

### Activity No. <3>

#### <Hands-on Activity 3.1 Linked Lists>

**Course Code:** CPE010

**Program:** Computer Engineering

**Course Title:** Data Structures and Algorithms

**Date Performed:** 8/14/25

**Section:** CPE21S4

**Date Submitted:** 8/14/25

**Name(s):** Alcantara, Jason P.

**Instructor:** Engr. Jimlord Quejado

#### 6. Output:



```

1 #include <iostream>
2
3 class Node {
4 public:
5     char data;
6     Node* next;
7 }
8
9 int main() {
10     // Step 1: Declare head pointer
11     Node* head = nullptr;
12
13     // Step 2: Create nodes
14     Node* second = new Node();
15     Node* third = new Node();
16     Node* fourth = new Node();
17     Node* fifth = new Node();
18     Node* last = new Node();
19
20     // Step 3: Assign data and link nodes
21     head = new Node();
22     head->data = 'C';
23     head->next = second;
24
25     second->data = 'P';
26     second->next = third;
27
28     third->data = 'E';
29     third->next = fourth;
30
31     fourth->data = 'I';
32     fourth->next = fifth;
33
34     fifth->data = 'O';
35     fifth->next = last;
36
37     last->data = 'NULL';
38     last->next = nullptr;
39
40     // Optional: print the list to verify
41     Node* temp = head;
42     while (temp != nullptr) {
43         std::cout << temp->data << " -> ";
44         temp = temp->next;
45     }
46     std::cout << "NULL" << std::endl;
47
48     return 0;
49 }

```

Compiler Resources Compiler Log Debug Find Results Console Close

Abort Compilation

Compilation results...

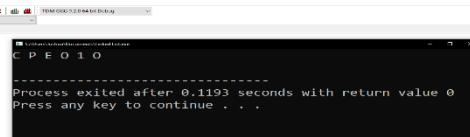
Execution: 0  
Output: 0  
Output Filename: C:\Users\Joshua\Documents\Linked List.exe  
Output Size: 3.072938474023 Mib  
Compilation Time: 0.18s

Line 50 Col 1 Set 0 Lines: 50 Length: 962 Insert Done parsing in 0.922 seconds

**Discussion:** I implemented linked list where I can store characters and display it on a sequence order by linking the nodes correctly.

