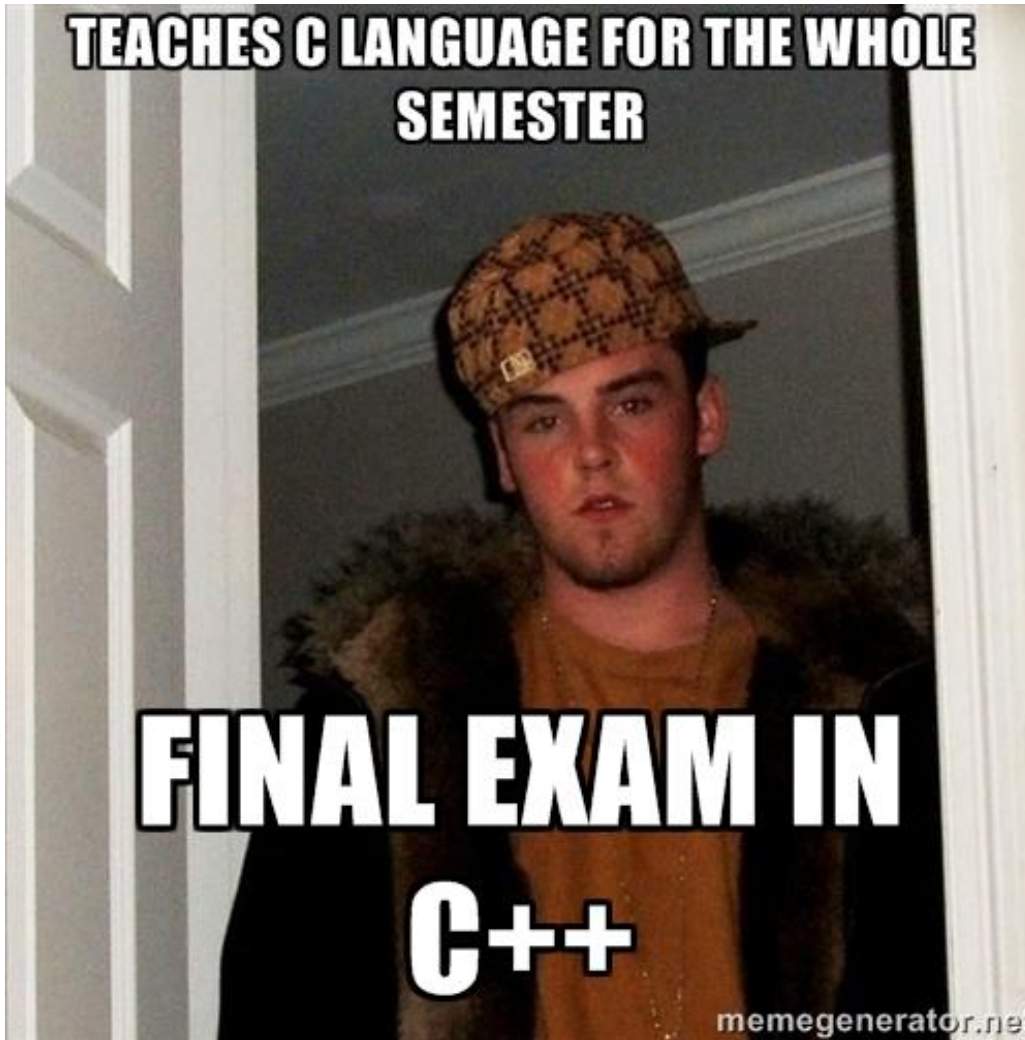


ОСНОВЫ ПРОГРАММИРОВАНИЯ НА C++

Лекция 6. Списки

Mem



Что такое список

Список - структура данных, которая предоставляет место для хранения однотипных данных в памяти. В отличие от массива, список ограничен только памятью компьютера, но плох в плане доступа к элементу по индексу и по используемой памяти. Алгоритмы добавления и удаления работают эффективнее, но некоторые алгоритмы, такие как поиск и доступ к элементу, работают медленнее. Также на элемент списка нужно больше памяти.

Типы поведения списка

Для каждого списка требуется прописать его поведение для операции. Например, куда добавлять новый элемент, как связывать списки, как удалять элемент, доступ к данным, обход списка и так далее.

