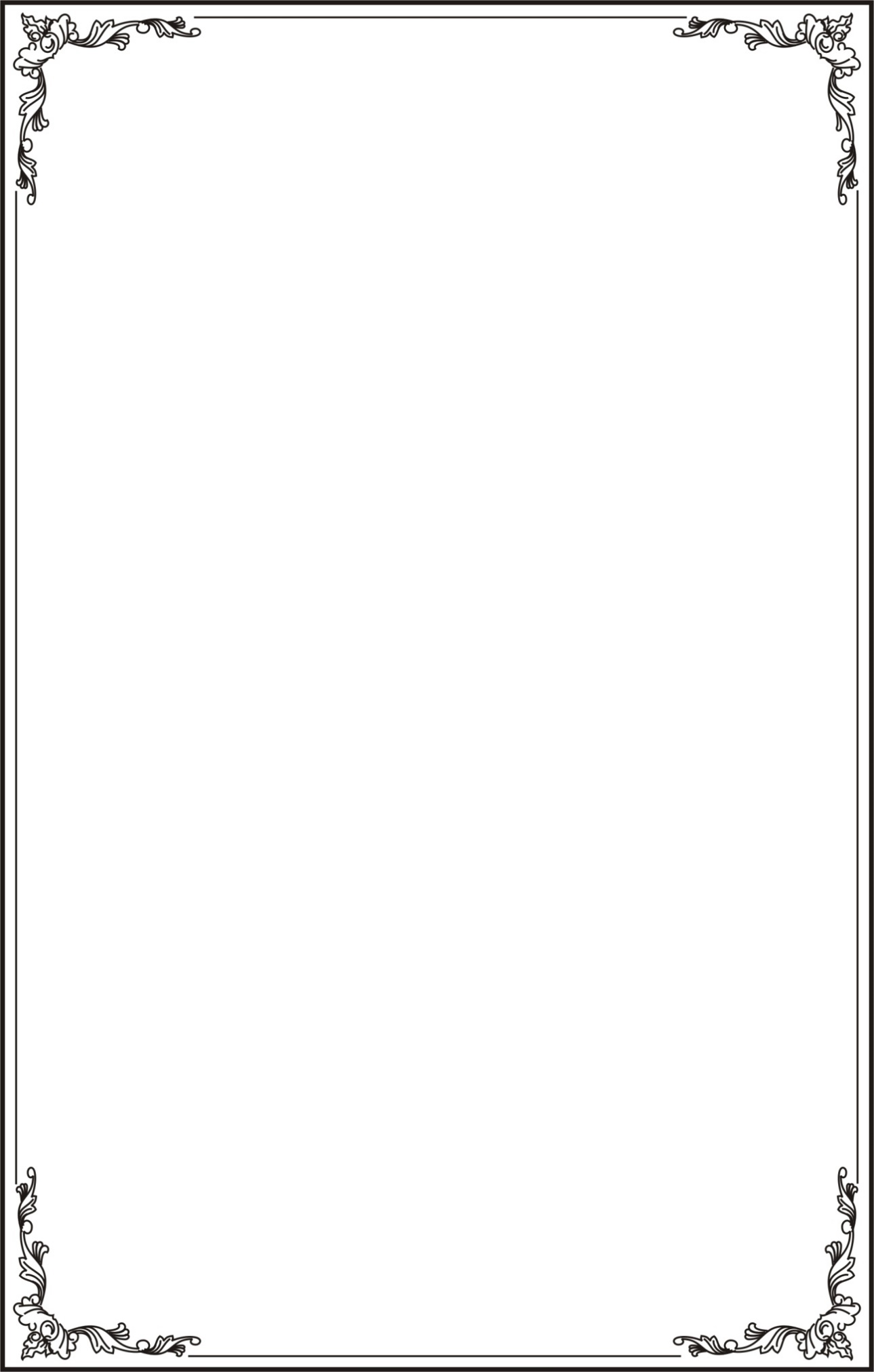
Logo, icon

Description automatically generated

**Mã môn học: [DASA230179\_22\_1\_09](https://utex.hcmute.edu.vn/course/view.php?id=25009" \o "Cau truc du lieu va giai thuat_ Nhom 09)**

**Giảng viên: Huỳnh Xuân Phụng**

**Nhóm 07**

|  |  |  |
| --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** |
| 1 | Nguyễn Hùng Dũng | 21110408 |
| 2 | Phan Ngọc Hân | 21110433 |

