Data structure – it is a concept of set of algorithms used to structure the information

**STACK code**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*STACK implementation using static array*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*#include* <stdio.h>

*#include*<stdlib.h>

*#define* CAPACITY 5 *// THIS IS A MACRO(A CONSTANT)*

*//we define macro using pre-processor directives*

*//Pre-processing happens before compilation*

int stack[CAPACITY];

int top = -1;*// because top location is not pointing to any index at start*

int isFull(){ *// check if full or not*

if(top==CAPACITY-1){

return 1;

}

return 0;

}

int isEmpty(){ *// check if empty or not*

if(top==(-1)){

return 1;

}

return 0;

}

void Push(){ *// insert an element*

if(isFull()){

printf("Stack is full\n");

}

else{int element;

printf("enter the element\n");

scanf("%d",&element);

top++;

stack[top]=element;

printf("%d inserted\n",element);

}

}

int Pop(){ *// remove an element and return it*

if(isEmpty()){

printf("Stack is empty\n");

return 0;

}

else{

printf("popped -- %d\n",stack[top]);

top--;

}

}

int Peek(){ *// do not remove just display the top element*

if(isEmpty()){

printf("Stack is empty\n");

return 0;

}

else{

return stack[top];

}

}

void Traverse(){

if(isEmpty()){

printf("Stack is empty\n");

}

else{

for (int i = top; i >=0; i--) {

printf("%d\n",stack[i]);

}

}

}

int main()

{

int choice;

while(1){

printf("\*\*\*\*\n");

printf("1.Push\n");

printf("2.Pop\n");

printf("3.Peek\n");

printf("4.Traverse\n");

printf("5.Quit\n");

printf("Please enter your choice\n");

scanf("%d",&choice);

printf("\*\*\*\*\n");

switch(choice){

case 1 : Push();break;

case 2 : Pop();break;

case 3 : Peek();break;

case 4 : Traverse();break;

case 5 : exit(0);break;

default: printf("Invalid input \n\n");break;

}

}

}

Linked List

Appending into the list

1. struct node{
2. int data;
3. struct node \*link;
4. };
5. struct node \*root=NULL;*//Initially pointing to null*


9. void append(){ ***// insert at last or Append***
10. struct node \*temp;
11. temp = (struct node\*)malloc(sizeof(struct node));
12. printf("Enter the value \n");
13. scanf("%d",&temp->data);
14. *//since pointer is pointing to structure we use -> arrow operator*
15. temp->link=NULL;
16. if(root == NULL){
17. root=temp;
18. }
19. else{
20. struct node \*p;
21. p = root;
22. while(p->link!=NULL){
23. p=p->link;
24. }
25. p->link=temp;
26. }
27. }

Finding Length of the Linked list

1. *// Finding length of linked list*
2. int length(){
3. int cnt=0;
4. struct node \*temp;
5. temp = root;
6. while(temp!=NULL){ *// we can also use temp->link but then we will have one count less*
7. temp=temp->link;
8. cnt++;
9. }
10. return cnt;
11. }

Add node at the beginning of the Linked list

1. *//Add node at the beginning of the list*
2. void addAtBegin(){
3. struct node \*temp;
4. temp = (struct node\*)malloc(sizeof(struct node));
5. printf("Enter the value \n");
6. scanf("%d",&temp->data);
7. *//since pointer is pointing to structure we use -> arrow operator*
8. temp->link=NULL;
9. if(root==NULL){
10. root=temp;
11. }
12. else{
13. temp->link = root;
14. root = temp;
15. }
17. }

Delete from the Linked list

1. void delete(){ // Delete from the linked list
2. struct node \*temp;
3. int loc;
4. printf("Please enter which node you want to delete \n"); scanf("%d",&loc);
5. if(loc>length()){
6. printf("Invalid Location\n")
7. }
8. else if(loc ==1){ *// if we want to delete first node*
9. temp=root;
10. root = temp->link; *// connect root to next node*
12. temp->link=NULL; *// remove connection of first and second node*
13. free(temp); *// free the first node*
14. }
15. else{
16. struct node \*p=root,q;
17. while(i<loc-1){
18. p=p->link;
19. i++;
20. }
21. q=p->link;
22. p->link=q->link;
23. q->link=NULL;
24. free(q);
25. }
26. }

