```
Class Person {
public:

private:
    std::string name;
};
```

```
Class Student : public Person {
public:

private:
    double gpa;
};
```

```
Student s;
```

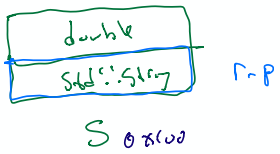
```
Person p = s;
```

s

Object Slicing

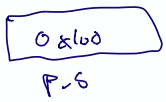
Student s;

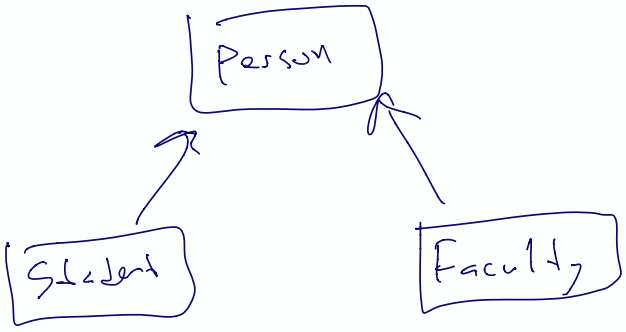
Person &r_p = s;



Student s;

Person *p_s = &s;






```
Student S;
Faculty F;

Vector<Person*> people;
people.push_back(&S);
people.push_back(&F);
```

Object slicing

```
Vector<Person> people;
people.push_back(S);
people = push_back(F);
```

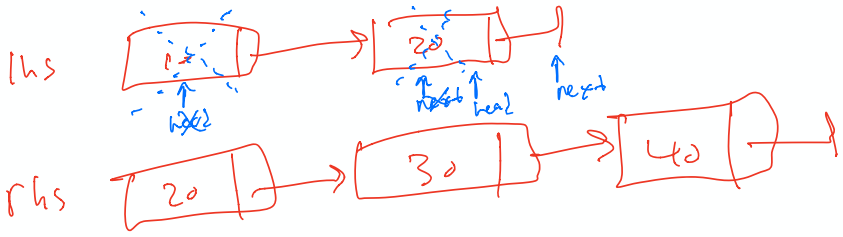
```
template <typename T>
struct Node {
  Node<T>* next;
  T data;
  Node();
};

template <typename T>
class SinglyLinkedList {
public:
  SinglyLinkedList<T>& operator=(const SinglyLinkedList<T>&);

private:
  Node<T>* head;
  Node<T>* tail;
};
```

```
template <typename T>
SinglyLinkedList<T>& SinglyLinkedList<T>::
    operator=(const SinglyLinkedList<T>& rhs)
{
  // checks for self-copy
  // deallocate lhs for lhs
  // deep copy of rhs
  // return self-reference
```

lhs = rhs;



if (this == &rhs)
    return *this;
} checks for self-copy

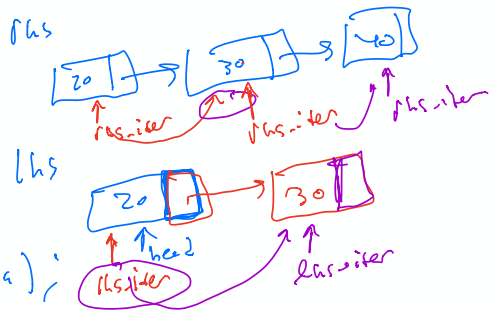
```
Node<T>* next = nullptr;
while (head)
{
    next = head->next;
    delete head;
    head = next;
}
```

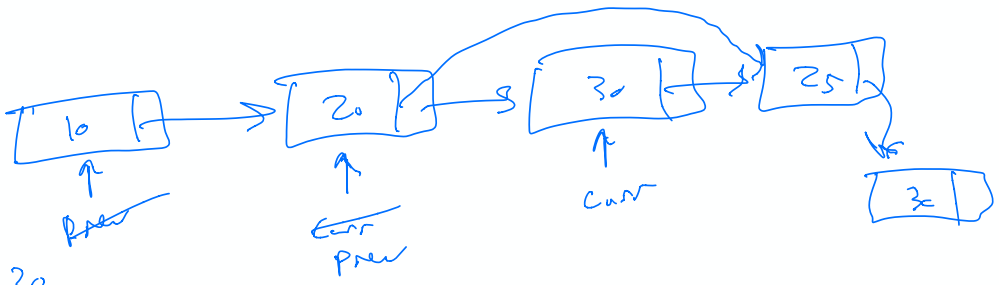
```
if (rhs.head == nullptr)
{
    head = tail = nullptr;
    return;
}

head = new Node<T>(rhs.head->data);

Node<T>* rhs_iter = rhs.head;
Node<T>* lhs_iter = head;
rhs_iter = rhs_iter->next;

while (rhs_iter != nullptr)
{
    lhs->next = new Node<T>(rhs_iter->data);
    rhs_iter = rhs_iter->next;
    lhs_iter = lhs_iter->next;
}
return *this;
```





Node* curr = head->next;
Node* prev = head;
while (curr != nullptr)
{ if (curr->data == max) prev->next = curr->next; delete curr;
else
prev = curr;
curr = curr->next;
}