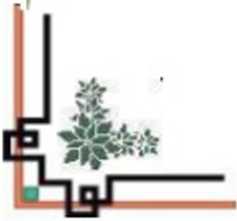




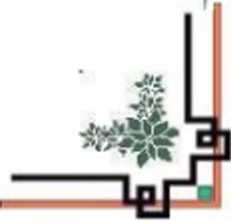
**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

🙤🙧🟍🙥🙦



***TP Hồ Chi Minh, ngày 21 Tháng 4 năm 2018***





C:\Users\Dell\Downloads\media\image6.jpeg



C:\Users\Dell\Downloads\media\image5.jpeg





C:\Users\Dell\Downloads\media\image6.jpegC:\Users\Dell\Downloads\media\image6.jpeg

**Môn: Cấu Trúc Dữ Liệu Và Giải Thuật**

**Sinh Viên Trình Bày: Võ Nhật Bảo**

**MSSV:17520277**

**Lớp: TMĐT2017**

******

**• Cấu trúc Node**

**• Cấu trúc List**

**• Khởi tạo danh sách rỗng**

**• Tạo 1 nút có Data = x**

**• Chèn 1 phần tử vào ĐẦU danh sách**

**• Chèn 1 phần tử vào CUỐI danh sách**

**• Chèn 1 phần tử vào TRƯỚC phần tử Q danh sách**

**• Chèn 1 phần tử vào SAU phần tử Q danh sách**

**• Huỷ 1 phần tử ĐẦU trong danh sách**

**• Huỷ 1 phần tử CUỐI trong danh sách**

**• Huỷ 1 phần tử có khóa = x trong danh sách**

**• Hủy phần tử đứng trước Q**

**• Duyệt danh sách**

**• Tìm phần tử**

**• In các phần tử trong list**

**• Hủy danh sách liên kết**

**• Sắp xếp**

**---End---**

C:\Users\Dell\Downloads\media\image6.jpegC:\Users\Dell\Downloads\media\image6.jpeg