**ĐỀ TÀI:**

**MÔ PHỎNG CÁC CẤU TRÚC DỮ LIỆU:**

**STACK, QUEUE, DEQUEUE, LINKED LIST, CIRCLE LIST**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT**

**THÀNH PHỐ HỒ CHÍ MINH**

Mục lục

1. Cấu trúc dữ liệu Stack 1

2. Cấu trúc dữ liệu Queue 4

3. Cấu trúc dữ liệu Dequeue 6

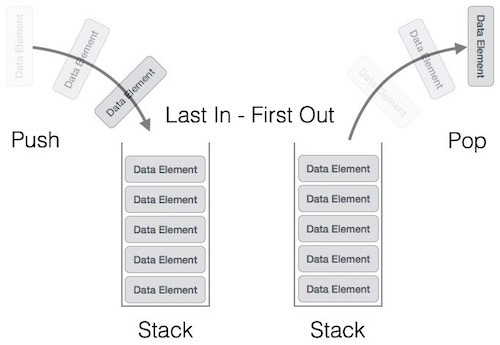
4. Cấu trúc dữ liệu Linked List 8

5. Cấu trúc dữ liệu Circle List 11

1. **Cấu trúc dữ liệu Stack**

Stack là cấu trúc dữ liệu hoạt động theo nguyên tắt LIFO (Last In First Out), vào sau ra trước.

Bài báo cáo này sẽ trình bày xây dựng cấu trúc Stack sử dụng danh sách liên kết đơn.



Hình ảnh mô tả hoạt động của cấu trúc dữ liệu Stack

Cấu trúc Stack:

|  |
| --- |
| struct stack {  int top;  int A[MAX];  }; |

-Top: Đỉnh của Stack.

-A[MAX]: mảng các phần tử tối đa của Stack

Khởi tạo Stack:

|  |
| --- |
| void Init(STACK& s) {  s.top = -1;  } |

Kiểm tra rỗng:

|  |
| --- |
| int Empty(STACK s) {  if (s.top == -1) {  return 1;  }  return 0;  } |

Kiểm tra đầy:

|  |
| --- |
| int IsFull(STACK s) {  if (s.top == MAX - 1) {  return 1;  }  return 0;  } |

Chèn data vào Stack :

|  |
| --- |
| int Top(STACK& s) {  int x;  // Kiểm tra stack khác rỗng  if (!Empty(s)) {  // Thực hiện gán phần tử cuối bằng x  x = s.A[s.top];  // Trả về x được gán bằng phần tử cuối cùng của mảng  return x;  }  else {  // Nếu stack rỗng thì trả về giá trị NULL  return NULL;  }  } |

|  |
| --- |
| void Push(STACK& s, int x) {  if (IsFull(s) == 0) {  s.top++;  s.A[s.top] = x;  }  } |

Dữ liệu vào

|  |
| --- |
| void Input(STACK& s, int n) {  // Duyệt từ 0 đến n  for (int i = 0; i < n; i++) {  // Thực hiện nhập giá trị vào biến x  int x;  cout<<"Nhap gia tri phan tu thu "<<i<<": ";  cin >> x;  // Thực hiện push vào Stack  Push(s, x);  } |

Dữ liệu ra:

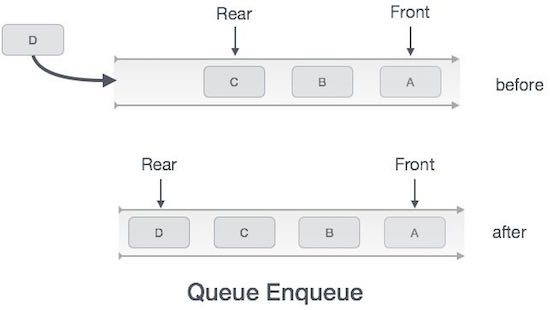
|  |
| --- |
| void Output(STACK s) {  // Duyệt từ phần tử Top về phần tử cuối trong Stack  for (int i = s.top; i > -1; i--) {  // Hiển thị kết quả  cout<< s.A[i]<<" ";  }  } |

Xóa phần tử:

|  |
| --- |
| int Pop(STACK& s) {  int x;  if (!Empty(s)) {  x = s.A[s.top];  s.top--;  }  return x;  } |

1. **Cấu trúc dữ liệu Queue**

Queue là một cấu trúc dữ liệu trừu tượng. Đặc điểm của hàng đợi là FIFO (first in first out) - có nghĩa là vào trước ra trước. Đặt tên là hàng đợi bởi vì nó là một cái gì đó tương tự như hàng đợi trong đời sống hàng ngày (xếp hàng).



