**Singly Linked List**

|  |
| --- |
| /\*  \* C++ Program to Implement Singly Linked List  \*/  #include<iostream>  #include<cstdio>  #include<cstdlib>  using namespace std;  /\*  \* Node Declaration  \*/  struct node  {  int info;  struct node \*next;  }\*start;    /\*  \* Class Declaration  \*/  class single\_llist  {  public:  node\* create\_node(int);  void insert\_begin();  void insert\_pos();  void insert\_last();  void delete\_pos();  void sort();  void search();  void update();  void reverse();  void display();  single\_llist()  {  start = NULL;  }  };    /\*  \* Main :contains menu  \*/  main()  {  int choice, nodes, element, position, i;  single\_llist sl;  start = NULL;  while (1)  {  cout<<endl<<"---------------------------------"<<endl;  cout<<endl<<"Operations on singly linked list"<<endl;  cout<<endl<<"---------------------------------"<<endl;  cout<<"1.Insert Node at beginning"<<endl;  cout<<"2.Insert node at last"<<endl;  cout<<"3.Insert node at position"<<endl;  cout<<"4.Sort Link List"<<endl;  cout<<"5.Delete a Particular Node"<<endl;  cout<<"6.Update Node Value"<<endl;  cout<<"7.Search Element"<<endl;  cout<<"8.Display Linked List"<<endl;  cout<<"9.Reverse Linked List "<<endl;  cout<<"10.Exit "<<endl;  cout<<"Enter your choice : ";  cin>>choice;  switch(choice)  {  case 1:  cout<<"Inserting Node at Beginning: "<<endl;  sl.insert\_begin();  cout<<endl;  break;  case 2:  cout<<"Inserting Node at Last: "<<endl;  sl.insert\_last();  cout<<endl;  break;  case 3:  cout<<"Inserting Node at a given position:"<<endl;  sl.insert\_pos();  cout<<endl;  break;  case 4:  cout<<"Sort Link List: "<<endl;  sl.sort();  cout<<endl;  break;  case 5:  cout<<"Delete a particular node: "<<endl;  sl.delete\_pos();  break;  case 6:  cout<<"Update Node Value:"<<endl;  sl.update();  cout<<endl;  break;  case 7:  cout<<"Search element in Link List: "<<endl;  sl.search();  cout<<endl;  break;  case 8:  cout<<"Display elements of link list"<<endl;  sl.display();  cout<<endl;  break;  case 9:  cout<<"Reverse elements of Link List"<<endl;  sl.reverse();  cout<<endl;  break;  case 10:  cout<<"Exiting..."<<endl;  exit(1);  break;  default:  cout<<"Wrong choice"<<endl;  }  }  }    /\*  \* Creating Node  \*/  node \*single\_llist::create\_node(int value)  {  struct node \*temp, \*s;  temp = new(struct node);  if (temp == NULL)  {  cout<<"Memory not allocated "<<endl;  return 0;  }  else  {  temp->info = value;  temp->next = NULL;  return temp;  }  }    /\*  \* Inserting element in beginning  \*/  void single\_llist::insert\_begin()  {  int value;  cout<<"Enter the value to be inserted: ";  cin>>value;  struct node \*temp, \*p;  temp = create\_node(value);  if (start == NULL)  {  start = temp;  start->next = NULL;  }  else  {  p = start;  start = temp;  start->next = p;  }  cout<<"Element Inserted at beginning"<<endl;  }    /\*  \* Inserting Node at last  \*/  void single\_llist::insert\_last()  {  int value;  cout<<"Enter the value to be inserted: ";  cin>>value;  struct node \*temp, \*s;  temp = create\_node(value);  s = start;  while (s->next != NULL)  {  s = s->next;  }  temp->next = NULL;  s->next = temp;  cout<<"Element Inserted at last"<<endl;  }    /\*  \* Insertion of node at a given position  \*/  void single\_llist::insert\_pos()  {  int value, pos, counter = 0;  cout<<"Enter