|  |  |  |
| --- | --- | --- |
| **NỘI DUNG** | **DSLK ĐƠN** | **DSLK KÉP** |
| 1. **Cấu trúc Node** | typedef struct tagNode  {  Data Info;  struct tagNode \*pNext;  }Node; | typedef struct tagDnode  {  Data Info;  struct tagDnode \*pPre;  struct tagDnode \*pNext;  }DNode; |
| 1. **Cấu trúc List** | typedef struct tagList  {  Node \*pHead;  Node \*pTail;  }LIST; | typedef struct tagDList  {  DNode \*pHead;  DNode \*pTail;  }DList; |
| 1. **Khởi tạo danh sách rỗng** | void CreateList(List &l)  {  l.pHead = NULL;  l.pTail = NULL;  } | void CreateDList(DList &l)  {  l.DHead = NULL;  l.DTail = NULL;  } |
| 1. **Tạo 1 nút có Data = x** | Node\* CreateNode(Data x)  {  Node \*p;  p = new Node;  if (p == NULL) exit(1);  p->Info = x;  p->pNext = NULL;  return p;  } | DNode \*CreateDNode(int x)  {  DNode \*tam;  tam = new DNode;  if (tam == NULL)  {  cout<<"khong con du bo nho";  exit(1);  }  else  {  tam->Info = x;  tam->pNext = NULL;  tam->pPre = NULL;  return tam;  }  } |
| 1. **Chèn 1 phần tử vào ĐẦU danh sách** | void AddHead(LIST &l, Node\* p)  {  if (l.pHead == NULL)  {  l.pHead = p;  l.pTail = l.pHead;  }  else  {  p->pNext = l.pHead;  l.pHead = p;  }  } | void AddFirst(DList &l, DNode \*tam)  {  if (l.pHead == NULL)  {  l.pHead = tam;  l.pTail = l.pHead;  }  else  {  tam->pNext = l.pHead;  l.pHead->pPre = tam;  l.pHead = tam;  }  } |
| 1. **Chèn 1 phần tử vào CUỐI danh sách** | void AddTail(LIST &l, Node \*p)  {  if (l.pHead == NULL)  {  l.pHead = p;  l.pTail = l.pHead;  }  else  {  l.pTail->Next = p;  l.pTail = p;  }  } | void AddEnd(DList &l, DNode \*tam)  {  if (l.pHead == NULL)  {  l.pHead = tam;  l.pTail = l.pHead;  }  else  {  tam->pPre = l.pTail;  l.pTail->pNext = tam;  tam = l.pTail;  }  } |
| 1. **Chèn 1 phần tử vào TRƯỚC phần tử Q danh sách** | void InsertAfterQ(LIST &l, Node \*p, Node \*q)  {  if (q != NULL)  {  p->pNext = Q->Next;  q->pNext = p;  if (l.pTail == q)  l.Tail = q;  }  else  AddHead(l, q);  } | void AddLastQ(DList &l, DNode \*tam, DNode \*q)  {  DNode \*p;  p = q->pNext;  if (q != NULL)  {  tam->pNext = p;  tam->pPre = q;  q->pNext = tam;  if (p != NULL)  p->pPre = tam;  if (q == l.pTail)  l.pTail = tam;  }  else  AddFirst(l, tam);  } |
| 1. **Chèn 1 phần tử vào SAU phần tử Q danh sách** | void InsertAfterQ(LIST &l, Node \*p, Node \*q)  {  if (q != NULL)  {  p->pNext = Q->Next;  q->pNext = p;  if (l.pTail == q)  l.Tail = q;  }  else  AddTail(l, q);  } | void AddBeforeQ(DList &l, DNode \*tam, DNode \*q)  {  DNode \*p;  p = q->pPre;  if (q != NULL)  {  tam->pNext = q;  q->pPre = tam;  tam->pPre = p;  if (p != NULL)  p->pNext = tam;  if (q == l.pHead)  l.pHead = tam;  }  else  AddEnd(l, tam);  } |
| 1. **Huỷ 1 phần tử ĐẦU trong danh sách** | int RemoveHead(LIST &l, int &x)  {  Node \*p;  if (l.pHead != NULL)  {  p = l.pHead;  x = p->Info;  l.pHead = l.pHead->pNext;  delete p;  if (l.pHead == NULL)  l.pTail = NULL;  return 1;  }  return 0;  } | void DeleteFirst(DList &l)  {  DNode \*p;  if (l.pHead != NULL)  {  p = l.pHead;  l.pHead = l.pHead>pNext;  l.pHead->pPre = NULL;  delete p;  if (l.pHead == NULL)  l.pTail = NULL;  }  } |
| 1. **Huỷ 1 phần tử CUỐI trong danh sách** | int endTail(LIST &l, int &tam)  {  Node \*p;  if (tam->data == x && tam->next == NULL)  {  Node\* tam1 = tam;  tam = NULL;  delete(tam1);  return;  }  } | void DeleteEnd(DList &l)  {  DNode \*p;  if (l.pHead != NULL)  {  p = l.pTail;  l.pTail = l.pTail->Pre;  l.pTail->pNext = NULL;  delete p;  if (l.pTail == NULL)  l.pHead = NULL;  }  } |