Hình ảnh mô tả hoạt động của cấu trúc dữ liệu Queue

Cấu trúc Queue:

|  |
| --- |
| struct Queue  {  int Front, Rear;  Item Data[Max];  int count;  }; |

-Front: phần tử đầu hàng

-Rear: phần tử cuối hàng

-Data[Max]: mảng các phần tử tối đa của Queue

-count: đếm số phần tử của Queue

Khởi tạo Queue:

|  |
| --- |
| void Init(Queue& Q)  {  Q.Front = 0;  Q.Rear = -1;  Q.count = 0;  } |

Kiểm tra rỗng:

|  |
| --- |
| int Isempty(Queue Q)  {  if (Q.count == 0)  return 1;  return 0;  } |

Kiểm tra đầy:

|  |
| --- |
| int Isfull(Queue Q)  {  if (Q.count == Max)  return 1;  return 0;  } |

Thêm phần tử:

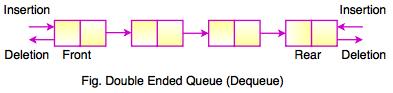
|  |
| --- |
| void Push(Queue& Q, item x)  {  if (Isfull(Q)) printf("Hang doi day !");  else  {  Q.Data[++Q.Rear] = x;  Q.count++;  }  } |

Xóa phần tử:

|  |
| --- |
| int Pop(Queue& Q)  {  if (Isempty(Q)) printf("Hang doi rong !");  else  {  item x = Q.Data[Q.Front];  for (int i = Q.Front; i < Q.Rear; i++)  Q.Data[i] = Q.Data[i + 1];  Q.Rear--;  Q.count--;  return x;  }  } |

1. **Cấu trúc dữ liệu Dequeue**

Dequeue là Hàng đợi kết thúc kép, trong đó chúng ta có thể chèn và xóa dữ liệu từ cả hai đầu có nghĩa là chúng ta có thể thực hiện các hoạt động enqueue và dequeue từ cả hai đầu.



Hình ảnh mô tả hoạt động của cấu trúc Dequeue

Phần tử của Dequeue: Node

|  |
| --- |
| struct NODE {  string data;  NODE\* next;  NODE\* prev;  }; |

Cấu trúc của Dequeue:

|  |
| --- |
| struct DEQUEUE {  NODE\* LeftMost;  NODE\* RightMost;  } |

Khởi tạo Dequeue:

|  |
| --- |
| void init() {  LeftMost = nullptr;  RightMost = nullptr;  } |

Kiểm tra rỗng:

|  |
| --- |
| bool isEmpty() {  return RightMost == nullptr;  } |

Khởi tạo NODE:

|  |
| --- |
| NODE\* makeNode(string x) {  NODE\* newNode = new NODE();  newNode->data = x;  newNode->next = nullptr;  newNode->prev = nullptr;  return newNode;  } |

Chèn dữ liệu:

|  |
| --- |
| void insert(string x, bool left) {  NODE\* newNode = makeNode(x);  if (this->isEmpty()) {  LeftMost = RightMost = newNode;  return;  }  if (left) {  newNode->next = LeftMost;  if (LeftMost != nullptr)  LeftMost->prev = newNode;  LeftMost = newNode;  }  else {  RightMost->next = newNode;  newNode->prev = RightMost;  RightMost = newNode;  }  } |

Xóa dữ liệu:

|  |
| --- |
| void deleteElement(bool left) {  if (this->isEmpty())  return;  // TRUE  if (left) {  NODE\* tmp = LeftMost;  LeftMost = LeftMost->next;  if (LeftMost == nullptr)  RightMost = nullptr;  else  LeftMost->prev = nullptr;  delete tmp;  }  // FALSE  else {  NODE\* tmp = RightMost;  RightMost = RightMost->prev;  if (RightMost == nullptr)  LeftMost = nullptr;  else  RightMost->next = nullptr;  delete (tmp);  }  } |

1. **Cấu trúc dữ liệu Linked List**

Linked list là một cấu trúc dữ liệu tuyến tính, bao gồm một chuỗi các node kết nối với nhau. Mỗi node có thể xem như một phần tử trong danh sách. Mỗi node sẽ lưu trữ dữ liệu (data) của node đó và địa chỉ (address) của node kế tiếp. Bài báo cáo này sẽ trình bày xây dựng cấu trúc Linked List sử dụng danh sách liên kết đơn.



