Динамические структуры данных

Лекция 5

Динамические структуры данных

Определяют логику организации хранения и обработки данных.

Динамические структуры данных

Определяют логику организации хранения и обработки данных.

Список

Стек

Очередь

Дерево

Граф

Линейные списки

Линейный список - это конечная последовательность ***однотипных*** элементов (узлов), возможно, с повторениями.

Линейные списки

Линейный список - это конечная последовательность ***однотипных*** элементов (узлов), возможно, с повторениями.

Количество элементов в последовательности называется **длиной списка**, причем длина в процессе работы программы может изменяться.

Операции со списками

- Формирование списка
- Вывод содержимого списка
- Поиск элемента в списке
- Удаление элемента из списка
- Включение элемента в список

Методы хранения линейных списков

методы последовательного хранения

методы связанного хранения

Методы хранения линейных списков

методы последовательного хранения

методы связанного хранения

элементы линейного списка размещаются **в массиве**

Методы хранения линейных списков

методы последовательного хранения

элементы линейного списка размещаются **в массиве**

методы связанного хранения

в качестве элементов хранения используются **структуры**

Линейные списки

Линейные списки

- **Односвязные**
- **Двухсвязные**

Линейные списки

Различают списки **односвязные и двухсвязные.**

Односвязные списки связаны по одному из компонентов структуры в цепочку.

Двухсвязные списки связаны по двум компонентам структуры.

Структура элемента списка

Для организации связных списков используются структуры, состоящие из двух частей: информационной и адресной.

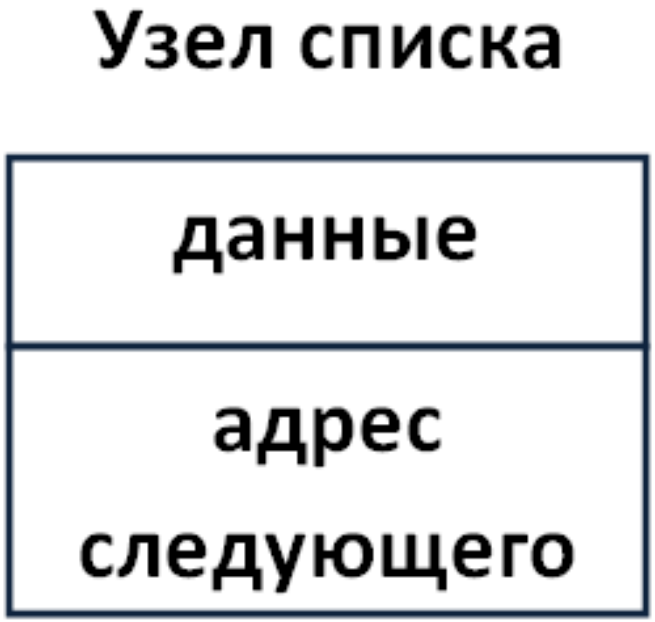
Структура элемента списка

- информационная часть содержит подлежащую обработке информацию,
- в адресной части структуры хранится адрес на соседний элемент списка.

Пример организации односвязного списка

Структура элемента списка

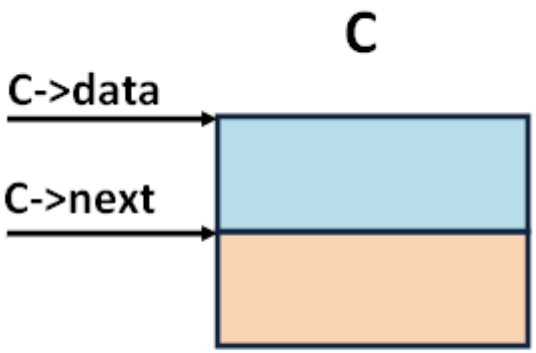
```
struct Node{
    int data;
    Node* next;
};
```