| 1. **Huỷ 1 phần tử có khóa = x trong danh sách** | int RemoveX(LIST &l, int x)  {  Node \*p, \*q = NULL;  p = l.Head;  while ((p != NULL) && (p->Info != x))  {  q = p;  p = p->Next;  }  if (p == NULL)  return 0;  if (q != NULL)  DeleteAfterQ(l, q, x);  else  RemoveHead(l, x);  return 1;  } | int DeleteX(DList &l, int x)  {  DNode \*p;  DNode \*q;  q = NULL;  p = l.pHead;  while (p != NULL)  {  if (p->Info == x)  break;  q = p;  p = p->pNext;  }  if (q == NULL) return 0;  if (q != NULL)  DeleteLastQ(l, q);  else  DeleteFirst(l);  return 1;  } |
| 1. **Hủy phần tử đứng trước Q** | int RemoveBeforeQ(LIST &l, Node \*q, int &x)  {  Node \*p;  if (q != NULL)  {  p = q->pTail;  if (p != NULL)  {  if (p == l.pNext)  l.pNext = q;  q->pTail = p->pTail; x = p->Info;  delete p;  }  return 1;  }  else  return 0;  } | void DeleteBeforeQ(DList &l, DNode \*q)  {  DNode \*p;  if (q != NULL)  {  p = q->pPre;  if (p != NULL)  {  q->pPre = p->pPre;  if (p == l.pHead)  l.pHead = q;  else  p->pPre->pNext = q;  delete p;  }  }  else  DeleteEnd(l);  } |
| 1. **Duyệt danh sách** | void ProcessList(LIST &l)  {  Node \*p;  p = l.pHead;  While(p != NULL)  {  ProcessList(p);  p = p->pNext;  }  } | void BrowseNext(Dlist &l)  {  Node \*p;  p = l.pHead;  While(p != NULL)  {  BrowseNext(p);  p = p->pNext;  }  }  void BrowsePrev(LIST &l)  {  Node \*p;  p = l.pTail;  While(p != NULL)  {  BrowsePrev(p);  p = p->pPre;  }  } |
| 1. **Tìm phần tử** | Node \*Search(LIST l, Data x)  {  Node \*p;  p = l.pHead;  while ((p != NULL) && (p->Info != x))  p = p->pNext;  return p;  } | DNode \* SearchNode(Lít list, int key)  {  Node \*i = list.head;  while (i&&i->info != key)  i = i->next;  return i;  } |
| 1. **In các phần tử trong list** | void PrintList(List l)  {  Node \*p;  p = l.pHead;  while (p != NULL)  {  cout << p->Info<<endl;  p = p->pNext;  }  } | void BrowseNext(List list)  {  for (Node\*i = list.head; i; i = i->next)  { ////  }  }  void BrowsePrev(List list)  {  for (Node\*i = list.tail; i; i = i->prev)  { ////  }  } |
| 1. **Hủy danh sách liên kết** | void RemoveList(List &l)  {  Node \*p;  while (l.pHead != NULL)//còn phần tử trong List  {  p = l.pHead;  l.pHead = p->pNext;  delete p;  }  } | void RemoveList(List &list)  {  DNode\* i;  while (list.head)  {  i = list.head;  list.head = list.head->next;  delete i;  }  } |
| 1. **Sắp xếp** | void QuickSort(List &l)  {  Node \*p, \*X;//X lưu địa chỉ của phần tử cầm canh  List l1, l2;  if (l.pHead == l.pTail) return;//đã có thứ tự  CreateList(l1);  CreateList(l2);  X = l.pHead;  l.pHead = X->pNext;  while (l.pHead != NULL)//tách L = L1 va L2  {  p = l.pHead;  l.pHead = p->pNext;  p->pNext = NULL;  if (p->Info <= X->Info)  AddHead(l1, p);  else  AddHead(l2, p);  }  QuickSort(l1);//Gọi đệ quy sắp xếp L1  QuickSort(l2);//Gọi đệ quy sắp xếp L2  if (l1.pHead != NULL)//nối l1, l2 va X vao l  {  l.pHead = l1.pHead;  l1.pTail->pNext = X;//nối X vào  }  else  l.pHead = X;  X->pNext = l2.pHead;  if (l2.pHead != NULL) //l2 có trên một phần tử  l.pTail = l2.pTail;  else //l2 không có phần tử nào  l.pTail = X;  } | void DoiChoTrucTiep(DList &l)  {  DNode \*p, \*q;  p = l.pHead;  while (p != l.pTail)  {  q = p->pNext;  while (q != NULL)  {  if (p->Info>q->Info)  HV(p, q);  q = q->pNext;  }  p = p->pNext;  }  } |