Лекция 10

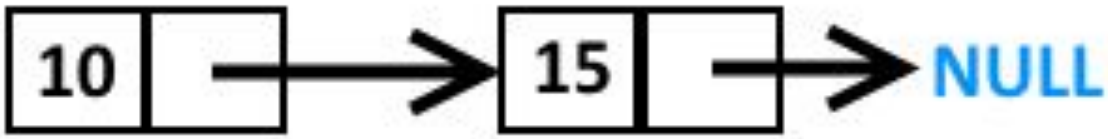
Алгоритмические языки и программирование

Сложные структуры данных
Связные списки

Часть 1

Структуры, ссылающиеся на себя

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



Связный список

- Структура данных, представляющая собой конечное множество упорядоченных элементов (узлов), связанных друг с другом посредством указателей, называется связным списком.
- Каждый элемент связного списка содержит поле с данными, а также указатель на следующий и/или предыдущий элемент. Эта структура позволяет эффективно выполнять операции добавления и удаления элементов для любой позиции в последовательности.

Недостатки связного списка

- Недостатком связного списка, как и других структур типа «список», в сравнении его с массивом, является отсутствие возможности работать с данными в режиме произвольного доступа, т. е. список – структура последовательно доступа, в то время как массив – произвольного.

Односвязный список

- Каждый узел односвязного (однонаправленного связного) списка содержит указатель на следующий узел. Из одной точки можно попасть лишь в следующую точку, двигаясь тем самым в конец. Так получается своеобразный поток, текущий в одном направлении.

Односвязный список

Каждый узел однонаправленного (односвязного) линейного списка (ОЛС) содержит одно поле указателя на следующий узел. Поле указателя последнего узла содержит нулевое значение (указывает на NULL)

Узел ОЛС можно представить в виде структуры

```
typedef struct list
{
  int field; // поле данных
  struct list *ptr; // указатель на следующий элемент
} list;
```

Односвязный список

Основные действия, производимые над элементами ОЛС:

- Инициализация списка
- Добавление узла в список
- Удаление узла из списка
- Удаление корня списка
- Вывод элементов списка

Инициализация ОЛС

Инициализация списка предназначена для создания корневого узла списка, у которого поле указателя на следующий элемент содержит нулевое значение.

```
struct list * init(int a) // a- значение первого узла
{
  struct list *lst;
  // выделение памяти под корень списка
  lst = (struct list*)malloc(sizeof(struct list));
  lst->ptr = NULL; // это последний узел списка
  lst->field = a;
  return(lst);
}
```

Добавление узла в ОЛС

Функция добавления узла в список принимает два аргумента:

- Указатель на узел, после которого происходит добавление
- Данные для добавляемого узла.

Добавление узла в ОЛС

Процедуру добавления узла можно отобразить следующей схемой:

Добавление узла в ОЛС

Добавление узла в ОЛС включает в себя следующие этапы:

- создание добавляемого узла и заполнение его поля данных;
- переустановка указателя узла, предшествующего добавляемому, на добавляемый узел;
- установка указателя добавляемого узла на следующий узел (тот, на который указывал предшествующий узел).

Добавление узла в ОЛС

Таким образом, функция добавления узла в ОЛС имеет вид:

```
struct list * addelem(list *lst, int number)
{
  struct list *temp, *p;
  temp = (struct list*)malloc(sizeof(list));
  p = lst->ptr; // сохранение указателя на следующий узел
  lst->ptr = temp; // предыдущий узел указывает на создаваемый
  temp->field = number; // сохранение поля данных добавляемого узла
  temp->ptr = p; // созданный узел указывает на следующий элемент
  return(temp);
}
```

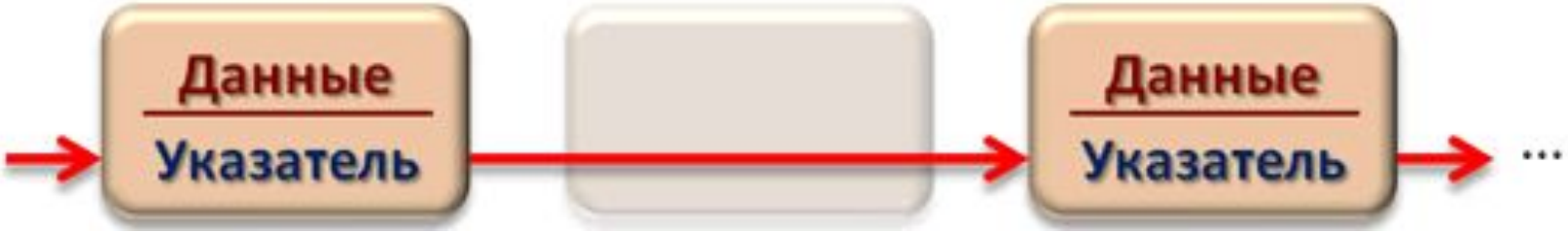
Возвращаемым значением функции является адрес добавленного узла.

Удаление узла ОЛС

- В качестве аргументов функции удаления элемента ОЛС передаются указатель на удаляемый узел, а также указатель на корень списка.
- Функция возвращает указатель на узел, следующий за удаляемым.

Удаление узла ОЛС

Удаление узла может быть представлено следующей схемой:



Удаление узла ОЛС включает в себя следующие этапы:

- установка указателя предыдущего узла на узел, следующий за удаляемым;
- освобождение памяти удаляемого узла.

Удаление узла ОЛС

Реализация удаления элемента ОЛС:

```
struct list * deletelem(list *lst, list *root)
{
  struct list *temp;
  temp = root;
  while (temp->ptr != lst) // просматриваем список начиная с корня
  { // пока не найдем узел, предшествующий lst
    temp = temp->ptr;
  }
  temp->ptr = lst->ptr; // переставляем указатель
  free(lst); // освобождаем память удаляемого узла
  return(temp);
}
```

Удаление корня списка

Функция удаления корня списка в качестве аргумента получает указатель на текущий корень списка. Возвращаемым значением будет новый корень списка - тот узел, на который указывает удаляемый корень.

```
struct list * deletehead(list *root)
{
  struct list *temp;
  temp = root->ptr;
  free(root); // освобождение памяти текущего корня
  return(temp); // новый корень списка
}
```

Вывод элементов списка

В качестве аргумента в функцию вывода элементов передается указатель на корень списка.
Функция осуществляет последовательный обход всех узлов с выводом их значений.

```
void listprint(list *lst)
{
  struct list *p;
  p = lst;
  do {
    printf("%d ", p->field); // вывод значения элемента p
    p = p->ptr; // переход к следующему узлу
  } while (p != NULL);
}
```

Лабораторные работы

Сортировка

- Напишите программу, которая из обычного массива, заполненного дробными числами, делает односвязный линейный список.
- Добавить в программу удаление всех элементов списка.

```
Входной массив: 0,50 0,90 1,90 9,90 7,00 6,90 8,70 7,20 5,50 9,00
Выходной массив: 0,50 0,90 1,90 9,90 7,00 6,90 8,70 7,20 5,50 9,00
Список очищен!



Process returned 0 (0x0)   execution time : 0.055 s
Press any key to continue.
```