#### Operation:

#### Traversal:



```

1 #include <iostream>
2
3 class Node {
4 public:
5     char data;
6     Node* next;
7 }
8
9 // Create a function to print the list
10 void printList(Node* head) {
11     Node* temp = head;
12     while (temp != nullptr) {
13         std::cout << temp->data << " -> ";
14         temp = temp->next;
15     }
16     std::cout << "NULL" << std::endl;
17 }
18
19 // Create a function to insert a node
20 void insertNode(Node** head, char data, Node* n = nullptr) {
21     Node* temp = *head;
22     if (*head == nullptr) {
23         *head = new Node(data);
24         *head->next = n;
25     } else {
26         while (temp->next != nullptr) {
27             temp = temp->next;
28         }
29         temp->next = n;
30     }
31 }
32
33 // Create a function to delete a node
34 void deleteNode(Node** head, char data) {
35     Node* temp = *head;
36     if (*head == nullptr) {
37         std::cout << "List is empty" << std::endl;
38     } else {
39         while (temp->next != nullptr) {
40             if (temp->next->data == data) {
41                 temp->next = temp->next->next;
42                 delete temp->next;
43                 break;
44             }
45             temp = temp->next;
46         }
47     }
48 }
49
50 // Create a function to search a node
51 void searchNode(Node* head, char data) {
52     Node* temp = head;
53     while (temp != nullptr) {
54         if (temp->data == data) {
55             std::cout << "Node found" << std::endl;
56             break;
57         }
58         temp = temp->next;
59     }
60 }
61
62 // Create a function to sort a list
63 void sortList(Node** head) {
64     Node* temp = *head;
65     Node* current = new Node('A');
66     Node* previous = nullptr;
67     Node* next;
68
69     while (temp != nullptr) {
70         if (temp->data > current->data) {
71             if (previous == nullptr) {
72                 *head = current;
73                 current->next = temp;
74                 temp = temp->next;
75                 current->next = temp;
76             } else {
77                 previous->next = current;
78                 current->next = temp;
79                 temp = temp->next;
80                 current->next = temp;
81             }
82         } else {
83             previous = temp;
84             temp = temp->next;
85         }
86     }
87 }
88
89 // Create a function to reverse a list
90 void reverseList(Node** head) {
91     Node* temp = *head;
92     Node* previous = nullptr;
93     Node* next;
94
95     while (temp != nullptr) {
96         next = temp->next;
97         temp->next = previous;
98         previous = temp;
99         temp = next;
100    }
101   *head = previous;
102 }
103
104 // Create a function to copy a list
105 void copyList(Node* source, Node** destination) {
106     Node* temp = source;
107     Node* dest = *destination;
108
109     while (temp != nullptr) {
110         dest = new Node(temp->data);
111         dest->next = *destination;
112         *destination = dest;
113         temp = temp->next;
114     }
115 }
116
117 // Create a function to merge two lists
118 void mergeList(Node* head1, Node* head2) {
119     Node* temp1 = head1;
120     Node* temp2 = head2;
121
122     while (temp1 != nullptr && temp2 != nullptr) {
123         if (temp1->data < temp2->data) {
124             temp1 = temp1->next;
125         } else {
126             temp2 = temp2->next;
127         }
128     }
129
130     if (temp1 == nullptr) {
131         temp1 = temp2;
132     }
133
134     *head1 = temp1;
135 }
136
137 // Create a function to find the middle of a list
138 void findMiddle(Node* head) {
139     Node* slow = head;
140     Node* fast = head;
141
142     while (fast->next != nullptr && fast->next->next != nullptr) {
143         slow = slow->next;
144         fast = fast->next->next;
145     }
146
147     std::cout << "Middle element is " << slow->data << std::endl;
148 }
149
150 // Create a function to find the intersection of two lists
151 void findIntersection(Node* head1, Node* head2) {
152     Node* temp1 = head1;
153     Node* temp2 = head2;
154
155     while (temp1 != nullptr && temp2 != nullptr) {
156         if (temp1->data == temp2->data) {
157             std::cout << "Intersection found" << std::endl;
158             break;
159         }
160         temp1 = temp1->next;
161         temp2 = temp2->next;
162     }
163 }
164
165 // Create a function to find the length of a list
166 void findLength(Node* head) {
167     Node* temp = head;
168     int count = 0;
169
170     while (temp != nullptr) {
171         count++;
172         temp = temp->next;
173     }
174
175     std::cout << "Length of the list is " << count << std::endl;
176 }
177
178 // Create a function to find the sum of all elements in a list
179 void findSum(Node* head) {
180     Node* temp = head;
181     int sum = 0;
182
183     while (temp != nullptr) {
184         sum += temp->data;
185         temp = temp->next;
186     }
187
188     std::cout << "Sum of all elements is " << sum << std::endl;
189 }
190
191 // Create a function to find the product of all elements in a list
192 void findProduct(Node* head) {
193     Node* temp = head;
194     int product = 1;
195
196     while (temp != nullptr) {
197         product *= temp->data;
198         temp = temp->next;
199     }
200
201     std::cout << "Product of all elements is " << product << std::endl;
202 }
203
204 // Create a function to find the average of all elements in a list
205 void findAverage(Node* head) {
206     Node* temp = head;
207     int sum = 0;
208     int count = 0;
209
210     while (temp != nullptr) {
211         sum += temp->data;
212         count++;
213         temp = temp->next;
214     }
215
216     std::cout << "Average of all elements is " << sum / count << std::endl;
217 }
218
219 // Create a function to find the maximum element in a list
220 void findMax(Node* head) {
221     Node* temp = head;
222     int max = head->data;
223
224     while (temp != nullptr) {
225         if (temp->data > max) {
226             max = temp->data;
227         }
228         temp = temp->next;
229     }
230
231     std::cout << "Maximum element is " << max << std::endl;
232 }
233
234 // Create a function to find the minimum element in a list
235 void findMin(Node* head) {
236     Node* temp = head;
237     int min = head->data;
238
239     while (temp != nullptr) {
240         if (temp->data < min) {
241             min = temp->data;
242         }
243         temp = temp->next;
244     }
245
246     std::cout << "Minimum element is " << min << std::endl;
247 }
248
249 // Create a function to find the frequency of an element in a list
250 void findFrequency(Node* head, char data) {
251     Node* temp = head;
252     int frequency = 0;
253
254     while (temp != nullptr) {
255         if (temp->data == data) {
256             frequency++;
257         }
258         temp = temp->next;
259     }
260
261     std::cout << "Frequency of " << data << " is " << frequency << std::endl;
262 }
263
264 // Create a function to find the mode of a list
265 void findMode(Node* head) {
266     Node* temp = head;
267     int mode = head->data;
268     int frequency = 1;
269
270     while (temp != nullptr) {
271         int currentFrequency = 0;
272
273         while (temp->data == mode) {
274             currentFrequency++;
275             temp = temp->next;
276         }
277
278         if (currentFrequency > frequency) {
279             mode = temp->data;
280             frequency = currentFrequency;
281         }
282     }
283
284     std::cout << "Mode of the list is " << mode << std::endl;
285 }
286
287 // Create a function to find the median of a list
288 void findMedian(Node* head) {
289     Node* temp = head;
290     int count = 0;
291
292     while (temp != nullptr) {
293         count++;
294         temp = temp->next;
295     }
296
297     if (count % 2 == 0) {
298         std::cout << "Median is " << (temp->data + temp->next->data) / 2 << std::endl;
299     } else {
300         std::cout << "Median is " << temp->data << std::endl;
301     }
302 }
303
304 // Create a function to find the variance of a list
305 void findVariance(Node* head) {
306     Node* temp = head;
307     int sum = 0;
308     int count = 0;
309
310     while (temp != nullptr) {
311         sum += temp->data;
312         count++;
313         temp = temp->next;
314     }
315
316     double mean = sum / count;
317
318     double variance = 0;
319
320     while (temp != nullptr) {
321         variance += (temp->data - mean) * (temp->data - mean);
322         temp = temp->next;
323     }
324
325     variance /= count;
326
327     std::cout << "Variance of the list is " << variance << std::endl;
328 }
329