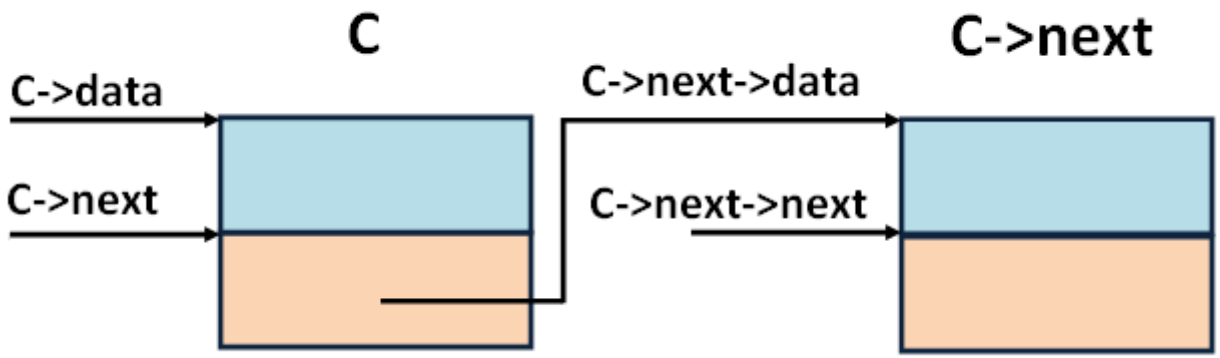
Глобальные переменные

```
Node* F=NULL; //первый элемент списка
Node* C=NULL; //текущий элемент списка
int Count=0;  //длина списка
```

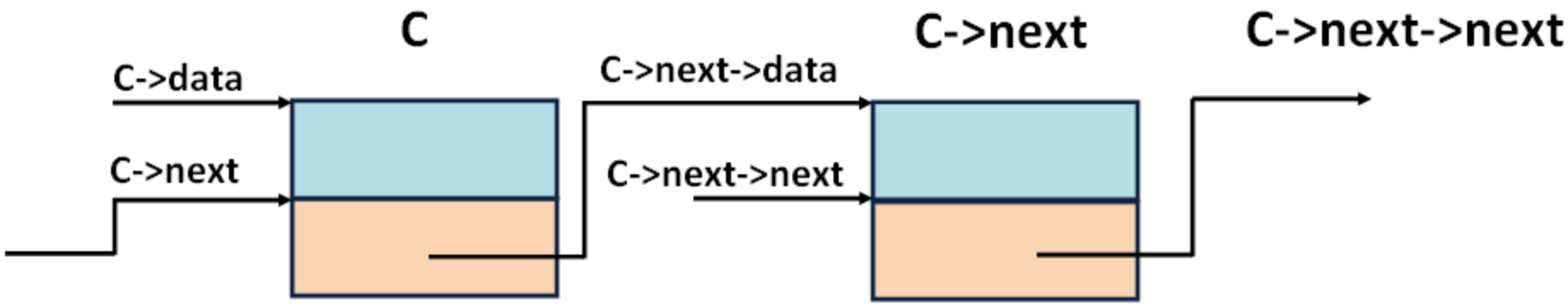
Хранение списка в памяти



Хранение списка в памяти



Хранение списка в памяти



Создание первого узла списка

Алгоритм

1. Создать первый узел
2. Заполнить все его поля
3. Первый узел сделать текущим
4. Изменить длину списка

Создание первого узла списка

```
bool CreateList(int data)
{
        F=new Node;
        F->next=NULL;
        F->data=data;
        C=F;
        Count++;
        return true;
}
```

Добавление узла после текущего

Алгоритм

1. Если текущий не выбран ->списка нет-> добавляем первый
2. Запоминаем положение следующего за текущим (**temp**)
3. После текущего создаем узел
4. Связываем его с **temp**
5. Вставленный узел делаем текущим
6. Заполняем данные текущего узла
7. Изменяем длину списка

Добавление узла после текущего

```
bool AddNext(int data)
{
    if(F==NULL)
        return CreateList(data);
    Node* temp=C->next;
    C->next=new Node;
    C->next->next=temp;
    C= C->next;
    C->data=data;
    Count++;
    return true;
}
```

Удаление узла из списка

Основные положения алгоритма:

- Удалять можно только текущий =>**После удаления любого элемента нужно изменять указатель на текущий элемент!**
- Удалять из пустого списка нечего

Возможны три разных случая:

- Удаление первого
- Удаление последнего
- Удаление текущего

Удаление узла из списка -1

```
bool Remove(int& data)
{
    if(C==NULL)
        return false;

    Node* temp;
```

Удаление узла из списка -2

```
if(C==F)
{
      temp=F;
      F=F->next;
      data=temp->data;
      delete temp;
      Count--;
      C=F;
      return true;
}
```

Удаление узла из списка -3

```
if( C->next == NULL )
{
      data=C->data;
      temp=F;
      while( temp->next->next != NULL)
            temp=temp->next;
      delete temp->next;
      temp->next=NULL;
      Count--;
      C=temp;
      return true;
}
```

