

COURSE TITLE : DATA STRUCTURES AND ALGORITHMS INSTRUCTOR : MR. MARK KENNETH LIMJOCO

1. Array Data Structure

- **Key Concept**

An **array** is a fundamental data structure that stores a collection of elements of the same data type in contiguous memory locations. Each element in an array is identified by an index, and you can access elements by their index.

- **Learning Tasks**

- **Merging four arrays into one.**

```
ViñasJudahPaulo_BSIT2F.cpp
1 // MERGING ARRAYS
2 #include <iostream>
3 using namespace std;
4
5 int main() {
6     int arr1[] = {1, 2, 3};
7     int arr2[] = {4, 5, 6};
8     int arr3[] = {7, 8, 9};
9     int arr4[] = {10, 11, 12};
10    int merged[12]; // Merged array (size - 12)
11
12    // Copy elements from arr1 to merged
13    for (int i = 0; i < 3; i++) {
14        merged[i] = arr1[i];
15    }
16
17    // Copy elements from arr2 to merged
18    for (int i = 0; i < 3; i++) {
19        merged[i + 3] = arr2[i];
20    }
21
22    // Copy elements from arr3 to merged
23    for (int i = 0; i < 3; i++) {
24        merged[i + 6] = arr3[i];
25    }
26
27    // Copy elements from arr4 to merged
28    for (int i = 0; i < 3; i++) {
29        merged[i + 9] = arr4[i];
30    }
31
32    // Print the merged array
33    cout << "MERGED : ";
34    for (int i = 0; i < 12; i++) {
35        cout << merged[i] << " ";
36    }
37    return 0;
38 }
```

```
DA'SCHOOL\Data Structure and Algorithms\ViñasJudahPaulo_B...
MERGED : 1 2 3 4 5 6 7 8 9 10 11 12
Process exited after 0.1344 seconds with return value 0
Press any key to continue . . .
```

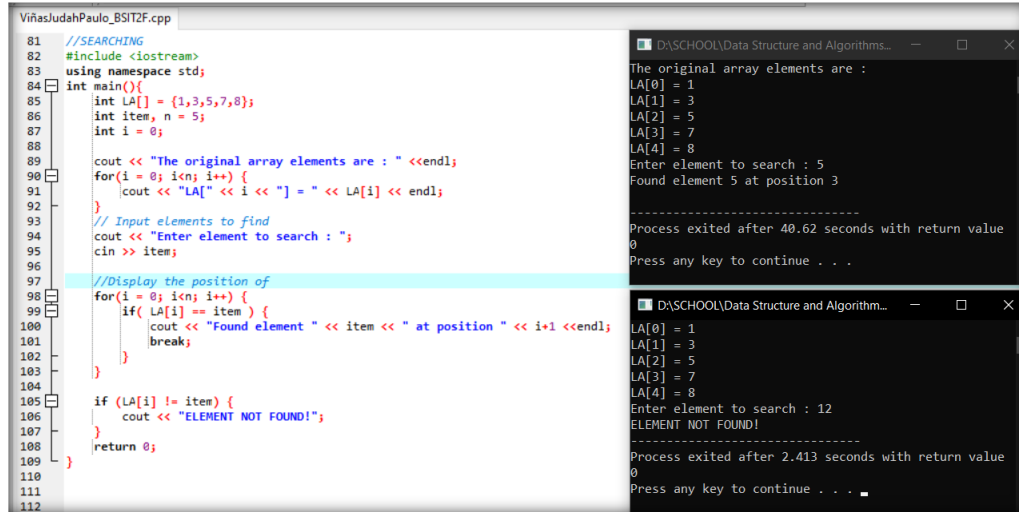
- **Delete element in an array.**

```
ViñasJudahPaulo_BSIT2F.cpp
40 //DELETION
41 #include <iostream>
42 using namespace std;
43
44 int main() {
45     int i, j, n = 3, elem;
46     int LA[3] = {1, 3, 5};
47     bool x=true;
48
49     cout << "The original array elements are : " << endl;
50     for (i = 0; i < n; i++) {
51         cout << "LA[" << i << "] = " << LA[i] << endl;
52     }
53     cout << "Enter an element to delete : ";
54     cin >> elem;
55
56     for (i = 0; i < n; i++) {
57         if (LA[i] == elem) {
58             for (j = i; j < (n - 1); j++) {
59                 LA[j] = LA[j + 1];
60             }
61             n--; // Reduce the size of the array
62             i--; // Recheck the current index in case the next element is also to be deleted
63             x = false;
64         }
65     }
66
67     if (x) {
68         cout << "-- ELEMENT NOT FOUND! --";
69     }
70     else {
71         cout << "The array elements after deletion : " << endl;
72         for (i = 0; i < n; i++) {
73             cout << "LA[" << i << "] = " << LA[i] << endl;
74         }
75     }
76     return 0;
77 }
```

```
DA'SCHOOL\Data Structure and Algorithms\ViñasJ...
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
Enter an element to delete : 3
The array elements after deletion :
LA[0] = 1
LA[1] = 5
Process exited after 14.77 seconds with return value 0
Press any key to continue . . .

DA'SCHOOL\Data Structure and Algorithms\ViñasJ...
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
Enter an element to delete : 12
-- ELEMENT NOT FOUND! --
Process exited after 8.02 seconds with return value 0
Press any key to continue . . .
```