Add after a certain node

1. void addAfter(){ // adds after a certain node
2. struct node \*temp,\*p;
3. int loc,len,i=1;
4. printf("Enter the Location after you want to insert\n");
5. scanf("%d",&loc);
6. len = length();
7. if(loc>len){
8. printf(" Invalid location \n");
9. printf(" The curret nodes are %d\n",loc);
10. }
11. else{
12. p = root;
13. while(i<loc){
14. p=p->link;
15. i++;
16. }
17. temp = (struct node\*)malloc(sizeof(struct node));
18. printf("Enter the value \n");
19. scanf("%d",&temp->data);
20. *//since pointer is pointing to structure we use -> arrow operator*
21. temp->link=p->link;
22. p->link = temp;
23. }
24. }

Reverse Linked list

1. void reverse(){

2. struct node \*current=head,\*next=head,\*previous=NULL;

3. while(current!=NULL){

4. next=next->link;

5. current->link=previous;

6. previous=current;

7. current=next;

8. }head=previous;

9. }

10.

1. Full Linked list Programme  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
2. *appending, finding length, values to the list*
3. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*
5. *#include* <stdio.h>
6. *#include* <stdlib.h>
8. struct node{
9. int data;
10. struct node \*link;
11. };
12. struct node \*root=NULL;*//Initially pointing to null*


16. void append(){ *// insert at last*
17. struct node \*temp;
18. temp = (struct node\*)malloc(sizeof(struct node));
19. printf("Enter the value \n");
20. scanf("%d",&temp->data);
21. *//since pointer is pointing to structure we use -> arrow operator*
22. temp->link=NULL;
23. if(root == NULL){
24. root=temp;
25. }
26. else{
27. struct node \*p;
28. p = root;
29. while(p->link!=NULL){
30. p=p->link;
31. }
32. p->link=temp;
33. }
34. }
36. *// Finding length of linked list*
37. int length(){
38. int cnt=0;
39. struct node \*temp;
40. temp = root;
41. while(temp!=NULL){ *// we can also use temp->link but then we will have one count less*
42. temp=temp->link;
43. cnt++;
44. }
45. return cnt;
46. }
48. *//Add node at the beginning of the list*
49. void addAtBegin(){
50. struct node \*temp;
51. temp = (struct node\*)malloc(sizeof(struct node));
52. printf("Enter the value \n");
53. scanf("%d",&temp->data);
54. *//since pointer is pointing to structure we use -> arrow operator*
55. temp->link=NULL;
56. if(root==NULL){
57. root=temp;
58. }
59. else{
60. temp->link = root;
61. root = temp;
62. }
64. }
66. void delete(){
67. struct node \*temp;
68. int loc;
69. printf("Please enter which node you want to delete \n");
70. scanf("%d",&loc);
71. if(loc>length()){
72. printf("Invalid Location\n");
73. }
74. else if(loc ==1){ *// if we want to delete first node*
75. temp=root;
76. root = temp->link; *// connect root to next node*
78. temp->link=NULL; *// remove connection of first and second node*
79. free(temp); *// free the first node*
80. }
81. else{
82. int i=1;
83. struct node \*p=root,\*q;
84. while(i<loc-1){
85. p=p->link;
86. i++;
87. }
88. q=p->link;
89. p->link=q->link;
90. q->link=NULL;
91. free(q);
92. }
93. }
95. void addAfter(){
96. struct node \*temp,\*p;
97. int loc,len,i=1;
98. printf("Enter the Location after you want to insert\n");
99. scanf("%d",&loc);
100. len = length();
101. if(loc>len){
102. printf(" Invalid location \n");
103. printf(" The curret nodes are %d\n",loc);
104. }
105. else{
106. p = root;
107. while(i<loc){
108. p=p->link;
109. i++;
110. }
111. temp = (struct node\*)malloc(sizeof(struct node));
112. printf("Enter the value \n");
113. scanf("%d",&temp->data);
114. *//since pointer is pointing to structure we use -> arrow operator*
115. temp->link=p->link;
116. p->link = temp;
117. }
118. }
120. void display(){ *// print/traverse through*
121. struct node \*temp;
122. temp=root;
123. for (int i = 0; i < length(); i++) {
124. printf("%d",temp->data);
125. temp=temp->link;
126. }
127. }
129. int main()
130. { append();
131. append();
132. append();
133. addAtBegin();
134. delete();
135. addAfter();
136. display();
137. return 0;
138. }

**Linear Search**

1. *#include* <iostream>

2.