Удаление узла из списка -4

```
temp=F;
data=C->data;
do{
    if( temp->next == C )
            {   temp->next=C->next;
                delete C;
                C=temp;
                Count--;
            return true;
            }
    temp=temp->next;
} while ( temp->next != NULL );

return false;
}
```

Вывод содержимого списка

```
bool Print()
{
    if (Count==0)
        cout<<endl<<"-->List is empty"<<endl;
    else
        cout<<endl<<"List:\t";
    if( F == NULL ) return false;
    Node* temp=F;
    do{
            cout<<temp->data<<" ";
            temp=temp->next;
      } while( temp != NULL );

    cout<<endl<<"count = "<< Count <<endl;
}
```

Переход на первый узел

```
bool MoveFirst()
{
     if( F == NULL ) return false;
     C=F;
   return true;
}
```

Переход на следующий узел

```
bool MoveNext()
{
    if( F == NULL ) return false;
    if( C == NULL )
            {
                    C=F;
                return true;
            }
    if(C->next == NULL) return false;
    C = C->next; return true;
}
```

Пример решения задачи со списком

Заполним список целыми числами.

Удалим все четные.

Удалим список по окончании работы.

Пример: заполнение списка

```
int i;

for( i=0; i<10; i++)
      AddNext(i);

Print();
```

Пример: удаление четных чисел из списка

```
Print();

MoveFirst();
while( C != NULL )
{
        if ( C->data % 2 == 0 )
            Remove(i);
        else if(!MoveNext())
            C=NULL;
}
Print();
MoveFirst();
```

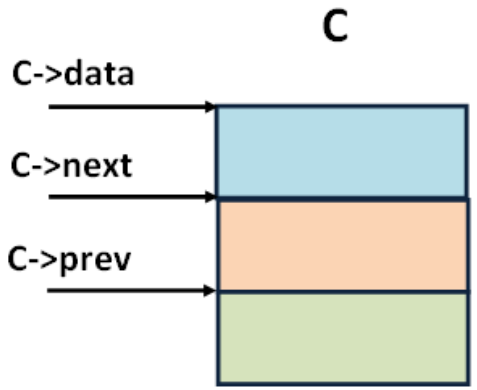
Пример: удаление списка

```
MoveFirst();
while (Remove(i));
Print();
```

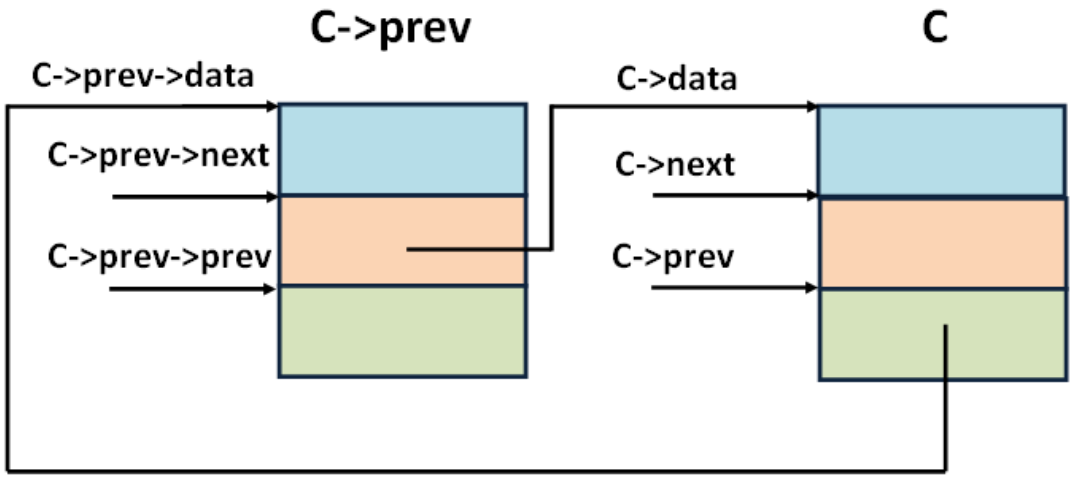
Окно запуска:

```
List:    0 1 2 3 4 5 6 7 8 9
count = 10

List:    1 3 5 7 9
count = 5

-->List is empty
```

Двухсвязный список



Двухсвязный список



Двухсвязный список