➤ **Search element in an array.**



```
81 //SEARCHING
82 #include <iostream>
83 using namespace std;
84 int main(){
85     int LA[] = {1,3,5,7,8};
86     int item, n = 5;
87     int i = 0;
88
89     cout << "The original array elements are : " << endl;
90     for(i = 0; i<n; i++) {
91         cout << "LA[" << i << "] = " << LA[i] << endl;
92     }
93     // Input elements to find
94     cout << "Enter element to search : ";
95     cin >> item;
96
97     //Display the position of
98     for(i = 0; i<n; i++) {
99         if( LA[i] == item ) {
100             cout << "Found element " << item << " at position " << i+1 << endl;
101             break;
102         }
103     }
104
105     if (LA[i] != item) {
106         cout << "ELEMENT NOT FOUND!";
107     }
108     return 0;
109 }
110
111
112
```

Output 1:

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
Enter element to search : 5
Found element 5 at position 3
Process exited after 40.62 seconds with return value 0
Press any key to continue . . .
```

Output 2:

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
Enter element to search : 12
ELEMENT NOT FOUND!
Process exited after 2.413 seconds with return value 0
Press any key to continue . . .
```

• **Takeaways**

- Arrays are collections of elements stored in contiguous memory locations, accessible by index.
- Merging arrays involves copying elements from multiple arrays into a single array.
- Deleting an element from an array may require shifting elements to fill the gap.
- Searching for an element in an array involves iterating through the array and comparing each element until a match is found or the end is reached.

2. LinkList Data Structure

- **Key Concept**

A **linked list** is a linear data structure in which elements are stored in nodes, and each node points to the next node in the sequence. Linked lists consist of a head (the first node) and a tail (the last node). They are dynamic in size and efficient for insertions and deletions.

- **Learning Tasks**

➤ **Merging of three LinkList into one.**

```
ViñasJudahPaulo_BSIT2F.cpp
112 // MERGING 3 LINKLIST
113 #include <bits/stdc++.h>
114 #include <iostream>
115 using namespace std;
116
117 class Node {
118 public:
119     int value;
120     Node* next;
121 };
122
123 // Function to merge two linked lists
124 void merge(Node* &head1, Node* &head2) {
125     if (head1 == NULL) {
126         head1 = head2;
127         return;
128     }
129     Node* temp = head1;
130     while (temp->next != NULL) {
131         temp = temp->next;
132     }
133     temp->next = head2;
134     head2 = NULL; // Set head2 to NULL to avoid dangling pointers.
135 }
136
137 int main() {
138     Node* head1 = NULL;
139     Node* head2 = NULL;
140     Node* head3 = NULL;
141
142     // Create the first Linked List
143     Node* one = new Node();
144     Node* two = new Node();
145     Node* three = new Node();
146
147     one->value = 10;
148     two->value = 20;
149     three->value = 30;
```

```
1st Linklist : 10 20 30
2nd Linklist : 40 50 60
3rd Linklist : 70 80 90
----- MERGED LINKLIST -----
10 20 30 40 50 60 70 80 90

Process exited after 0.1484 seconds with return value 0
Press any key to continue . . .
```

```
ViñasJudahPaulo_BSIT2F.cpp
150
151     one->next = two;
152     two->next = three;
153     three->next = NULL;
154
155     cout << "1st Linklist : ";
156     cout << one->value << " " << two->value << " " << three->value << endl;
157
158     // Create the second Linked List
159     Node* four = new Node();
160     Node* five = new Node();
161     Node* six = new Node();
162
163     four->value = 40;
164     five->value = 50;
165     six->value = 60;
166
167     four->next = five;
168     five->next = six;
169     six->next = NULL;
170
171     cout << "2nd Linklist : ";
172     cout << four->value << " " << five->value << " " << six->value << endl;
173
174     // Create the third Linked List
175     Node* seven = new Node();
176     Node* eight = new Node();
177     Node* nine = new Node();
178
179     seven->value = 70;
180     eight->value = 80;
181     nine->value = 90;
182
183     seven->next = eight;
184     eight->next = nine;
185     nine->next = NULL;
186
187     cout << "3rd Linklist : ";
```

```
1st Linklist : 10 20 30
2nd Linklist : 40 50 60
3rd Linklist : 70 80 90
----- MERGED LINKLIST -----
10 20 30 40 50 60 70 80 90

Process exited after 0.1484 seconds with return value 0
Press any key to continue . . .
```

```

186  cout << "3rd LinkList : ";
187  cout << seven->value << " " << eight->value << " " << nine->value << endl;
188
189
190  // Merge all three linked lists
191  merge(head1, one);
192  merge(head1, four);
193  merge(head1, seven);
194
195  // Print the merged linked list
196  Node* current = head1;
197  cout << "----- MERGED LINKLIST -----" << endl;
198  while (current != NULL) {
199      cout << current->value << " ";
200      current = current->next;
201  }
202  cout << endl;
203
204  return 0;
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
  