Hình ảnh mô tả hoạt động của cấu trúc dữ liệu Linked List

Phần tử của Linked List: Node

|  |
| --- |
| struct node {  int data;  node\* next;  }; |

Khởi tạo NODE:

|  |
| --- |
| node\* createNode(int x) {  node\* temp = new node;  temp->next = NULL;  temp->data = x;  return temp;  } |

Chèn vào vị trí đầu tiên:

|  |
| --- |
| node\* addHead(node\* l, int x) {  node\* temp = new node;  temp->data = x;  temp->next = l;  l = temp;  return l;  } |

Chèn vào vị trí bất kì:

|  |
| --- |
| node\* addAt(node\* l, int k, int x) {  node\* p = l;  for (int i = 0; i < k - 1; i++) {  p = p->next;  }  node\* temp = new node;  temp->data = x;  temp->next = p->next;  p->next = temp;  return l;  } |

Chèn vào vị trí cuối:

|  |
| --- |
| node\* addTail(node\* l, int x) {  node\* p = l;  while (p->next != NULL) {  p = p->next;  }  node\* temp = new node;  temp->data = x;  temp->next = NULL;  p->next = temp;  return l;  } |

Xóa ở vị trí đầu tiên:

|  |
| --- |
| node\* deleteHead(node\* l) {  node\* p = l;  p = p->next;  delete(l);  return p;  } |

Xóa ở vị trí bất kì:

|  |
| --- |
| node\* deleteAt(node\* l, int k) {  node\* p = l;  for (int i = 0; i < k - 1; i++) {  p = p->next;  }  node\* temp = p->next;  p->next = p->next->next;  delete(temp);  return l;  } |

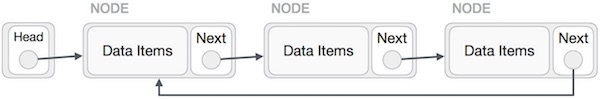
Xóa ở vị trí cuối:

|  |
| --- |
| node\* deleteTail(node\* l) {  node\* p = l;  while (p->next->next != NULL) {  p = p->next;  }  delete(p->next);  p->next = NULL;  return l;  } |

1. **Cấu trúc dữ liệu Circle List**

Circle List là một biến thể của Linked List trong đó phần tử đầu tiên trỏ đến phần tử cuối cùng và phần tử cuối cùng trỏ đến phần tử đầu tiên.

Bài báo cáo này sẽ trình bày xây dựng cấu trúc Circle List sử dụng danh sách liên kết đơn.



Hình ảnh mô tả hoạt động của cấu trúc dữ liệu Circle List

Phần tử của Circle List: Node

|  |
| --- |
| struct NODE  {  int data;  NODE\* next;  }; |

Khởi tạo NODE:

|  |
| --- |
| NODE\* createNODE(int x)  {  NODE\* temp = new NODE;  temp->next = NULL;  temp->data = x;  return temp;  } |

Chèn vào vị trí đầu tiên:

|  |
| --- |
| NODE\* addBegin(NODE\* last, int x)  {  NODE\* temp = new NODE;  temp->data = x;  temp->next = last;  last = temp;  return last;  } |

Chèn vào vị trí bất kì:

|  |
| --- |
| NODE\* addAfter(NODE\* last, int k, int x)  {  NODE\* p = last;  for (int i = 0; i < k - 1; i++)  {  p = p->next;  }  NODE\* temp = new NODE;  temp->data = x;  temp->next = p->next;  p->next = temp;  return last;  } |

Chèn vào vị trí cuối:

|  |
| --- |
| NODE\* addEnd(NODE\* last, int x)  {  NODE\* p = last;  while (p->next != NULL){  p = p->next;  }  NODE\* temp = new NODE;  temp->data = x;  temp->next = NULL;  p->next = temp;  return last;  } |

Xóa ở vị trí đầu tiên:

|  |
| --- |
| NODE\* deleteBegin(NODE\* last)  {  NODE\* p = last;  p = p->next;  delete(last);  return p;  } |

Xóa ở vị trí bất kì:

|  |
| --- |
| NODE\* deleteAfter(NODE\* last, int k)  {  NODE\* p = last;  for (int i = 0; i < k - 1; i++)  {  p = p->next;  }  NODE\* temp = p->next;  p->next = p->next->next;  delete(temp);  return last;  } |

Xóa ở vị trí cuối:

|  |
| --- |
| NODE\* deleteEnd(NODE\* last)  {  NODE\* p = last;  while (p->next->next != NULL)  {  p = p->next;  }  delete(p->next);  p->next = NULL;  return last;  } |