3. using namespace std;

4. char\* linearSearch(int arr[],int size, int element){

5. for(int i=0;i<size;i++){

6. if(element==arr[i])

7. return "Found";

8. }

9. return "Not Found";

10.

11. }

12.

13. int main()

14. { int array[]={3,4,5,2,4,6,7,4,1,134,6,8,8654};

15. cout<<linearSearch(array,sizeof(array)/4,1);

16.

17. return 0;

18. }

19.

**Binary Search**

* 1. **Normal Function**
  2. **Recursion**

**i)**

1. *#include* <iostream>

2. using namespace std;

3. char\* binarySearch(int arr[],int size, int element){

4. int min=0,max=(size-1),mid=0;

5. while(min<=max){

6. mid= (min+max)/2;

7. if(arr[mid]==element){

8. return "Found";

9. }

10. else if(element > arr[mid])

11. min=mid;

12. else if (element > arr[mid])

13. max=mid;

14. else return "Not Found";

15. }

16. }

17. int main()

18. { int array[]={1,3,5,7,9,11,13,15,17,19};

19. cout<<binarySearch(array,sizeof(array)/4,2);

20.

21. return 0;

22. }

**ii) Recursion**

1. *#include* <iostream>

2. using namespace std;

3. *#define* searchElement 7

4. char\* binarySearch(int arr[],int size, int element,int min, int max){

5. int mid= (min+max)/2;

6.

7. if(arr[mid]==element)

8. return "Found";

9.

10. else if(element > arr[mid])

11. return binarySearch(arr,size,searchElement,mid,max);

12.

13. else if (element < arr[mid])

14. return binarySearch(arr,size,searchElement,min,mid);

15.

16. else return "Not Found";

17.

18. }

19. int main()

20. { int array[]={1,3,5,7,9,11,13,15,17,19};

21. int min =0;

22. int max = (sizeof(array)/4 - 1);

23. cout<<binarySearch(array,sizeof(array)/4,searchElement,min,max);

24.

25. return 0;

26. }

27.

|  |  |  |  |
| --- | --- | --- | --- |
| SORTING Techniques | Complexity | STABLE(Keep the order of ranking same) | ADAPTIVE (Take advantage of already sorted array) |
| 1. Bubble Sort | O(n^2) |  |  |
| 1. Selection Sort | NO | NO |
| 1. Insertion Sort |  |  |
|  |  |  |  |
| 1. Quick Sort | O(nlogn) |  |  |
| 1. Merg Sort |  |  |
| 1. Radix Sort |  |  |
| 1. Heap Sort |  |  |
|  |  |  |  |
| 1. Count Sort | O(n) |  |  |

1. Bubble sort

1. *#include* <bits/stdc++.h>

2. using namespace std;

3. void swap(int \*a,int \*b){

4. int temp = \*a;

5. \*a = \*b;

6. \*b = temp;

7. }

8. void bubbleSort(int arr[],int size){

9. for(int i=0;i<size;i++){

10. for(int j=0;j<size-i-1;j++)

11. if(arr[j]>arr[j+1]) swap(arr[j],arr[j+1]);

12. }

13. }

14. int main()

15. {

16. int k[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };

17. bubbleSort(k,10);

18. for(int i=0;i<10;i++)

19. cout << k[i] << " ";

20.

21. return 0;

22. }

2. Selection sort

1. *#include* <bits/stdc++.h>

2. using namespace std;

3. void swap(int \*a,int \*b){

4. int temp = \*a;

5. \*a = \*b;

6. \*b = temp;

7. }

8. void selctionSort(int arr[],int size){

9. for(int i=0;i<size;i++){ // Here i acts as sorted till flag

10. for(int j=(i+1);j<size;j++){

11. if(arr[j]<arr[i]) {swap(arr[j],arr[i]);}

12. }

13. }

14. }

15. int main()

16. {

17. int k[] = { 1, 5, 8, 9, 6, 7, 4,3 , 2, 0 };

18. selctionSort(k,10);

19. for(int i=0;i<10;i++)

20. cout << k[i] << " ";

21.

22. return 0;

23. }

24.