```

```

D:\SCHOOL\Data Structure and Algorithms\Vi+asJudahPa...
1st LinkList : 10 20 30
2nd LinkList : 40 50 60
3rd LinkList : 70 80 90
----- MERGED LINKLIST -----
10 20 30 40 50 60 70 80 90
-----
Process exited after 0.1484 seconds with return value 0
Press any key to continue . . .
  
```

➤ **Delete node at the beginning and end.**

```

207  //DELETING NODE
208  #include <bits/stdc++.h>
209  #include <iostream>
210  using namespace std;
211
212  // creating a node
213  class Node {
214  public:
215      int value;
216      Node* next;
217  };
218
219  int main() {
220      Node* head;
221      Node* one = NULL;
222      Node* two = NULL;
223      Node* three = NULL;
224      Node* four = NULL;
225      Node* five = NULL;
226      // allocate 3 nodes in the heap
227      one = new Node();
228      two = new Node();
229      three = new Node();
230      four = new Node();
231      five = new Node();
232      // Assign value values
233      one->value = 1;
234      two->value = 2;
235      three->value = 3;
236      four->value = 45;
237      five->value = 55;
238      // Connect nodes
239      one->next = two;
240      two->next = three;
241      three->next = four;
242      four->next = five;
243      five->next = NULL;
244      // display before deleting
  
```

```

D:\SCHOOL\Data Structure and Algorithms\Vi+asJud...
----- Before Deleting -----
1
2
3
45
55
----- After Deleting -----
2
3
45
-----
Process exited after 0.1106 seconds with return value 0
Press any key to continue . . .
  