Различаются списки по связям:

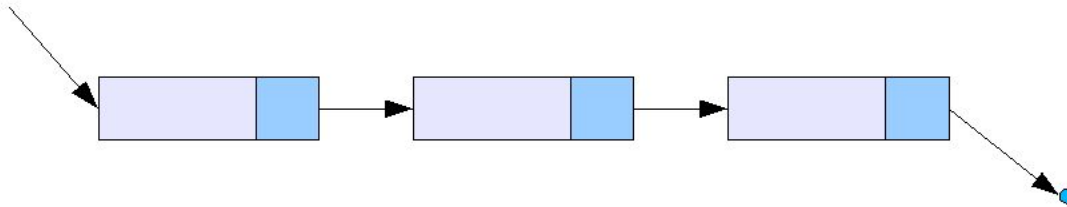
- ▶ Односвязный список
- ▶ Двусвязный список
- ▶ Кольцевой односвязный/двусвязный
- ▶ Развернутый связный

Односвязный список

Узел списка

```
struct node  
{  
  int data;  
  node *next;  
};
```

Организация списка

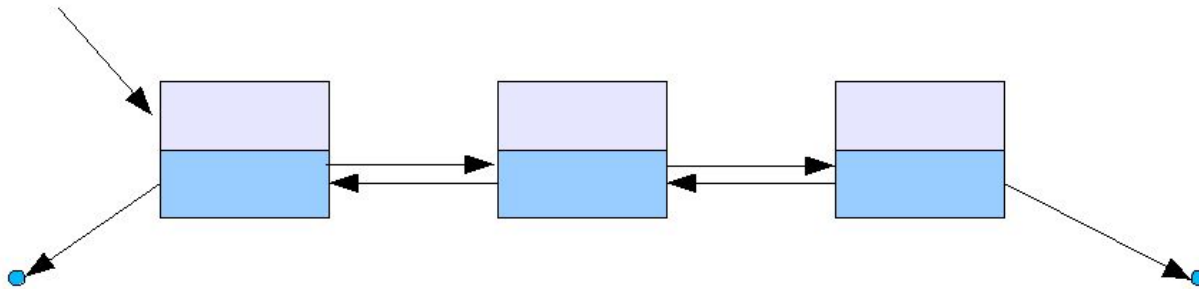


Двусвязный СПИСОК

Узел списка

```
struct node
{
    int data;
    node *next;
    node *prev;
};
```

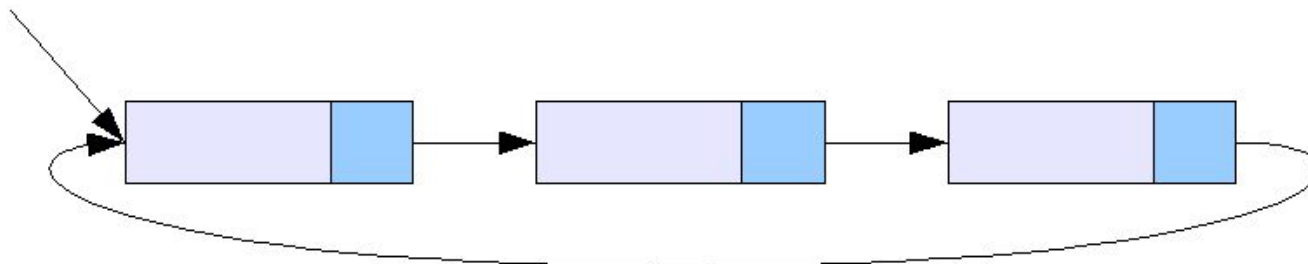
Организация списка



Кольцевой список

Узел списка
`struct node`
{
 `int data;`
 `node *next;`
 `node *prev;`
};