3. Insertion sort

1. *#include* <bits/stdc++.h>

2. using namespace std;

3. void swap(int \*a,int \*b){

4. int temp = \*a;

5. \*a = \*b;

6. \*b = temp;

7. }

8. void insertionSort(int arr[],int size){

9. for(int i=0;i<=size;i++){ // Here i acts as sorted till flag

10. for(int j=(i+1);j>0;j--){

11. if(arr[j]<arr[j-1]) {swap(arr[j],arr[j-1]);}

12. }

13. }

14. }

15. int main()

16. {

17. int k[] = { 7,12,3,4,1 };

18. insertionSort(k,5);

19. for(int i=0;i<5;i++)

20. cout << k[i] << " ";

21.

22. return 0;

23. }

24.

4. Quick Sort

1. *#include*<stdio.h>

2. void swap(int arr[],int i,int j)

3. {

4. int temp;

5. temp=arr[i];

6. arr[i]=arr[j];

7. arr[j]=temp;

8. }

9.

10. int partition(int arr[],int lb,int ub)

11. {

12. int pivot=arr[lb];

13. int start=lb;

14. int end=ub;

15. while(end>start)

16. {

17. while(!(arr[start]>pivot))

18. start++;

19. while(!(arr[end]<=pivot))

20. end--;

21. if(end>start)

22. swap(arr,start,end);

23.

24. swap(arr,lb,end);*// swap pivot with end*

25. return end;

26. }

27. return 0;

28. }

29.

30. int quicksort(int arr[],int lb,int ub)

31. { if(ub>lb)

32. {

33. int loc=partition(arr,lb,ub);

34. quicksort(arr,lb,loc-1);

35. quicksort(arr,loc+1,ub);

36. }

37. return 0;

38. }

39. int main()

40. {

41. int arr[]={10,8,21,24,55,36,72};

42. int lb=0,ub=(sizeof(arr)/sizeof(arr[0]))-1;

43. quicksort(arr,lb,ub);

44. for(int i=0;i<ub+1;i++)

45. printf("%d ",arr[i]);

46. }

47.

5. Count Sort

1. *#include*<stdio.h>

2.

3. int main()

4. {

5. int arr[]={2,1,1,0,2,5,4,0,2,8,7,7,9,2,0,1,9},temp,j,k=10;

6.

7. int n=sizeof(arr)/sizeof(arr[0]);

8. int count[10];

9. int b[n];

10. for(int i=0;i<n;i++)

11. {

12. ++count[arr[i]];

13. }

14. for(int i=1;i<k;i++)

15. {

16. count[i]=count[i-1]+count[i];

17. }

18. for(int i=n-1;i>0;i--)

19. {

20. b[--count[arr[i]]]=arr[i];

21. }

22. for(int i=0;i<n;i++)

23. {

24. printf("%d",b[i]);

25. }

26. }

27.

6.Merg sort

//Merge sort

1. *#include*<stdlib.h>

2. *#include*<stdio.h>

3.

4. void merge(int arr[],int lb, int mid, int ub)

5. {

6. int i,j,k;

7. int n1=mid-lb+1;

8. int n2=ub-mid;

9.

10. int L[n1],R[n2];

11. for(i=0;i<n1;i++)

12. {

13. L[i]=arr[lb+i];

14. }

15. for(j=0;j<n2;j++)

16. {

17. R[j]=arr[mid+1+j];

18. }

19. i = 0;

20. j = 0;

21. k = lb;

22. while (i < n1 && j < n2) {

23. if (L[i] <= R[j]) {

24. arr[k] = L[i];

25. i++;

26. }

27. else {

28. arr[k] = R[j];

29. j++;

30. }

31. k++;

32. }

33.

34. while (i < n1) {

35. arr[k] = L[i];

36. i++;

37. k++;

38. }

39.

40. while (j < n2) {

41. arr[k] = R[j];

42. j++;

43. k++;

44. }

45.

46. }

47. void mergearray(int arr[],int lb,int ub)

48. {

49. int mid;

50. if(lb<ub)

51. {

52. mid=(lb+ub)/2;

53. mergearray(arr,lb,mid);

54. mergearray(arr,mid+1,ub);

55. merge(arr,lb,mid,ub);

56. }

57. }

58. void printArray(int A[], int size)

59. {

60. int i;

61. for (i = 0; i < size; i++)

62. printf("%d ", A[i]);

63. printf("\n");

64. }

65.