```

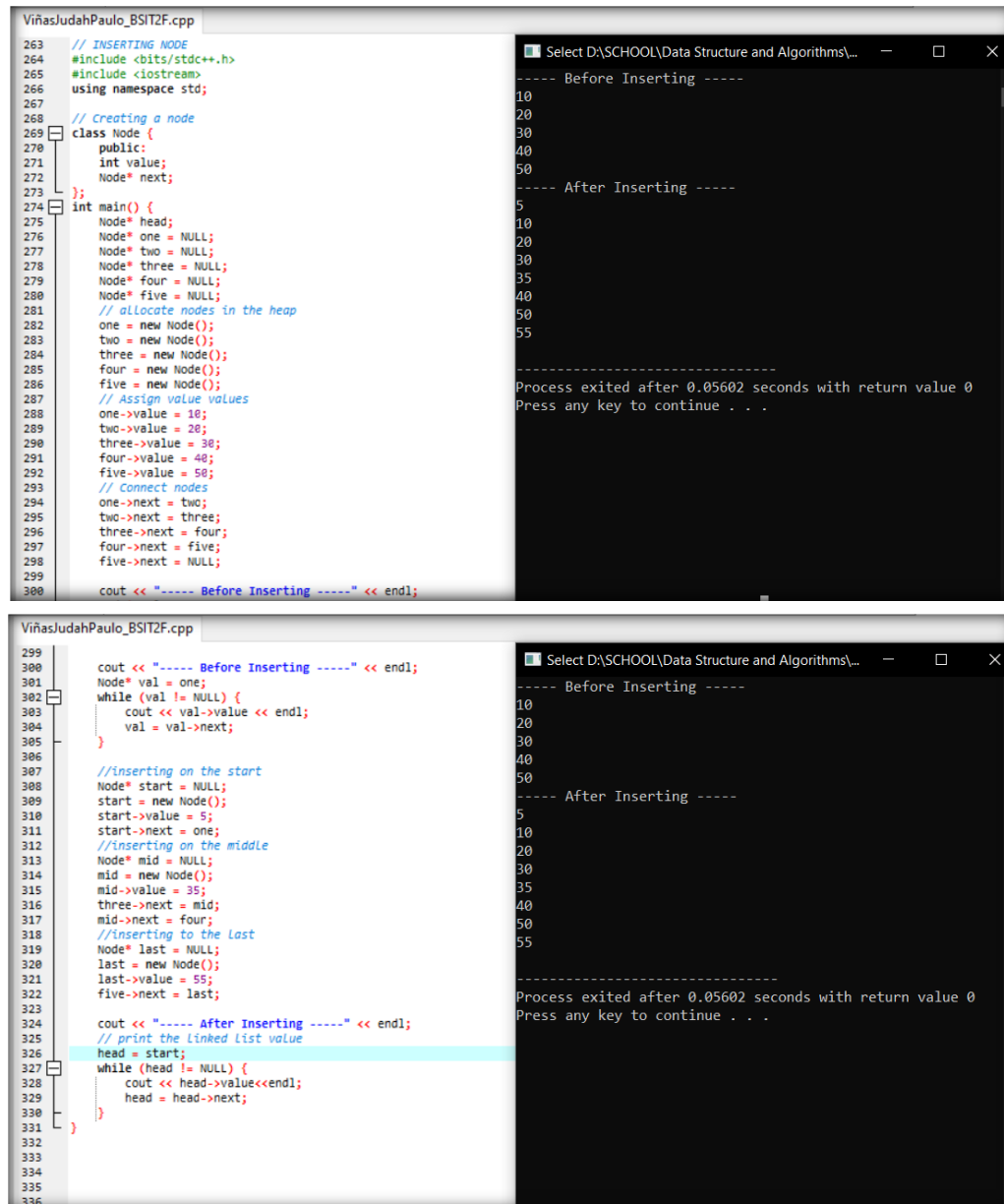
```

243  // display before deleting
244  Node* val = one;
245  cout << "----- Before Deleting -----" << endl;
246  while (val != NULL) {
247      cout << val->value << endl;
248      val = val->next;
249  }
250
251  cout << "----- After Deleting -----" << endl;
252  head = one;
253  head = head->next; //delete form start
254
255  // print the linked list value
256  while (head->next != NULL) { //delete form end
257      cout << head->value << endl;
258      head = head->next;
259  }
260
261
262
263
264
265
266
267
268
  
```

```

D:\SCHOOL\Data Structure and Algorithms\Vi+asJud...
----- Before Deleting -----
1
2
3
45
55
----- After Deleting -----
2
3
45
-----
Process exited after 0.1106 seconds with return value 0
Press any key to continue . . .
  
```

➤ **Insert node at the beginning, middle and end.**



The image shows two screenshots of a C++ program and its terminal output. The first screenshot shows the initial code for creating a linked list with five nodes (values 10, 20, 30, 40, 50) and printing them. The second screenshot shows the code for inserting new nodes at the beginning, middle, and end of the list, followed by printing the updated list. The terminal output shows the list before and after each insertion operation.

```
ViñasJudahPaulo_BSIT2F.cpp
263 // INSERTING NODE
264 #include <bits/stdc++.h>
265 #include <iostream>
266 using namespace std;
267
268 // Creating a node
269 class Node {
270 public:
271     int value;
272     Node* next;
273 };
274
275 int main() {
276     Node* head;
277     Node* one = NULL;
278     Node* two = NULL;
279     Node* three = NULL;
280     Node* four = NULL;
281     Node* five = NULL;
282     // allocate nodes in the heap
283     one = new Node();
284     two = new Node();
285     three = new Node();
286     four = new Node();
287     five = new Node();
288     // Assign value values
289     one->value = 10;
290     two->value = 20;
291     three->value = 30;
292     four->value = 40;
293     five->value = 50;
294     // Connect nodes
295     one->next = two;
296     two->next = three;
297     three->next = four;
298     four->next = five;
299     five->next = NULL;
300     cout << "----- Before Inserting -----" << endl;
301
302     //inserting on the start
303     Node* start = NULL;
304     start = new Node();
305     start->value = 5;
306     start->next = one;
307
308     //inserting on the middle
309     Node* mid = NULL;
310     mid = new Node();
311     mid->value = 35;
312     three->next = mid;
313     mid->next = four;
314
315     //inserting to the Last
316     Node* last = NULL;
317     last = new Node();
318     last->value = 55;
319     five->next = last;
320
321     cout << "----- After Inserting -----" << endl;
322     // print the linked list value
323     head = start;
324     while (head != NULL) {
325         cout << head->value << endl;
326         head = head->next;
327     }
328
329 }
```

```
Select D:\SCHOOL\Data Structure and Algorithms\...
----- Before Inserting -----
10
20
30
40
50
----- After Inserting -----
5
10
20
30
35
40
50
55
-----
Process exited after 0.05602 seconds with return value 0
Press any key to continue . . .
```

```
Select D:\SCHOOL\Data Structure and Algorithms\...
----- Before Inserting -----
10
20
30
40
50
----- After Inserting -----
5
10
20
30
35
40
50
55
-----
Process exited after 0.05602 seconds with return value 0
Press any key to continue . . .
```