330 // Create a function to find the standard deviation of a list
331 void findStandardDeviation(Node* head) {
332     Node* temp = head;
333     int sum = 0;
334     int count = 0;
335
336     while (temp != nullptr) {
337         sum += temp->data;
338         count++;
339         temp = temp->next;
340     }
341
342     double mean = sum / count;
343
344     double standardDeviation = 0;
345
346     while (temp != nullptr) {
347         standardDeviation += (temp->data - mean) * (temp->data - mean);
348         temp = temp->next;
349     }
350
351     standardDeviation /= count;
352
353     std::cout << "Standard Deviation of the list is " << standardDeviation << std::endl;
354 }
355
356 // Create a function to find the range of a list
357 void findRange(Node* head) {
358     Node* temp = head;
359     int min = head->data;
360     int max = head->data;
361
362     while (temp != nullptr) {
363         if (temp->data < min) {
364             min = temp->data;
365         }
366         if (temp->data > max) {
367             max = temp->data;
368         }
369         temp = temp->next;
370     }
371
372     std::cout << "Range of the list is " << max - min << std::endl;
373 }
374
375 // Create a function to find the skewness of a list
376 void findSkewness(Node* head) {
377     Node* temp = head;
378     int sum = 0;
379     int count = 0;
380
381     while (temp != nullptr) {
382         sum += temp->data;
383         count++;
384         temp = temp->next;
385     }
386
387     double mean = sum / count;
388
389     double skewness = 0;
390
391     while (temp != nullptr) {
392         skewness += (temp->data - mean) * (temp->data - mean) * (temp->data - mean);
393         temp = temp->next;
394     }
395
396     skewness /= count;
397
398     std::cout << "Skewness of the list is " << skewness << std::endl;
399 }
400
401 // Create a function to find the kurtosis of a list
402 void findKurtosis(Node* head) {
403     Node* temp = head;
404     int sum = 0;
405     int count = 0;
406
407     while (temp != nullptr) {
408         sum += temp->data;
409         count++;
410         temp = temp->next;
411     }
412
413     double mean = sum / count;
414
415     double kurtosis = 0;
416
417     while (temp != nullptr) {
418         kurtosis += (temp->data - mean) * (temp->data - mean) * (temp->data - mean) * (temp->data - mean);
419         temp = temp->next;
420     }
421
422     kurtosis /= count;
423
424     std::cout << "Kurtosis of the list is " << kurtosis << std::endl;
425 }
426
427 // Create a function to find the entropy of a list
428 void findEntropy(Node* head) {
429     Node* temp = head;
430     int sum = 0;
431     int count = 0;
432
433     while (temp != nullptr) {
434         sum += temp->data;
435         count++;
436         temp = temp->next;
437     }
438
439     double mean = sum / count;
440
441     double entropy = 0;
442
443     while (temp != nullptr) {
444         entropy += (temp->data - mean) * (temp->data - mean);
445         temp = temp->next;
446     }
447
448     entropy /= count;
449
450     std::cout << "Entropy of the list is " << entropy << std::endl;
451 }
452
453 // Create a function to find the gini coefficient of a list
454 void findGiniCoefficient(Node* head) {
455     Node* temp = head;
456     int sum = 0;
457     int count = 0;
458
459     while (temp != nullptr) {
460         sum += temp->data;
461         count++;
462         temp = temp->next;
463     }
464
465     double mean = sum / count;
466
467     double giniCoefficient = 0;
468
469     while (temp != nullptr) {
470         giniCoefficient += (temp->data - mean) * (temp->data - mean);
471         temp = temp->next;
472     }
473
474     giniCoefficient /= count;
475
476     std::cout << "Gini Coefficient of the list is " << giniCoefficient << std::endl;
477 }
478
479 // Create a function to find the jensen's inequality of a list
480 void findJensensInequality(Node* head) {
481     Node* temp = head;
482     int sum = 0;
483     int count = 0;
484
485     while (temp != nullptr) {
486         sum += temp->data;
487         count++;
488         temp = temp->next;
489     }
490
491     double mean = sum / count;
492
493     double jensensInequality = 0;
494
495     while (temp != nullptr) {
496         jensensInequality += (temp->data - mean) * (temp->data - mean);
497         temp = temp->next;
498     }
499
500     jensensInequality /= count;
501
502     std::cout << "Jensen's Inequality of the list is " << jensensInequality << std::endl;
503 }
504
505 // Create a function to find the gini index of a list
506 void findGiniIndex(Node* head) {
507     Node* temp = head;
508     int sum = 0;
509     int count = 0;
510
511     while (temp != nullptr) {
512         sum += temp->data;
513         count++;
514         temp = temp->next;
515     }
516
517     double mean = sum / count;
518
519     double giniIndex = 0;
520
521     while (temp != nullptr) {
522         giniIndex += (temp->data - mean) * (temp->data - mean);
523         temp = temp->next;
524     }
525
526     giniIndex /= count;
527
528     std::cout << "Gini Index of the list is " << giniIndex << std::endl;
529 }
530
531 // Create a function to find the information gain of a list
532 void findInformationGain(Node* head) {
533     Node* temp = head;
534     int sum = 0;
535     int count = 0;
536
537     while (temp != nullptr) {
538         sum += temp->data;
539         count++;
540         temp = temp->next;
541     }
542
543     double mean = sum / count;
544
545     double informationGain = 0;
546
547     while (temp != nullptr) {
548         informationGain += (temp->data - mean) * (temp->data - mean);
549         temp = temp->next;
550     }
551
552     informationGain /= count;
553
554     std::cout << "Information Gain of the list is " << informationGain << std::endl;
555 }
556
557 // Create a function to find the gini diversity of a list
558 void findGiniDiversity(Node* head) {
559     Node* temp = head;
560     int sum = 0;
561     int count = 0;
562
563     while (temp != nullptr) {
564         sum += temp->data;
565         count++;
566         temp = temp->next;
567     }
568
569     double mean = sum / count;
570
571     double giniDiversity = 0;
572
573     while (temp != nullptr) {
574         giniDiversity += (temp->data - mean) * (temp->data - mean);
575         temp = temp->next;
576     }
577
578     giniDiversity /= count;
579
580     std::cout << "Gini Diversity of the list is " << giniDiversity << std::endl;
581 }
582
583 // Create a function to find the gini purity of a list
584 void findGiniPurity(Node* head) {
585     Node* temp = head;
586     int sum = 0;
587     int count = 0;
588
589     while (temp != nullptr) {
590         sum += temp->data;
591         count++;
592         temp = temp->next;
593     }
594
595     double mean = sum / count;
596
597     double giniPurity = 0;
598
599     while (temp != nullptr) {
600         giniPurity += (temp->data - mean) * (temp->data - mean);
601         temp = temp->next;
602     }
603
604     giniPurity /= count;
605
606     std::cout << "Gini Purity of the list is " << giniPurity << std::endl;
607 }
608
609 // Create a function to find the gini impurity of a list
610 void findGiniImpurity(Node* head) {
611     Node* temp = head;
612     int sum = 0;
613     int count = 0;
614
615     while (temp != nullptr) {
616         sum += temp->data;
617         count++;
618         temp = temp->next;
619     }
620
621     double mean = sum / count;
622
623     double giniImpurity = 0;
624
625     while (temp != nullptr) {
626         giniImpurity += (temp->data - mean) * (temp->data - mean);
627         temp = temp->next;
628     }
629
630     giniImpurity /= count;
631
632     std::cout << "Gini Impurity of the list is " << giniImpurity << std::endl;
633 }
634
635 // Create a function to find the gini diversity index of a list
636 void findGiniDiversityIndex(Node* head) {
637     Node* temp = head;
638     int sum = 0;
639     int count = 0;
640
641     while (temp != nullptr) {
642         sum += temp->data;
643         count++;
644         temp = temp->next;
645     }
646
647     double mean = sum / count;
648
649     double giniDiversityIndex = 0;
650
651     while (temp != nullptr) {
652         giniDiversityIndex += (temp->data - mean) * (temp->data - mean);
653         temp = temp->next;
654     }
655
656     giniDiversityIndex /= count;
657
658     std::cout << "Gini Diversity Index of the list is " << giniDiversityIndex << std::endl;
659 }
660
661 // Create a function to find the gini diversity ratio of a list
662 void findGiniDiversityRatio(Node* head) {
663     Node* temp = head;
664     int sum = 0;
665     int count = 0;
666
667     while (temp != nullptr) {
668         sum += temp->data;