66. int main()

67. {

68. int arr[]={0,1,0,1,2,4};

69. int left,right,mid;

70. int arrsize=sizeof(arr)/sizeof(arr[0]);

71. mergearray(arr,0,arrsize-1);

72. printArray(arr, arrsize);

73. return 0;

74. }

7.Radix sort

1. *#include*<stdio.h>

2.

3. int countsort(int arr[],int n, int pos)

4. {

5. int count[10]={0},k=10;

6. int b[10];

7. for(int i=0;i<n;i++)

8. {

9. ++count[(arr[i]/pos)%10];

10.

11. }

12. for(int i=1;i<k;i++)

13. {

14. count[i]=count[i-1]+count[i];

15. }

16. for(int i=n-1;i>=0;i--)

17. {

18. b[--count[(arr[i]/pos)%10]]=arr[i];

19. }

20. for(int i=0;i<n;i++)

21. {

22. arr[i]=b[i];

23. }

24. }

25.

26. int main()

27. {

28. int arr[]={432,8,530,90,88,231,11,45,677,199},temp,j,max;

29. int pos;

30. int n=sizeof(arr)/sizeof(arr[0]);

31. max=arr[0];

32. for(int i=1;i<n;i++)

33. {

34. if(max<arr[i])

35.

36. max=arr[i];

37.

38. }

39. *//printf("%d",max);*

40. for(pos=1;max/pos>0;pos\*=10){

41. countsort(arr,n,pos);}

42.

43. for(int i=0;i<n;i++)

44. {

45. printf("%d\n",arr[i]);

46. }

47.

48. }

49.

Queue

1. *#include* <iostream>

2. *#define* capacity 5

3.

4. using namespace std;

5. int front=0,rear=0;//Deletion from front, Insertion from rear

6. int queue[capacity];

7.

8. void Traverse(){

9. if(front==rear) printf("queue is empty\n");

10. else{ int k=0;

11. while(k<=rear-1){

12. printf("%d ",queue[k]);

13. k++;

14. }

15. }

16. }

17. void insert(){

18. if(rear== (capacity-1)){

19. printf("Queue is full\n");

20. }

21. else{

22. printf("Enter the element to be inserted\n");

23. scanf("%d",&queue[rear++]);

24. }

25. }

26. void deletei(){

27. if(front==rear){

28. printf("Stack is empty\n");

29. }

30. else{

31. printf("\n%d deleted\n",queue[front]);

32. for(int i=1;i<rear;i++){

33. queue[i-1]=queue[i];

34. }rear--;

35. }

36. }

37.

38. int main()

39. {

40. insert();

41. insert();

42. insert();

43. Traverse();

44. deletei();

45. Traverse();

46.

47.

48. return 0;

49. }

50.

Double Linked LIST

1. *#include* <iostream>

2. using namespace std;

3.

4. struct node{

5. int data;

6. struct node \*left;

7. struct node \*right;

8. };

9. struct node \*head=NULL;

10. void traverse(){

11. struct node \*temp;

12. if(head==NULL){cout<<"List is empty\n";}

13. else{

14. temp=head;

15. cout<< "front\n";

16. while(temp->right!=NULL){

17. cout<<temp->data<<" ";

18. temp=temp->right;

19. }cout<<temp->data<<" ";

20. cout<< "\nback\n";

21. while(temp!=NULL){

22. cout<<temp->data<<" ";

23. temp=temp->left;

24. }

25. }

26. }

27. void append(int data){

28. struct node \*temp;

29. struct node \*newnode=(struct node\*)malloc(sizeof(struct node\*));

30. newnode->data=data;

31. newnode->left=NULL;

32. newnode->right=NULL;

33.

34. if(head==NULL){

35. head=newnode;

36. }

37. else{

38. temp=head;

39. while(temp->right!=NULL){

40. temp=temp->right;

41. }temp->right=newnode;

42. newnode->left=temp;

43. }

44. }

45. void insertAT(int data,int pos){

46. struct node \*temp1=head,\*temp2;

47. struct node \*newnode=(struct node\*)malloc(sizeof(struct node\*));

48. newnode->data=data;

49. newnode->left=NULL;

50. newnode->right=NULL;

51.