• **Takeaways**

- Linked lists are dynamic data structures where elements are stored in nodes, and each node has a pointer to the next node.
- Merging linked lists involves updating the next pointers appropriately.
- Deleting and inserting nodes at different positions in a linked list require updating pointers accordingly.
- Linked lists are useful when you need dynamic size and efficient insertions and deletions compared to arrays.

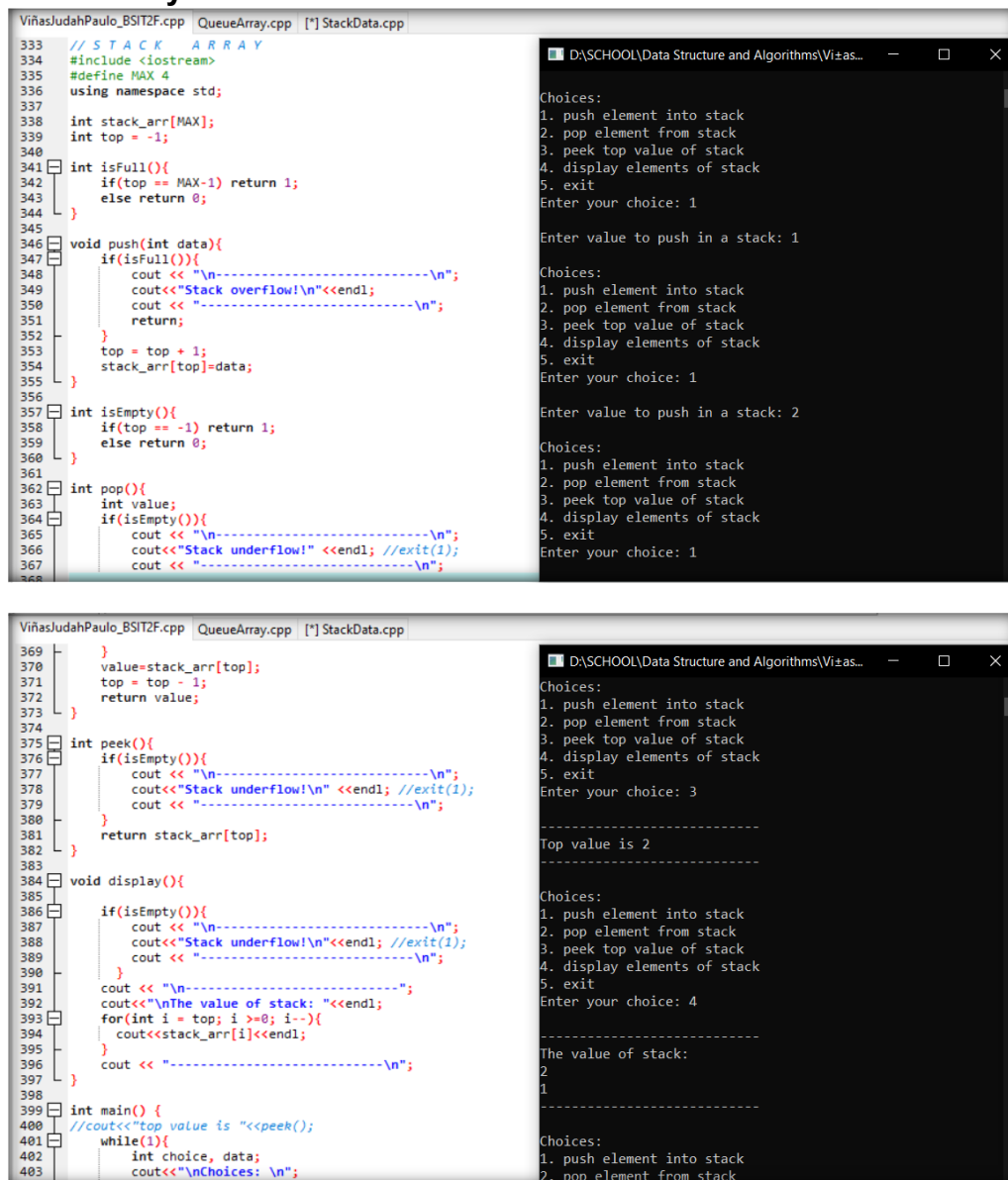
3. Stack Data Structure

- **Key Concept**

A **stack** is a linear data structure that follows the Last-In-First-Out (LIFO) principle. In a stack, elements are added and removed from the same end, known as the top. It is often used for tasks that involve keeping track of the order of elements, such as function call execution and expression evaluation.