669         count++;
670         temp = temp->next;
671     }
672
673     double mean = sum / count;
674
675     double giniDiversityRatio = 0;
676
677     while (temp != nullptr) {
678         giniDiversityRatio += (temp->data - mean) * (temp->data - mean);
679         temp = temp->next;
680     }
681
682     giniDiversityRatio /= count;
683
684     std::cout << "Gini Diversity Ratio of the list is " << giniDiversityRatio << std::endl;
685 }
686
687 // Create a function to find the gini diversity coefficient of a list
688 void findGiniDiversityCoefficient(Node* head) {
689     Node* temp = head;
690     int sum = 0;
691     int count = 0;
692
693     while (temp != nullptr) {
694         sum += temp->data;
695         count++;
696         temp = temp->next;
697     }
698
699     double mean = sum / count;
700
701     double giniDiversityCoefficient = 0;
702
703     while (temp != nullptr) {
704         giniDiversityCoefficient += (temp->data - mean) * (temp->data - mean);
705         temp = temp->next;
706     }
707
708     giniDiversityCoefficient /= count;
709
710     std::cout << "Gini Diversity Coefficient of the list is " << giniDiversityCoefficient << std::endl;
711 }
712
713 // Create a function to find the gini diversity measure of a list
714 void findGiniDiversityMeasure(Node* head) {
715     Node* temp = head;
716     int sum = 0;
717     int count = 0;
718
719     while (temp != nullptr) {
720         sum += temp->data;
721         count++;
722         temp = temp->next;
723     }
724
725     double mean = sum / count;
726
727     double giniDiversityMeasure = 0;
728
729     while (temp != nullptr) {
730         giniDiversityMeasure += (temp->data - mean) * (temp->data - mean);
731         temp = temp->next;
732     }
733
734     giniDiversityMeasure /= count;
735
736     std::cout << "Gini Diversity Measure of the list is " << giniDiversityMeasure << std::endl;
737 }
738
739 // Create a function to find the gini diversity ratio measure of a list
740 void findGiniDiversityRatioMeasure(Node* head) {
741     Node* temp = head;
742     int sum = 0;
743     int count = 0;
744
745     while (temp != nullptr) {
746         sum += temp->data;
747         count++;
748         temp = temp->next;
749     }
750
751     double mean = sum / count;
752
753     double giniDiversityRatioMeasure = 0;
754
755     while (temp != nullptr) {
756         giniDiversityRatioMeasure += (temp->data - mean) * (temp->data - mean);
757         temp = temp->next;
758     }
759
760     giniDiversityRatioMeasure /= count;
761
762     std::cout << "Gini Diversity Ratio Measure of the list is " << giniDiversityRatioMeasure << std::endl;
763 }
764
765 // Create a function to find the gini diversity coefficient measure of a list
766 void findGiniDiversityCoefficientMeasure(Node* head) {
767     Node* temp = head;
768     int sum = 0;
769     int count = 0;
770
771     while (temp != nullptr) {
772         sum += temp->data;
773         count++;
774         temp = temp->next;
775     }
776
777     double mean = sum / count;
778
779     double giniDiversityCoefficientMeasure = 0;
780
781     while (temp != nullptr) {
782         giniDiversityCoefficientMeasure += (temp->data - mean) * (temp->data - mean);
783         temp = temp->next;
784     }
785
786     giniDiversityCoefficientMeasure /= count;
787
788     std::cout << "Gini Diversity Coefficient Measure of the list is " << giniDiversityCoefficientMeasure << std::endl;
789 }
790
791 // Create a function to find the gini diversity measure coefficient of a list
792 void findGiniDiversityMeasureCoefficient(Node* head) {
793     Node* temp = head;
794     int sum = 0;
795     int count = 0;
796
797     while (temp != nullptr) {
798         sum += temp->data;
799         count++;
800         temp = temp->next;
801     }
802
803     double mean = sum / count;
804
805     double giniDiversityMeasureCoefficient = 0;
806
807     while (temp != nullptr) {
808         giniDiversityMeasureCoefficient += (temp->data - mean) * (temp->data - mean);
809         temp = temp->next;
810     }
811
812     giniDiversityMeasureCoefficient /= count;
813
814     std::cout << "Gini Diversity Measure Coefficient of the list is " << giniDiversityMeasureCoefficient << std::endl;
815 }
816
817 // Create a function to find the gini diversity ratio measure coefficient of a list
818 void findGiniDiversityRatioMeasureCoefficient(Node* head) {
819     Node* temp = head;
820     int sum = 0;
821     int count = 0;
822
823     while (temp != nullptr) {
824         sum += temp->data;
825         count++;
826         temp = temp->next;
827     }
828
829     double mean = sum / count;
830
831     double giniDiversityRatioMeasureCoefficient = 0;
832
833     while (temp != nullptr) {
834         giniDiversityRatioMeasureCoefficient += (temp->data - mean) * (temp->data - mean);
835         temp = temp->next;
836     }
837
838     giniDiversityRatioMeasureCoefficient /= count;
839
840     std::cout << "Gini Diversity Ratio Measure Coefficient of the list is " << giniDiversityRatioMeasureCoefficient << std::endl;
841 }
842
843 // Create a function to find the gini diversity coefficient ratio of a list
844 void findGiniDiversityCoefficientRatio(Node* head) {
845     Node* temp = head;
846     int sum = 0;
847     int count = 0;
848
849     while (temp != nullptr) {
850         sum += temp->data;
851         count++;
852         temp = temp->next;
853     }
854
855     double mean = sum / count;
856
857     double giniDiversityCoefficientRatio = 0;
858
859     while (temp != nullptr) {
860         giniDiversityCoefficientRatio += (temp->data - mean) * (temp->data - mean);
861         temp = temp->next;
862     }
863
864     giniDiversityCoefficientRatio /= count;
865
866     std::cout << "Gini Diversity Coefficient Ratio of the list is " << giniDiversityCoefficientRatio << std::endl;
867 }
868
869 // Create a function to find the gini diversity measure ratio of a list
870 void findGiniDiversityMeasureRatio(Node* head) {
871     Node* temp = head;
872     int sum = 0;
873     int count = 0;
874
875     while (temp != nullptr) {
876         sum += temp->data;
877         count++;
878         temp = temp->next;
879     }
880
881     double mean = sum / count;
882
883     double giniDiversityMeasureRatio = 0;
884
885     while (temp != nullptr) {
886         giniDiversityMeasureRatio += (temp->data - mean) * (temp->data - mean);
887         temp = temp->next;
888     }
889
890     giniDiversityMeasureRatio /= count;
891
892     std::cout << "Gini Diversity Measure Ratio of the list is " << giniDiversityMeasureRatio << std::endl;
893 }
894
895 // Create a function to find the gini diversity coefficient measure ratio of a list
896 void findGiniDiversityCoefficientMeasureRatio(Node* head) {
897     Node* temp = head;
898     int sum = 0;
899     int count = 0;
900
901     while (temp != nullptr) {
902         sum += temp->data;
903         count++;
904         temp = temp->next;
905     }
906
907     double mean = sum / count;
908
909     double giniDiversityCoefficientMeasureRatio = 0;
910
911     while (temp != nullptr) {
912         giniDiversityCoefficientMeasureRatio += (temp->data - mean) * (temp->data - mean);
913         temp = temp->next;
914     }
915
916     giniDiversityCoefficientMeasureRatio /= count;
917
918     std::cout << "Gini Diversity Coefficient Measure Ratio of the list is " << giniDiversityCoefficientMeasureRatio << std::endl;
919 }
920
921 // Create a function to find the gini diversity measure coefficient ratio of a list
922 void findGiniDiversityMeasureCoefficientRatio(Node* head) {
923     Node* temp = head;
924     int sum = 0;
925     int count = 0;
926
927     while (temp != nullptr) {
928         sum += temp->data;
929         count++;
930         temp = temp->next;
931     }
932
933     double mean = sum / count;
934
935     double giniDiversityMeasureCoefficientRatio = 0;
936
937     while (temp != nullptr) {
938         giniDiversityMeasureCoefficientRatio += (temp->data - mean) * (temp->data - mean);
939         temp = temp->next;
940     }
941
942     giniDiversityMeasureCoefficientRatio /= count;
943
944     std::cout << "Gini Diversity Measure Coefficient Ratio of the list is " << giniDiversityMeasureCoefficientRatio << std::endl;
945 }
946
947 // Create a function to find the gini diversity coefficient measure ratio coefficient of a list
948 void findGiniDiversityMeasureCoefficientRatioCoefficient(Node* head) {
949     Node* temp = head;
950     int sum = 0;
951     int count = 0;
952
953     while (temp != nullptr) {
954         sum += temp->data;
955         count++;
956         temp = temp->next;
957     }
958
959     double mean = sum / count;
960
961     double giniDiversityMeasureCoefficient
```