52. if(pos==1){

53. newnode->left=NULL;

54. newnode->right=temp1;

55. temp1->left=newnode;

56. head=newnode;

57. }

58. else{ pos--;

59. while(--pos){

60. temp1=temp1->right;

61. }temp2=temp1->right;

62. newnode->right=temp2;

63. temp2->left=newnode;

64. newnode->left=temp1;

65. temp1->right=newnode;

66. }

67. }

68. void deleteAT(int pos){

69. struct node \*temp1=head,\*temp2;

70. if(pos==1){

71. head=head->right;

72. head->left=NULL;

73. temp1->right=NULL;

74. free(temp1);

75. }

76. else{ pos--;

77. while(--pos){

78. temp1=temp1->right;

79. }temp2=temp1->right;

80. temp1->right=temp2->right;

81. temp2->right->left=temp1;

82. temp2->right=NULL;

83. temp2->left=NULL;

84. free(temp2);

85. }

86. }

87.

88. void reverse(){

89. struct node \*temp1=head,\*temp2,\*temp3;

90. while(temp1!=NULL){

91. temp2=temp1;

92. temp1=temp1->right;

93. temp3=temp2->left;

94. temp2->left=temp2->right;

95. temp2->right=temp3;

96. }head=temp2;

97. }

98. int main()

99. {

100. traverse();

101. append(10);

102. append(20);

103. append(30);

104. append(40);

105. traverse();

106. cout<<"\*\*\*\*\*\*\n";

107. deleteAT(2);

108. traverse();

109. cout<<"\*\*\*\*\*\*\n";

110. insertAT(20,2);

111. traverse();

112. cout<<"\*\*\*\*\*\*\n";

113. reverse();

114. traverse();

115. return 0;

116. }

117.

Circular Double Linked List

1. *#include* <iostream>

2. using namespace std;

3.

4. struct node{

5. int data;

6. struct node \*left;

7. struct node \*right;

8. };

9. struct node \*head=NULL;

10. void traverse(){

11. struct node \*temp;

12. if(head==NULL){cout<<"Circular List is empty\n";}

13. else{

14. temp=head;

15. cout<< "\nCircular List --> ";

16. while(temp->right!=head){

17. cout<<temp->data<<" ";

18. temp=temp->right;

19. }cout<<temp->data<<" ";

20. cout<<temp->right->data<<" ";

21. }

22. }

23. void append(int data){

24. struct node \*temp;

25. struct node \*newnode=(struct node\*)malloc(sizeof(struct node\*));

26. newnode->data=data;

27. newnode->left=NULL;

28. newnode->right=NULL;

29.

30. if(head==NULL){

31. head=newnode;

32. head->right=head;

33. head->left=head;

34.

35. }

36. else{

37. temp=head;

38. while(temp->right!=head){

39. temp=temp->right;

40. }temp->right=newnode;

41. newnode->left=temp;

42. newnode->right=head;

43. head->left=newnode;

44. }

45. }

46. void deleteAT(int pos){

47. struct node \*temp1=head,\*temp2;

48. if(pos==1){head=temp1->right;

49. temp1->left->right=temp1->right;

50. temp1->right->left=temp1->left;

51. free(temp1);

52. }

53. else{

54. while(--pos){

55. temp1=temp1->right;

56. }

57. temp1->left->right=temp1->right;

58. temp1->right->left=temp1->left;

59. free(temp1);

60. }

61. }

62. void insertAT(int data, int pos){

63. struct node \*temp1=head,\*temp2;

64. struct node \*newnode=(struct node\*)malloc(sizeof(struct node\*));

65. newnode->data=data;

66. newnode->left=NULL;

67. newnode->right=NULL;

68.

69. if(pos==1){

70. newnode->left=temp1->left;

71. newnode->right=temp1;

72. temp1->left->right=newnode;

73. temp1->left=newnode;

74. head=newnode;

75. }

76. else{

77. while(--pos){

78. temp1=temp1->right;

79. }

80. newnode->left=temp1->left;

81. newnode->right=temp1;

82. temp1->left->right=newnode;

83. temp1->left=newnode;

84. }

85.

86. }

87. void reverse(){

88. }

89. int main()

90. {

91. traverse();

92.

93. append(10);

94. append(20);

95. append(30);

96. append(40);

97. traverse();

98.

99. deleteAT(2);

100. traverse();

101.