- **Learning Tasks**

- **Stack Array**



```
ViñalJudahPaulo_BSITZF.cpp QueueArray.cpp [*] StackData.cpp
333 // STACK ARRAY
334 #include <iostream>
335 #define MAX 4
336 using namespace std;
337
338 int stack_arr[MAX];
339 int top = -1;
340
341 int isFull(){
342     if(top == MAX-1) return 1;
343     else return 0;
344 }
345
346 void push(int data){
347     if(isFull()){
348         cout << "\n-----\n";
349         cout << "Stack overflow!\n" << endl;
350         cout << "-----\n";
351         return;
352     }
353     top = top + 1;
354     stack_arr[top]=data;
355 }
356
357 int isEmpty(){
358     if(top == -1) return 1;
359     else return 0;
360 }
361
362 int pop(){
363     int value;
364     if(isEmpty()){
365         cout << "\n-----\n";
366         cout << "Stack underflow!" << endl; //exit(1);
367         cout << "-----\n";
368     }
369     value=stack_arr[top];
370     top = top - 1;
371     return value;
372 }
373
374
375 int peek(){
376     if(isEmpty()){
377         cout << "\n-----\n";
378         cout << "Stack underflow!\n" << endl; //exit(1);
379         cout << "-----\n";
380     }
381     return stack_arr[top];
382 }
383
384 void display(){
385     if(isEmpty()){
386         cout << "\n-----\n";
387         cout << "Stack underflow!\n" << endl; //exit(1);
388         cout << "-----\n";
389     }
390     cout << "\n-----\n";
391     cout << "The value of stack: " << endl;
392     for(int i = top; i >=0; i--){
393         cout << stack_arr[i] << endl;
394     }
395     cout << "-----\n";
396 }
397
398
399 int main() {
400     //cout << "top value is "<< peek();
401     while(1){
402         int choice, data;
403         cout << "\nChoices: \n";
404         cout << "1. push element into stack \n";
405         cout << "2. pop element from stack \n";
406         cout << "3. peek top value of stack \n";
407         cout << "4. display elements of stack \n";
408         cout << "5. exit \n";
409         int choice;
410         cout << "Enter your choice: ";
411         choice = getche();
412         switch(choice){
413             case '1':
414                 cout << "\nEnter value to push in a stack: ";
415                 data = getche();
416                 push(data);
417                 break;
418             case '2':
419                 pop();
420                 break;
421             case '3':
422                 cout << "\nTop value is ";
423                 cout << peek() << endl;
424                 break;
425             case '4':
426                 display();
427                 break;
428             case '5':
429                 exit(0);
430                 break;
431             default:
432                 cout << "\nInvalid choice! \n";
433                 break;
434         }
435     }
436 }
```

Choices:
1. push element into stack
2. pop element from stack
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice: 1
Enter value to push in a stack: 1
Choices:
1. push element into stack
2. pop element from stack
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice: 1
Enter value to push in a stack: 2
Choices:
1. push element into stack
2. pop element from stack
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice: 1
Enter value to push in a stack: 3
Choices:
1. push element into stack
2. pop element from stack
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice: 3
Top value is 2
Choices:
1. push element into stack
2. pop element from stack
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice: 4
The value of stack:
3
2
1
Choices:
1. push element into stack
2. pop element from stack

```
ViñasJudahPaulo_BSIT2F.cpp QueueArray.cpp StackData.cpp
399 int main() {
400     //cout<<"top value is "<<peek();
401     while(1){
402         int choice, data;
403         cout<<"\nChoices: \n";
404         cout<<"1. push element into stack \n";
405         cout<<"2. pop element from stack \n";
406         cout<<"3. peek top value of stack \n";
407         cout<<"4. display elements of stack \n";
408         cout<<"5. exit \n";
409         cout<<"Enter your choice: ";
410         cin>>choice;
411         switch(choice){
412             case 1:
413                 cout<<"\nEnter value to push in a stack: ";
414                 cin>>data;
415                 push(data);
416                 break;
417             case 2:
418                 pop();
419                 break;
420             case 3:
421                 cout<<"\n-----\n";
422                 cout<<"\nTop value is "<<peek()<<"\n";
423                 cout<<"\n-----\n";
424                 break;
425             case 4:
426                 display();
427                 break;
428             case 5:
429                 exit(1);
430         }
431     }
432 }
433
434
```

```
D:\SCHOOL\Data Structure and Algorithms\ViñasJ...
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice: 4

-----
The value of stack:
2
1
-----

(Choices:
1. push element into stack
2. pop element from stack
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice:
```