Организация списка



Стек

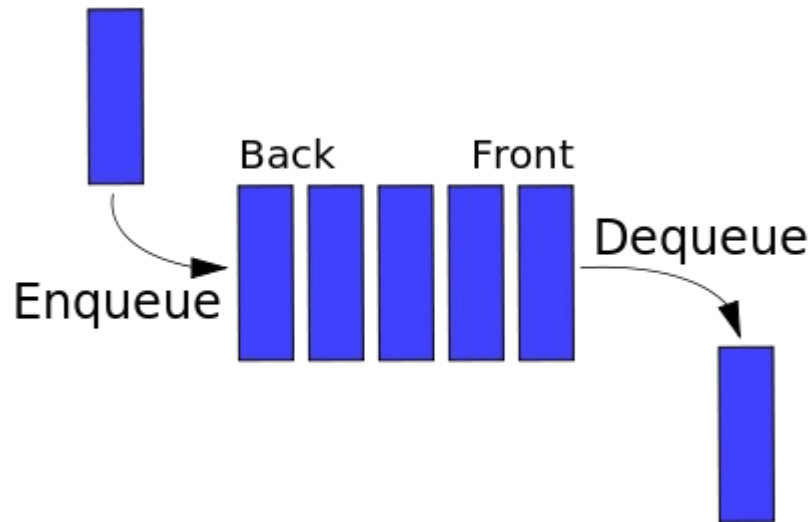
Стек - список элементов, организованных по принципу LIFO (англ. last in – first out, «последним пришёл – первым вышел»).

У стека только есть только один конец для работы. В него заносятся элементы и из него же они удаляются. Таким образом, для стека должны быть определены 2 операции: *push* и *pop*



Очередь

Очередь - список с дисциплиной доступа к элементам «первый пришёл – первый вышел» (FIFO, First In – First Out). Добавление элемента возможно лишь в конец очереди, выборка – только из начала очереди при этом выбранный элемент из очереди удаляется.



Двусторонняя очередь

Дек - список, в который элементы можно добавлять и удалять как в начало, так и в конец, то есть дисциплинами обслуживания являются одновременно FIFO и LIFO.



Основные операции, реализуемые над списком

- ▶ Опрос размера списка
- ▶ Добавление элемента
- ▶ Удаление элемента
- ▶ Вставка элемента
- ▶ Поиск
- ▶ Сортировка
- ▶ Выведение всего списка
- ▶ Очистка списка

Разберем пример: односвязный СПИСОК

```
//Структура узла
struct node
{
    int data;
    node *next;
};
```

```
//Структура списка
struct list
{
    int count;
    node *head;
    list()
    {
        count = 0;
        head = NULL;
    }
} List;
```

```
//Опрос размера списка
int Count()
{
    return List.count;
}
```

Добавление узла в начало списка

```
//Добавление в начало списка
void add(int dat){
    node *ins = new node(); //Создаем узел
    ins->data = dat;
    ins->next = NULL;
    if (List.head == NULL){ //Если список пуст
        List.head = ins; //Ставим как голову
    }
    else{ //Если не пуст
        ins->next = List.head;
        List.head = ins; //Перемещаем голову
    }
    List.count++;
}
```

Вставка по позиции

```
//Вставка на позицию
void insert(int dat, int pos){
    if (pos > Count()){ //Если места нет
        cout << "No place to insert" << endl;
        return;
    }
    node *ins = new node();
    ins->data = dat;
    //Если вставляется в начало
    if (pos == 1) { add(dat); return; }
    node *ptr = List.head;
    //Ищем место для вставки
    for (int i = 0; i < pos-1; i++, ptr = ptr->next);
    ins->next = ptr->next;
    ptr->next = ins;
    List.count++;
}
```

Удаление по позиции

```
//Удаление элемента по позиции
void remove(int pos)
{
    if (pos > Count()) { //Если места нет
        cout << "No place to delete" << endl;
        return;
    }
    if (pos == 1)
    {
        node *ptr = List.head;
        List.head = List.head->next;
        delete ptr;
        return;
    }
    node *ptr = List.head;
    for (int i = 1; i < pos-1; i++, ptr = ptr->next);
    node *delptr = ptr->next;
    ptr->next = ptr->next->next;
    delete delptr;
}
```

Показ всего списка и поиск

```
//Показ списка
void show()
{
    node *ptr = List.head;
    while (ptr != NULL)
    {
        cout << ptr->data << " ";
        ptr = ptr->next;
    }
    cout << endl;
}
```

```
//Поиск элемента по индексу
int search(int dat)
{
    node *ptr = List.head;
    int index=0;
    while (dat != ptr->data)
    {
        ptr = ptr->next;
        index++;
    }
    return index;
}
```


Мемчик в конце