## Insertion at Head:

The screenshot shows a C++ IDE interface with two windows. The left window displays the source code for inserting at the head of a linked list. The right window shows the terminal output of the program, which prints "C P E" and "Process exited after 0.09976 seconds with return value 0".

```
#include <iostream>
using namespace std;
struct Node {
    char data;
    Node* prev;
    Node* next;
};
Node(char val, Node* p = nullptr, Node* n = nullptr)
{
    data = val;
    prev = p;
    next = n;
}
void insertAtHead(Node*& head, char value) {
    if (head == nullptr) {
        head = new Node(value);
        return;
    }
    Node* node = new Node(value);
    node->next = head;
    head->prev = node;
    head = node;
}
void traverse(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << endl;
        temp = temp->next;
    }
}
int main() {
    Node* head = new Node('C');
    insertAtHead(head, 'B');
    insertAtHead(head, 'A');
    traverse(head); // Output: C P E
    return 0;
}
```

## Insertion at the end:

The screenshot shows a C++ IDE interface with two windows. The left window displays the source code for inserting at the end of a linked list. The right window shows the terminal output of the program, which prints "C P E" and "Process exited after 0.09562 seconds with return value 0".

```
#include <iostream>
using namespace std;
struct Node {
    char data;
    Node* prev;
    Node* next;
};
Node(char val, Node* p = nullptr, Node* n = nullptr)
{
    data = val;
    prev = p;
    next = n;
}
void insertAtPosition(Node*& head, char value, int pos) {
    if (pos == 0) { // Insert at head
        insertAtHead(head, value);
        return;
    }
    Node* now = head;
    for (int i = 1; i < pos - 1; i++) {
        now = now->next;
    }
    if (now == nullptr) {
        return;
    }
    Node* temp = now->next;
    now->next = new Node(value);
    now->next->prev = now;
    now->next->next = temp;
    if (temp != nullptr) {
        temp->prev = now;
    }
    cout << endl;
}
void traverse(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << endl;
        temp = temp->next;
    }
}
int main() {
    Node* head = new Node('C');
    head->next = new Node('B', head, nullptr);
    insertAtPosition(head, 'A', 2);
    traverse(head); // Output: C P E
    return 0;
}
```

