 // Maximum size of the stack
 	
int stack[MAX];
int top = -1; // Initialize top as -1 to indicate an empty stack
 	
// Function to check if the stack is empty
int isEmpty() {
    return top == -1;
}
 	
// Function to check if the stack is full
int isFull() {
    return top == MAX - 1;
}
 	
// Function to add an element to the stack
void push(int value) {
    if (isFull()) {
        printf("Stack Overflow! Cannot push %d\n", value);
	#include <iostream>
	#include <stack> // Include the stack library
	using namespace std;
	
	int main() {
	    stack<int> s; // Declare a stack of integers
	
	    // Push elements onto the stack
	    s.push(10);
	    s.push(20);
	    s.push(30);
	
	    // Access the top element
	    cout << "Top element is " << s.top() << endl;
	
	    // Pop elements from the stack
	    s.pop();
	    cout << "Top element after pop is " << s.top() << endl;
	
	    // Check if the stack is empty

} else {
stack[++top] = value;
printf("Pushed %d to stack\n", value);
}
}
 	
// Function to remove an element from the stack
void pop() {
if (isEmpty()) {
printf("Stack Underflow! Cannot pop\n");
} else {
printf("Popped %d from stack\n", stack[top--]);
}
}
 	
// Function to get the top element
int peek() {
if (isEmpty()) {
printf("Stack is empty\n");
return -1;
}
return stack[top];
}
 	
int main() {
push(10);
push(20);
push(30);
printf("Top element is %d\n", peek());
pop();
pop();
printf("Top element after pops is %d\n", peek());
 	
return 0;
}
	if (s.empty()) {
	cout << "Stack is empty" << endl;
	} else {
	cout << "Stack is not empty" << endl;
	}
	
	return 0;
	}

Adjacency List Representation

C Language	C++ Language

```
#include <stdio.h>
#include <stdlib.h>

// Structure to represent a node in the
adjacency list
struct Node {
   int vertex;
   struct Node* next;
};

// Structure to represent an adjacency list
struct Graph {
   int numVertices;
   struct Node** adjLists;  // Array of
adjacency lists
};

// Function to create a new node
struct Node* createNode(int v) {
   struct Node* newNode =
malloc(sizeof(struct Node));
   newNode->vertex = v;
   newNode->next = NULL;
   return newNode;
}

// Function to create a graph
struct Graph* createGraph(int vertices) {
   struct Graph* graph =
malloc(sizeof(struct Graph));
   graph->numVertices = vertices;
   graph->adjLists = malloc(vertices *
sizeof(struct Node*));

   for (int i = 0; i < vertices; i++) {
      graph->adjLists[i] = NULL;  //
Initialize all adjacency lists as empty
   }
   return graph;
}

// Function to add an edge to the graph
void addEdge(struct Graph* graph, int
src, int dest) {
   // Add edge from src to dest
   struct Node* newNode =
createNode(dest);
   newNode->next =
graph->adjLists[src];
   graph->adjLists[src] = newNode;

   // Add edge from dest to src (for
undirected graph)
   newNode = createNode(src);
```

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

// Class to represent a graph using an
adjacency list
class Graph {
   int numVertices;  // Number of vertices
   vector<vector<int>> adjLists;  //
Adjacency list

public:
   Graph(int vertices) {
      numVertices = vertices;
      adjLists.resize(vertices);  // Resize to
hold the number of vertices
   }

   void addEdge(int src, int dest) {
      // Add edge from src to dest
      adjLists[src].push_back(dest);
      // Add edge from dest to src (for
undirected graph)
      adjLists[dest].push_back(src);
   }

   void printGraph() {
      for (int i = 0; i < numVertices; i++) {
         cout << "Vertex " << i << ":";
         for (int j : adjLists[i]) {
            cout << " -> " << j;
         }
         cout << endl;
      }
   }
};

int main() {
   Graph graph(5);

   // Adding edges
   graph.addEdge(0, 1);
   graph.addEdge(0, 4);
   graph.addEdge(1, 2);
   graph.addEdge(1, 3);
   graph.addEdge(1, 4);
   graph.addEdge(3, 4);

   graph.printGraph();

   return 0;
}
```

```
    newNode->next =
graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// Function to print the graph
void printGraph(struct Graph* graph) {
    for (int i = 0; i < graph->numVertices;
i++) {
        struct Node* temp =
graph->adjLists[i];
        printf("Vertex %d:", i);
        while (temp) {
            printf(" -> %d", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}

int main() {
    struct Graph* graph = createGraph(5);

    // Adding edges
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 4);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 3, 4);

    printGraph(graph);

    return 0;
}
```

Note: For a solid understanding of the fundamentals of the C++ language, you may refer to Neso Academy's C++ series.

Link: https://youtube.com/@nesoacademy?si=nqFR076gYff0Tsj8