➤ Stack LinkList

```
ViñasJudahPaulo_BSIT2F.cpp StackData.cpp QueueArray.cpp
437 // STACK LINKLIST
438 #include <iostream>
439 using namespace std;
440
441 struct node{
442     int value;
443     struct node* link;
444 } *top = NULL;
445
446 void push(int value){
447     struct node* newNode;
448     newNode = new node();
449     //newNode = malloc(sizeof(newNode));
450     if(newNode==NULL){
451         cout<<"Stack Overflow";
452         exit(1);
453     }
454     newNode->value=value;
455     newNode->link=NULL;
456     newNode->link=top;
457     top=newNode;
458 }
459 int isEmpty(){
460     if(top == NULL) return 1;
461     else return 0;
462 }
463 int pop(){
464     if(isEmpty()){
465         cout<<"Stack Underflow!";
466         exit(1);
467     }
468     int val = top->value;
469     top=top->link;
470     //free(top);
471     //top=NULL;
472     return val;
473 }
474
```

```
D:\SCHOOL\Data Structure and Algorithms\ViñasJudahPaul...
The topmost element of the stack is 1
The last deleted: 112

The stack Elements are:
1

-----
Process exited after 0.1446 seconds with return value 0
Press any key to continue . . .
```

```
ViñasJudahPaulo_BSIT2F.cpp StackData.cpp QueueArray.cpp
470 //free(top);
471 //top=NULL;
472 return val;
473 }
474 int peek(){
475     if(isEmpty()){
476         cout<<"Stack Underflow!";
477         exit(1);
478     }
479     return top->value;
480 }
481 int main() {
482     int val;
483     push(1);
484     push(112);
485     push(23);
486     val = pop();
487     val = pop(); //pop();pop();pop();pop();
488     cout<<"The topmost element of the stack is "<<peek()<<"\n";
489     cout<<"The last deleted: " << val <<"\n";
490     cout<<"The stack Elements are: \n";
491     while(top){
492         cout<<top->value<<"\n";
493         top=top->link;
494     }
495 }
496
497
```

```
D:\SCHOOL\Data Structure and Algorithms\ViñasJudahPaul...
The topmost element of the stack is 1
The last deleted: 112

The stack Elements are:
1

-----
Process exited after 0.1446 seconds with return value 0
Press any key to continue . . .
```



COLLEGE *of* COMPUTER STUDIES

- **Takeaways**

- A stack is a data structure that follows the LIFO (Last-In-First-Out) principle.
- Elements are added and removed from the top of the stack.
- The 'stack' container in C++ Standard Library simplifies stack operations.
- Stacks are commonly used in scenarios where the order of operations matters, such as function call execution, undo functionality, and expression evaluation.

4. Queue Data Structure

- **Key Concept**

A **queue** is a linear data structure that follows the First-In-First-Out (FIFO) principle. In a queue, elements are added at the rear (enqueue) and removed from the front (dequeue). It is often used in scenarios where elements are processed in the order they are added, like task scheduling, print queue, etc.

- **Learning Tasks**

- **Queue Array**

```
ViñasJudahPaulo_BSIT2F.cpp QueueArray.cpp
502 // QUEUE ARRAY
503 #include <iostream>
504 #define MAX 4
505 using namespace std;
506
507 int q_arr[MAX];
508 int rear = -1;
509 int front=0;
510 int count=0;
511
512 int isFull(){
513     return count == MAX;
514 }
515
516 void enqueue(int data){
517     if(!isFull()){
518         if(rear == MAX-1){
519             rear = -1;
520         }
521         q_arr[++rear] = data;
522         count++;
523     }else{
524         cout<<"Queue Overflow"<<endl;
525     }
526 }
527 int isEmpty(){
528     return count==0;
529 }
530
531 int dequeue(){
532     if(isEmpty()){
533         cout<<"Queue underflow!"<<endl; exit(1);
534     }
535     int value=q_arr[front++];
536     if(front==MAX){
537         front=0;
538     }
```

```
ViñasJudahPaulo_BSIT2F.cpp QueueArray.cpp
535     if(front==MAX){
536         front=0;
537     }
538     count--;
539     return value;
540 }
541
542 int peek(){
543     if(isEmpty()){
544         cout<<"Queue underflow!"<<endl; exit(1);
545     }
546     return q_arr[front];
547 }
548
549
550 int main() {
551     enqueue(22);
552     enqueue(2);
553     enqueue(233);
554     enqueue(244); //enqueue(4355);
555     dequeue();dequeue();//dequeue();dequeue();dequeue();
556     //cout<<"the first element is "<<peek()<<endl;
557     cout<<"Queue: ";
558     while(!isEmpty()){
559         int val = dequeue();
560         cout<<" "<<val;
561     }
562 }
```

```
D:\SCHOOL\Data Structure and Algorithms\ViñasJudahPaulo...
Queue:  233 244
-----
Process exited after 0.1198 seconds with return value 0
Press any key to continue . . .
```

```
D:\SCHOOL\Data Structure and Algorithms\ViñasJudahPaulo...
Queue:  233 244
-----
Process exited after 0.1198 seconds with return value 0
Press any key to continue . . .
```