## Deletion of a node:

The screenshot shows a C++ IDE interface with two windows. The left window displays the source code for deleting a node from a linked list. The right window shows the terminal output of the program, which prints "C P E" and "Process exited after 0.1001 seconds with return value 0".

```
#include <iostream>
using namespace std;
struct Node {
    char data;
    Node* prev;
    Node* next;
};
Node(char val, Node* p = nullptr, Node* n = nullptr)
{
    data = val;
    prev = p;
    next = n;
}
void deleteNode(Node*& head, char value) {
    Node* temp = head;
    while (temp != nullptr && temp->data != value) {
        temp = temp->next;
    }
    if (temp == nullptr) {
        return;
    }
    if (temp->prev == nullptr) {
        temp->prev->next = temp->next;
    } else {
        head = temp->next;
    }
    if (temp->next != nullptr) {
        temp->next->prev = temp->prev;
    }
    delete temp;
}
void traverse(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << endl;
        temp = temp->next;
    }
}
int main() {
    Node* head = new Node('C');
    head->next = new Node('B', head, nullptr);
    deleteNode(head, 'B');
    traverse(head); // Output: C E
    return 0;
}
```

## Source Code:

The screenshot shows a C++ IDE interface with a code editor and a terminal window. The code editor contains a file named `Linked List.cpp` with the following content:

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7
8     Node(char val, Node* n = nullptr) : data(val), next(n) {}
9 };
10
11 void traverse(Node* head) {
12     Node* temp = head;
13     while (temp != nullptr) {
14         cout << temp->data << " ";
15         temp = temp->next;
16     }
17     cout << endl;
18 }
19
20 int main() {
21     // Creating the list manually: C -> P -> E
22     Node* head = new Node('C');
23     head->next = new Node('P');
24     head->next->next = new Node('E');
25
26     traverse(head); // Output: C P E
27
28     return 0;
29 }
30
```

The terminal window shows the output of the program: `C P E`. Below the terminal window, a message says: `Process exited after 0.09997 seconds with return value 0 Press any key to continue . . .`

## Source Code:

The screenshot shows a C++ IDE interface with a code editor and a terminal window. The code editor contains a file named `Linked List.cpp` with the following content:

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7
8     Node(char val, Node* n = nullptr) : data(val), next(n) {}
9 };
10
11 void insertAtHead(Node*& head, char value) {
12     Node* newNode = new Node(value, head);
13     head = newNode;
14 }
15
16 void traverse(Node* head) {
17     Node* temp = head;
18     while (temp) {
19         cout << temp->data << " ";
20         temp = temp->next;
21     }
22     cout << endl;
23 }
24
25 int main() {
26     // Creating initial list: P -> E
27     Node* head = new Node('P');
28     head->next = new Node('E');
29
30     // Inserting at head: C -> P -> E
31     insertAtHead(head, 'C');
32
33     traverse(head); // Output: C P E
34
35     return 0;
36 }
```

The terminal window shows the output of the program: `C P E`. Below the terminal window, a message says: `Process exited after 0.09773 seconds with return value 0 Press any key to continue . . .`

## Source Code:

The screenshot shows a C++ IDE interface with a code editor and a terminal window. The code editor contains a file named `Linked List.cpp` with the following content:

```
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7
8     Node(char val, Node* n = nullptr) : data(val), next(n) {}
9 };
10
11 void insertAtEnd(Node*& head, char value) {
12     Node* newNode = new Node(value);
13
14     if (head == nullptr)
15         head = newNode;
16     else {
17         Node* temp = head;
18         while (temp->next != nullptr)
19             temp = temp->next;
20
21         temp->next = newNode;
22     }
23 }
24
25 void traverse(Node* head) {
26     Node* temp = head;
27     while (temp != nullptr) {
28         cout << temp->data << " ";
29         temp = temp->next;
30     }
31     cout << endl;
32 }
33
34 int main() {
35     Node* head = new Node('C');
36     head->next = new Node('P');
37
38     insertAtEnd(head, 'E');
39
40     traverse(head);
41
42     return 0;
43 }
```

The terminal window shows the output of the program: `C P E`. Below the terminal window, a message says: `Process exited after 0.1015 seconds with return value 0 Press any key to continue . . .`

## Source Code:

The screenshot shows the Embarcadero Dev-C++ IDE interface. The project is named "Linked List". The code in `Linked List.cpp` defines a `Node` structure, a `deleteNode` function, a `traverse` function, and a `main` function. The `main` function creates a list with nodes 'C', 'P', and 'E', then deletes the node with value 'P' and traverses the remaining list.

```

3 struct Node {
4     char data;
5     Node* next;
6 }
7 Node(char val, Node* n = nullptr) : data(val), next(n) {}
8
9 void deleteNode(Node*& head, char value) {
10    Node* temp = head;
11    Node* prev = nullptr;
12
13    while (temp != nullptr && temp->data != value) {
14        prev = temp;
15        temp = temp->next;
16    }
17
18    if (temp == nullptr) return;
19    if (prev == nullptr) {
20        head = temp->next;
21    } else {
22        prev->next = temp->next;
23    }
24
25    delete temp;
26}
27
28 void traverse(Node* head) {
29    Node* temp = head;
30    while (temp != nullptr) {
31        cout << temp->data << " ";
32        temp = temp->next;
33    }
34    cout << endl;
35}
36
37 int main() {
38    Node* head = new Node('C');
39    head->next = new Node('P');
40    head->next->next = new Node('E');
41
42    deleteNode(head, 'P');
43
44    traverse(head);
45
46    return 0;
47}
48
49
50

```

The console window shows the output: "C E", followed by a message: "Process exited after 0.09739 seconds with return value 0 Press any key to continue . . .".