102. insertAT(20,2);

103. traverse();

104. return 0;

105. }

106.

Single linked list(without scanf)

1. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

2.

3.  *Online C++ Compiler.*

4.  *Code, Compile, Run and Debug C++ program online.*

5. *Write your code in this editor and press "Run" button to compile and execute it.*

6.

7. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

8.

9. *#include* <iostream>

10. using namespace std;

11.

12. struct node{

13. int data;

14. struct node \*link;

15. };

16. struct node \*head=NULL;

17. void append(int);

18. void traverse();

19. void deleteAT(int);

20. void insertAT(int ,int);

21. void reverse();

22.

23.

24.

25. int main()

26. {

27. traverse();

28. append(10);

29. append(20);

30. append(30);

31. append(40);

32. traverse();

33.

34. deleteAT(2);

35. traverse();

36.

37. insertAT(20,2);

38. traverse();

39.

40. reverse();

41. traverse();

42. return 0;

43. }

44.

45. void traverse(){

46. struct node \*temp;

47. if(head==NULL){cout<<"List is empty\n";}

48. else{

49. temp=head;

50. cout<< "\nLinked list --";

51. while(temp->link!=NULL){

52. cout<<temp->data<<" ";

53. temp=temp->link;

54. }cout<<temp->data<<" ";

55. }

56. }

57. void append(int data){

58. struct node \*temp;

59. struct node \*newnode=(struct node\*)malloc(sizeof(struct node\*));

60. newnode->data=data;

61. newnode->link=NULL;

62.

63. if(head==NULL){

64. head=newnode;

65. }

66. else{

67. temp=head;

68. while(temp->link!=NULL){

69. temp=temp->link;

70. }temp->link=newnode;

71. }

72. }

73. void insertAT(int data,int pos){

74. struct node \*temp1=head,\*temp2;

75. struct node \*newnode=(struct node\*)malloc(sizeof(struct node\*));

76. newnode->data=data;

77. newnode->link=NULL;

78.

79. if(pos==1){

80. newnode->link=temp1;

81. head=newnode;

82. }

83. else{ pos--;

84. while(--pos){

85. temp1=temp1->link;

86. }temp2=temp1->link;

87. newnode->link=temp2;

88. temp1->link=newnode;

89. }

90. }

91. void deleteAT(int pos){

92. struct node \*temp1=head,\*temp2;

93. if(pos==1){

94. head=head->link;

95. temp1->link=NULL;

96. free(temp1);

97. }

98. else{ pos--;

99. while(--pos){

100. temp1=temp1->link;

101. }temp2=temp1->link;

102. temp1->link=temp2->link;

103. temp2->link=NULL;

104. free(temp2);

105. }

106. }

107.

108. void reverse(){

109. struct node \*current=head,\*next=head,\*previous=NULL;

110. while(current!=NULL){

111. next=next->link;

112. current->link=previous;

113. previous=current;

114. current=next;

115. }head=previous;

116.

117. }

118.

Stack Using Linked List

1. *#include* <iostream>

2. using namespace std;

3. *//Stack using linked list*

4. struct node{

5. int data;

6. struct node \*link;

7. };

8. struct node \*head=NULL;

9.

10. void push(int);

11. void pop();

12. void traverse();

13.

14. void push(int data){

15. struct node \*newnode=(struct node\*)malloc(sizeof(struct node\*));

16. newnode->data=data;

17. newnode->link=NULL;

18.

19. if(head==NULL){

20. head=newnode;

21. }

22. else{

23. newnode->link=head;

24. head=newnode;

25. }

26. }

27.

28. void pop(){

29. struct node \*temp1=head;

30. head=temp1->link;

31. free(temp1);

32. }

33.

34. void traverse(){

35. struct node \*temp;

36. if(head==NULL){cout<<"Stack is empty\n";}

37. else{

38. temp=head;

39. cout<< "\nStack Elements --";

40. while(temp->link!=NULL){

41. cout<<temp->data<<" ";

42. temp=temp->link;

43. }cout<<temp->data<<" ";

44. }

45. }

46.

47. int main()

48. {

49. traverse();

50. push(10);

51. push(20);

52. push(30);

53. push(40);

54. traverse();

55. pop();

56. traverse();

57.

58. push(40);

59. traverse();

60.