the value to be inserted: ";  cin>>value;  struct node \*temp, \*s, \*ptr;  temp = create\_node(value);  cout<<"Enter the postion at which node to be inserted: ";  cin>>pos;  int i;  s = start;  while (s != NULL)  {  s = s->next;  counter++;  }  if (pos == 1)  {  if (start == NULL)  {  start = temp;  start->next = NULL;  }  else  {  ptr = start;  start = temp;  start->next = ptr;  }  }  else if (pos > 1 && pos <= counter)  {  s = start;  for (i = 1; i < pos; i++)  {  ptr = s;  s = s->next;  }  ptr->next = temp;  temp->next = s;  }  else  {  cout<<"Positon out of range"<<endl;  }  }    /\*  \* Sorting Link List  \*/  void single\_llist::sort()  {  struct node \*ptr, \*s;  int value;  if (start == NULL)  {  cout<<"The List is empty"<<endl;  return;  }  ptr = start;  while (ptr != NULL)  {  for (s = ptr->next;s !=NULL;s = s->next)  {  if (ptr->info > s->info)  {  value = ptr->info;  ptr->info = s->info;  s->info = value;  }  }  ptr = ptr->next;  }  }    /\*  \* Delete element at a given position  \*/  void single\_llist::delete\_pos()  {  int pos, i, counter = 0;  if (start == NULL)  {  cout<<"List is empty"<<endl;  return;  }  cout<<"Enter the position of value to be deleted: ";  cin>>pos;  struct node \*s, \*ptr;  s = start;  if (pos == 1)  {  start = s->next;  }  else  {  while (s != NULL)  {  s = s->next;  counter++;  }  if (pos > 0 && pos <= counter)  {  s = start;  for (i = 1;i < pos;i++)  {  ptr = s;  s = s->next;  }  ptr->next = s->next;  }  else  {  cout<<"Position out of range"<<endl;  }  free(s);  cout<<"Element Deleted"<<endl;  }  }    /\*  \* Update a given Node  \*/  void single\_llist::update()  {  int value, pos, i;  if (start == NULL)  {  cout<<"List is empty"<<endl;  return;  }  cout<<"Enter the node postion to be updated: ";  cin>>pos;  cout<<"Enter the new value: ";  cin>>value;  struct node \*s, \*ptr;  s = start;  if (pos == 1)  {  start->info = value;  }  else  {  for (i = 0;i < pos - 1;i++)  {  if (s == NULL)  {  cout<<"There are less than "<<pos<<" elements";  return;  }  s = s->next;  }  s->info = value;  }  cout<<"Node Updated"<<endl;  }    /\*  \* Searching an element  \*/  void single\_llist::search()  {  int value, pos = 0;  bool flag = false;  if (start == NULL)  {  cout<<"List is empty"<<endl;  return;  }  cout<<"Enter the value to be searched: ";  cin>>value;  struct node \*s;  s = start;  while (s != NULL)  {  pos++;  if (s->info == value)  {  flag = true;  cout<<"Element "<<value<<" is found at position "<<pos<<endl;  }  s = s->next;  }  if (!flag)  cout<<"Element "<<value<<" not found in the list"<<endl;  }    /\*  \* Reverse Link List  \*/  void single\_llist::reverse()  {  struct node \*ptr1, \*ptr2, \*ptr3;  if (start == NULL)  {  cout<<"List is empty"<<endl;  return;  }  if (start->next == NULL)  {  return;  }  ptr1 = start;  ptr2 = ptr1->next;  ptr3 = ptr2->next;  ptr1->next = NULL;  ptr2->next = ptr1;  while (ptr3 != NULL)  {  ptr1 = ptr2;  ptr2 = ptr3;  ptr3 = ptr3->next;  ptr2->next = ptr1;  }  start = ptr2;  }    /\*  \* Display Elements of a link list  \*/  void single\_llist::display()  {  struct node \*temp;  if (start == NULL)  {  cout<<"The List is Empty"<<endl;  return;  }  temp = start;  cout<<"Elements of list are: "<<endl;  while (temp != NULL)  {  cout<<temp->info<<"->";  temp = temp->next;  }  cout<<"NULL"<<endl;  } |