Table 3-3. Code and Analysis for Singly Linked Lists

The screenshot shows the Embarcadero Dev-C++ IDE interface. The project is named "Linked List". The code in `Linked List.cpp` defines a `Node` structure, a `traverse` function, an `insertAtEnd` function, and a `main` function. The `main` function inserts nodes 'C', 'P', 'E', 'D', 'I', and 'O' at the end of the list and then traverses it.

```

1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 }
8 Node(char d, Node* n = nullptr) : data(d), next(n) {}
9
10 void traverse(Node* head) {
11    while (head) {
12        cout << head->data;
13        head = head->next;
14    }
15    cout << endl;
16}
17
18 void insertAtEnd(Node*& head, char v) {
19    Node* n = new Node(v);
20
21    if (head == nullptr) {
22        head = n;
23        return;
24    }
25
26    Node* t = head;
27    while (t->next)
28        t = t->next;
29
30    t->next = n;
31}
32
33 int main() {
34    Node* head = nullptr;
35
36    insertAtEnd(head, 'C');
37    insertAtEnd(head, 'P');
38    insertAtEnd(head, 'E');
39    insertAtEnd(head, 'D');
40    insertAtEnd(head, 'I');
41    insertAtEnd(head, 'O');
42
43    cout << "Initial list: ";
44    traverse(head);
45
46    return 0;
47}
48
49
50

```

The console window shows the initial list: "CPE010", followed by a message: "Process exited after 0.09641 seconds with return value 0 Press any key to continue . . .".

## Analysis:

Traversing the list by making the head pointer pass. Function walks from node to node and prints the stored characters.

The screenshot shows the Embarcadero Dev-C++ IDE interface. The project is named "Linked List". The code in `Linked List.cpp` defines a `Node` structure, a `traverse` function, an `insertAtHead` function, and a `main` function. The `main` function inserts nodes 'G', 'C', 'P', 'E', 'D', 'I', and 'O' at the head of the list and then traverses it.

```

1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7 }
8 Node(char d, Node* n = nullptr) : data(d), next(n) {}
9
10 void traverse(Node* head) {
11    while (head) {
12        cout << head->data;
13        head = head->next;
14    }
15    cout << endl;
16}
17
18 void insertAtHead(Node*& head, char v) {
19    Node* n = new Node(v);
20
21    if (head) {
22        n->next = head;
23        head = n;
24    }
25
26    Node* t = head;
27    while (t->next)
28        t = t->next;
29
30    t->next = n;
31}
32
33 void insertAtHead(Node*& head, char v) {
34    Node* n = new Node(v);
35
36    insertAtHead(head, 'G');
37    insertAtHead(head, 'C');
38    insertAtHead(head, 'P');
39    insertAtHead(head, 'E');
40    insertAtHead(head, 'D');
41    insertAtHead(head, 'I');
42    insertAtHead(head, 'O');
43
44    cout << "After inserting 'G' at head: GCPEDI";
45    traverse(head);
46
47    return;
48}
49
50

```

The console window shows the output: "After inserting 'G' at head: GCPEDI", followed by a message: "Process exited after 0.10116 seconds with return value 0 Press any key to continue . . .".

## Analysis:

creates a new node wherein next points to the previous head, then reassigns head.

Insert a head

```

1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7     Node(char d, Node* n = nullptr) : data(d), next(n) {}
8 }
9
10 void traverse(Node* head) {
11     while (head) {
12         cout << head->data;
13         head = head->next;
14     }
15     cout << endl;
16 }
17
18 void insertAtEnd(Node*& head, char v) {
19     Node* n = new Node(v);
20
21     if (!head) {
22         head = n;
23         return;
24     }
25
26     Node* t = head;
27     while (t->next) {
28         t = t->next;
29     }
30     t->next = n;
31 }
32
33 void insertAtHead(Node*& head, char v) {
34     Node* n = new Node(v, head);
35     head = n;
36 }
37
38 void insertAfter(Node* prev, char v) {
39     if (!prev) return;
40     Node* n = new Node(v, prev->next);
41     prev->next = n;
42 }
43
44 int main() {
45     Node* head = nullptr;
46     insertAtEnd(head, 'C');
47     insertAtEnd(head, 'D');
48     insertAtEnd(head, 'E');

```

**Analysis:** It finds node "P", allocate new node and adjust the next pointers.

```

1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7     Node(char d, Node* n = nullptr) : data(d), next(n) {}
8 }
9
10 void traverse(Node* head) {
11     while (head) {
12         cout << head->data;
13         head = head->next;
14     }
15     cout << endl;
16 }
17
18 void insertAtEnd(Node*& head, char v) {
19     Node* n = new Node(v);
20
21     if (head) {
22         head = n;
23         return;
24     }
25
26     Node* t = head;
27     while (t->next) {
28         t = t->next;
29     }
30     t->next = n;
31 }
32
33 void insertAtHead(Node*& head, char v) {
34     Node* n = new Node(v, head);
35     head = n;
36 }
37
38 void insertAfter(Node* prev, char v) {
39     if (!prev) return;
40     Node* n = new Node(v, prev->next);
41     prev->next = n;
42 }
43
44 void deleteNode(Node*& head, char key) {
45     if (!head) return;
46
47     if (head->data == key) {
48         Node* temp = head;
49         head = head->next;
50         delete temp;
51     }
52 }
53
54 int main() {
55     Node* head = nullptr;
56     insertAtEnd(head, 'C');
57     insertAtEnd(head, 'D');
58     insertAtEnd(head, 'E');
59     insertAtEnd(head, 'P');
60
61     deleteNode(head, 'P');
62
63     traverse(head);
64 }

