Linked Structures

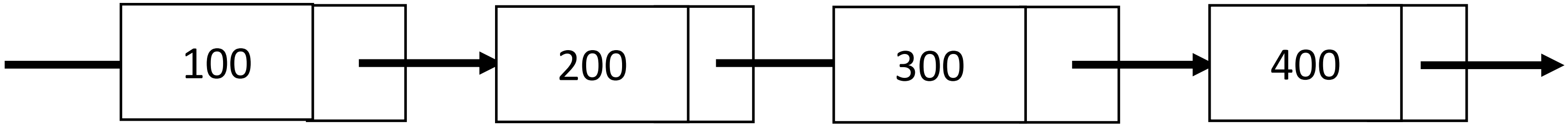
CSE 2020 Computer Science II

Learning Objective

- Implement linked structures, including insertion, search, traversal and deletion operations.

Linked Structures

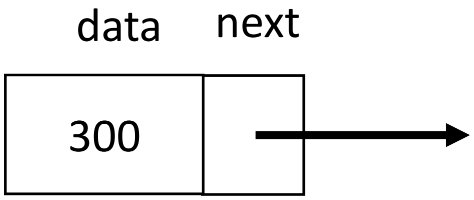
- A set of data linked together and organized by pointers
  - Linear linked structures (singly, doubly)
  - Nonlinear linked structures
- Example
  - A collection of integers
  - A collection of characters
  - A collection of points
  - A collection of employees

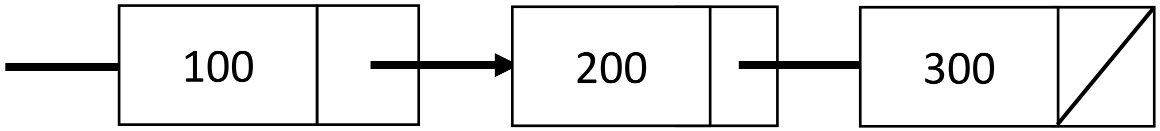


Examples

- A collection of integers

```
struct NodeType{
    int data;
    NodeType* next;
    NodeType(): data(0), next(nullptr) {}
    NodeType(int d): data(d), next(nullptr) {}
};
NodeType* p1 = new NodeType();  p1->data = 10;
NodeType* p2 = new NodeType(20);
NodeType* p3 = new NodeType(30);
p1->next = p2; p2->next = p3;
```





struct NodeType of Linked Points

```
struct NodeType{
        Point data;
        NodeType* next;
        NodeType(): data(), next(nullptr) {}
        NodeType(Point pt): data(pt), next(nullptr) {}
  };
```

Add Node to Head of Linked Points

- Please refer to Point class and struct NodeType
- pointers head, cur

```
NodeType* head = nullptr, * cur = nullptr;
Point pt(x, y); // pt is an object of Point class
head = new NodeType(pt);

while( condition ){
        cin >> x >> y;
        Point pt(x, y);
        cur = new NodeType(pt);
        cur->next = head;
        head = cur;
 }
```

Add Node to Back of Linked Points

- Three pointers: NodeType* head, * back, * cur;

```
Point pt(x, y);
head = new NodeType(pt);
back = head;
while( condition ){
   cin >> x >> y;
   Point pt(x, y);
   cur = new NodeType(pt);
   back->next = cur;
   back = cur;
}
```

Traverse Linked Points

- Print or traverse
  - pointers NodeType* head, *cur, head points to the first node

```
NodeType* cur = head;

while (cur != nullptr)
{
        cout << cur->data;
        cur = cur->next;
}
```

Search a point in Linked Points

- Search a point (a, b)
  - pointers NodeType* head, *cur, head points to the first node

```
NodeType* cur = head;
while (cur != nullptr &&
     (cur->data.get_x() != a || cur->data.get_y()!= b) )
{
        cur = cur->next;
}
if (cur == nullptr)
        cout << "(a,b) is not in the linked points"
else
        cout << "(a,b) is in the linked points"
```

Delete a Node Linked Points

- Delete the node head or pre->next

  pointers NodeType* head, *cur, *pre, head points to the first node

  delete head:

  ```
      NodeType* cur = head;
      head = head->next
     delete cur;
  ```

  delete pre->next:

  ```
      NodeType* cur = pre->next;
      pre->next = cur->next;
      delete cur;
  ```

Delete Linked Points

- Delete the linked structure
- pointers `NodeType*` head, *cur, head points to the first node

```
NodeType* cur = nullptr;
while (head != nullptr)
{
   cur = head;
   head = head->next;
   delete cur;
}
```

struct template

- struct template

```
template <typename T>
struct NodeType{
        T data;
        NodeType* next;
        NodeType(): data(), next(nullptr) {}
        NodeType(T d):data(d), next(nullptr) {}
};
NodeType<int>* p1 = new NodeType<int>();
NodeType<Point>* p2 = new NodeType<Point>();
NodeType<Employee>* p3 = new NodeType<Employee>();
```

Take-away

- Linked structures implementation
- Operations on Linked structures
  - Insertion
  - Deletion
  - Search
  - Traversal