➤ Queue LinkList

```
ViñalJudahPaulo_BSIT2F.cpp QueueArray.cpp
567 // QUEUE LINKLIST
568 #include <iostream>
569 using namespace std;
570
571 struct node{
572     int value;
573     struct node* link;
574 }
575 *front = NULL;
576 *rear = NULL;
577
578 int isEmpty(){
579     if(front == NULL) return 1;
580     else return 0;
581 }
582 void push(int data){
583     node* newNode = new node();
584     if (isEmpty()) {
585         front = newNode;
586         rear = newNode;
587     }
588     newNode->value=data;
589     rear->link = newNode;
590     rear = newNode;
591 }
592
593
594 int pop(){
595     if(isEmpty()){
596         cout<<"Stack Underflow!";
597         exit(1);
598     }
599     int val = front->value;
600     front=front->link;
601     return val;
602 }
```

```
D:\SCHOOL\Data Structure and Algorithms\ViñalJudahPa...
The stack Elements are:
63 63 63 63 63 63 63
-----
Process exited after 0.1318 seconds with return value 0
Press any key to continue . . .
```

```
ViñalJudahPaulo_BSIT2F.cpp QueueArray.cpp
597     exit(1);
598 }
599 int val = front->value;
600 front=front->link;
601 return val;
602
603
604 }
605 int peek(){
606     if(isEmpty()){
607         cout<<"Queue Underflow!";
608         exit(1);
609     }return front->value;
610 }
611
612 int main() {
613
614     push(112);
615     push(23);pop();pop();pop();pop();
616     push(53);
617     push(63);
618     pop();pop();pop();
619     push(63);push(63);push(63);push(63);push(63);push(63);
620     //cout<<"The topmost element of he stack is "<<peek()
621
622     cout<<"The stack Elements are: \n";
623     while(front){
624         cout<<front->value<<" ";
625         front=front->link;
626     }
627 }
628
629
630
631
632 }
```

```
D:\SCHOOL\Data Structure and Algorithms\ViñalJudahPa...
The stack Elements are:
63 63 63 63 63 63 63
-----
Process exited after 0.1318 seconds with return value 0
Press any key to continue . . .
```

• Takeaways

- A queue is a data structure that follows the FIFO (First-In-First-Out) principle.
- Elements are added to the rear (enqueue) and removed from the front (dequeue) of the queue.
- The 'queue' container in C++ Standard Library simplifies queue operations.
- Queues are commonly used in scenarios where the order of processing matters, such as task scheduling and managing resources.



Overall Reflection:

Course Impact:

The data structures course, which covered arrays, linked lists, queues, and stacks, greatly shaped my general understanding and perspective on data structures and algorithms. It has given me a firm understanding of the fundamental concepts, implementation details, and real-world applications of various data structures.

I obtained practical insights into how these data structures function and how they may be utilized for solving various computational problems through hands-on examples and code implementations. This knowledge has improved not just my programming skills but also my problem-solving ability.

The course also emphasized the importance of choosing the right data structure for individual applications, taking into account considerations such as efficiency, memory utilization, and ease of manipulation. It has increased my understanding of the importance of data structures in software development and their function in optimizing algorithmic solutions.

Recommendations:

- **Algorithm Analysis:** Include a section on algorithm analysis for a better understanding of time and space complexity in relation to data structures.
- **Problem-Solving Challenges:** Integrate coding challenges and problem-solving exercises that apply data structure concepts to real-world scenarios for practical experience.
- **Visualizations:** Incorporate visual aids and animations to make abstract data structure concepts more accessible and engaging.