```

**Analysis:** Delete a node by locating the node before the desired target. Changing its next to skip the target node.

```

1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* next;
7     Node(char d, Node* n = nullptr) : data(d), next(n) {}
8 }
9
10 void traverse(Node* head) {
11     while (head) {
12         cout << head->data;
13         head = head->next;
14     }
15     cout << endl;
16 }
17
18 void insertAtEnd(Node*& head, char v) {
19     Node* n = new Node(v, nullptr);
20
21     if (!head) {
22         head = n;
23         return;
24     }
25
26     Node* t = head;
27     while (t->next) {
28         t = t->next;
29     }
30     t->next = n;
31 }
32
33 void insertAtHead(Node*& head, char v) {
34     Node* n = new Node(v, head);
35     head = n;
36 }
37
38 void insertAfter(Node* prev, char v) {
39     if (!prev) return;
40     Node* n = new Node(v, prev->next);
41     prev->next = n;
42 }
43
44 void deleteNode(Node*& head, char key) {
45     if (!head) return;
46
47     if (head->data == key) {
48         Node* temp = head;
49         head = head->next;
50         delete temp;
51     }
52 }
53
54 int main() {
55     Node* head = nullptr;
56     insertAtEnd(head, 'C');
57     insertAtEnd(head, 'D');
58     insertAtEnd(head, 'E');
59     insertAtEnd(head, 'P');
60
61     deleteNode(head, 'P');
62
63     traverse(head);
64 }

```

**Analysis:** Applying deletion again to remove "P".

```

1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* prev;
7     Node* next;
8 };
9
10 void traverse(Node* head) {
11     while (head) {
12         cout << head->data;
13         head = head->next;
14     }
15     cout << endl;
16 }
17
18 int main() {
19     Node* head = nullptr;
20
21     head = new Node();
22     head->data = 'G';
23
24     head->next = new Node();
25     head->next->data = 'E';
26
27     head->next->next = new Node();
28     head->next->next->data = 'E';
29
30     head->next->next->next = new Node();
31     head->next->next->next->data = 'I';
32
33     head->next->next->next->next = new Node();
34     head->next->next->next->next->data = 'I';
35
36     head->next->next->next->next->next = new Node();
37     head->next->next->next->next->next->data = 'I';
38
39     head->next->next->next->next->next->next = new Node();
40     head->next->next->next->next->next->next->data = 'I';
41
42     head->next->next->next->next->next->next->next->next = nullptr;
43
44     cout << "Final list: ";
45     traverse(head);
46
47     return 0;
48 }

```

**Analysis:** Displaying the result after all operations.

**Table 3-4. Modified Operations for Doubly Linked Lists**

```

1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     char data;
6     Node* prev;
7     Node* next;
8 };
9
10 void traverse(Node* head) {
11     while (head) {
12         cout << head->data;
13         head = head->next;
14     }
15     cout << endl;
16 }
17
18 void insertAtEnd(Node*& head, char v) {
19     Node* n = new Node();
20     n->data = v;
21     n->next = nullptr;
22     n->prev = nullptr;
23     if (!head) {
24         head = n;
25         return;
26     }
27     Node* t = head;
28     while (t->next) {
29         t = t->next;
30     }
31     t->next = n;
32     n->prev = t;
33 }
34
35 void insertAtBeginning(Node*& head, char v) {
36     Node* n = new Node();
37     n->next = nullptr;
38     n->prev = head;
39     if (head) head->prev = n;
40     head = n;
41     n->data = v;
42 }
43
44 void insertAfter(Node* head, char prevData, char v) {

```

## 7. Supplementary Activity:

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 struct Song {
6     string title;
7     Song* next;
8     Song* prev;
9 };
10
11 void addSong(Song*& head, const string& title) {
12     Song* newSong = new Song(title, nullptr, nullptr);
13     if (!head) {
14         head = newSong;
15         head->next = head;
16         head->prev = head;
17         return;
18     }
19     Song* tail = head->prev;
20     tail->next = newSong;
21     newSong->prev = tail;
22     newSong->next = head;
23     head->prev = newSong;
24 }
25
26 void removeSong(Song*& head, const string& title) {
27     if (!head) return;
28     Song* curr = head;
29     do {
30         if (curr->title == title) {
31             if (curr->next == curr) {
32                 delete curr;
33                 head = nullptr;
34                 return;
35             }
36             curr->prev->next = curr->next;
37             curr->next->prev = curr->prev;
38             if (curr == head) head = curr->next;
39             delete curr;
40             return;
41         }
42         curr = curr->next;
43     } while (curr != head);
44 }
45
46 void playAll(Song* head) {
47     if (!head) {
48         cout << "Playlist is empty.\n";
49         return;
50     }
51     Song* curr = head;
52     do {
53         cout << "Playing: " << curr->title << endl;
54         curr = curr->next;
55     } while (curr != head);
56 }
57
58 Song* nextSong(Song* curr) {
59     if (!curr) return nullptr;
60     return curr->next;
61 }
62
63 Song* prevSong(Song* curr) {
64     if (!curr) return nullptr;
65     return curr->prev;
66 }
67
68 int main() {
69     Song* playlist = nullptr;
70     addSong(playlist, "Song A");
71     addSong(playlist, "Song B");
72     addSong(playlist, "Song C");
73     addSong(playlist, "Song D");
74     addSong(playlist, "Song E");
75
76     cout << "\nInitial Playlist:\n";
77     playAll(playlist);
78
79     cout << "\nRemoving Song B...\n";
80     removeSong(playlist, "Song B");
81     playAll(playlist);
82
83     Song* current = playlist;
84     cout << "\nCurrently playing: " << current->title << endl;
85     current = nextSong(current);
86     cout << "Next song: " << current->title << endl;
87     current = prevSong(current);
88     cout << "Previous song: " << current->title << endl;
89
90     return 0;
91 }
92
93

```

Initial Playlist:

Playing: Song A  
Playing: Song B  
Playing: Song C  
Playing: Song D  
Playing: Song E

Removing Song B...

Playing: Song A  
Playing: Song C  
Playing: Song D  
Playing: Song E

Currently playing: Song A  
Next song: Song C  
Previous song: Song A

-----

Process exited after 0.1066 seconds with return value 0  
Press any key to continue . . .

**8. Conclusion:** In this activity 3.1 Linked Lists, I understand how to revise the single linked lists with connection to doubly linked lists, which gave us an idea how did pointers work. We need to be careful in moving to the next and previous pointers. In general I need to study and learn more to improve my skills to organize my codes just to look clean and neat so it that it can understand it easily.

## 9. Assessment Rubric