61. return 0;

62. }

Binary Search Tree

1. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

2.

3.  *Online C++ Compiler.*

4.  *Code, Compile, Run and Debug C++ program online.*

5.  *Write your code in this editor and press "Run" button to compile and execute it.*

6.

7.  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

8.

9. *#include* <iostream>

10. using namespace std;

11.

12. struct node{

13. int data;

14. struct node \*left;

15. struct node \*right;

16. };

17.

18.

19. struct node \*head=NULL;

20.

21. void insert(int data)

22. {

23. struct node \*current,\*parent;

24. struct node \*newnode=(struct node\*)malloc(sizeof(struct node));

25. newnode->left=NULL;

26. newnode->right=NULL;

27. newnode->data=data;

28. if(head==NULL){

29. head=newnode;

30. current=head;

31. }

32. else{current=head;

33. while(current!=NULL)

34. {

35. parent=current;

36. if((newnode->data) < (current->data))

37. {

38. current=current->left;

39. }

40. else

41. {

42. current=current->right;

43. }

44. }

45. if((newnode->data) > (parent->data)) parent->right=newnode;

46. else parent->left=newnode;

47. }

48. }

49.

50. void DeleteNoChild(int data){

51. struct node \*current=head,\*parent;

52. while(current!=NULL)

53. {

54. if(data == (current->data))

55. {

56. if(current==parent->left) parent->left=NULL;

57. else parent->right=NULL;

58. free(current);

59. }

60. else if(data < (current->data))

61. {

62. parent=current;

63. current=current->left;

64. }

65. else

66. {

67. parent=current;

68. current=current->right;

69. }

70. }

71.

72. }

73.

74. void Delete1Child(int data){

75. struct node \*current=head,\*parent;

76. while(current!=NULL)

77. {

78. if(data == (current->data))

79. {

80. if(current==parent->left)

81. {

82. if(current->left!=NULL) parent->left = current->left;

83. else if(current->right!=NULL) parent->left = current->right;

84. }

85. else

86. {

87. if(current->left!=NULL) parent->right = current->left;

88. else if(current->right!=NULL) parent->right = current->right;

89. }

90. current=NULL;

91. free(current);

92. }

93. else if(data < (current->data)){

94. parent=current;

95. current=current->left;

96. }

97. else{

98. parent=current;

99. current=current->right;

100. }

101. }

102. }

103.

104.

105.

106. struct node\* findMax(struct node \*subParent){

107. struct node \*temp=subParent->left,\*nminone=subParent;

108. int cnt=0;

109. while(temp->right){

110. if((temp->right->right != NULL)) temp=temp->right;

111. else return temp;

112. }

113. return subParent;

114. }

115.

116. void Delete2Child(int data){

117. struct node \*current=head,\*parent,\*temp;

118. while(current!=NULL){

119. if(data == (current->data)){

120. {

121. temp=findMax(current);

122. if(current==temp){

123. current->data=temp->left->data;

124. current->left=temp->left->left;

125. break;

126. }

127. else {

128. current->data = temp->right->data;

129. free(temp->right);

130. temp->right=NULL;

131. break;

132. }

133. }

134. }

135. else if(data < (current->data)){

136. parent=current;

137. current=current->left;

138. }

139. else{

140. parent=current;

141. current=current->right;

142. }

143. }

144.

145. }

146.

147. void InOrder(struct node \*node)

148. {

149. if(node==NULL) return;

150. InOrder(node->left);

151. printf("%d ",node->data);

152. InOrder(node->right);

153. }

154.

155. void PreOrder(struct node \*node)

156. {

157. if(node==NULL) return;

158. printf("%d ",node->data);

159. InOrder(node->left);

160. InOrder(node->right);

161. }

162.

163. void PostOrder(struct node \*node)

164. {

165. if(node==NULL) return;

166. InOrder(node->left);

167. InOrder(node->right);

168. printf("%d ",node->data);

169. }

170.

171. int main()

172. {

173. insert(50);

174. insert(60);

175. insert(40);

176. insert(55);

177. insert(54);

178. insert(56);

179. insert(80);

180. struct node \*current=head;

181. InOrder(current);

182. cout<<"\n";

183. *//DeleteNoChild(30);*

184. *//Delete1Child(10);*

185. Delete2Child(60);

186. InOrder(current);

187. return 0;

188. }

189.

190.