Динамическое распределение памяти в С

ТЕМА 1.7

Функции распределения памяти

В языке программирования Си имеются следующие четыре функции для динамического распределения памяти, входящие в стандартную библиотеку:

- ***malloc (от англ. memory allocation, выделение памяти),***
- ***calloc (от англ. clear allocation, чистое выделение памяти)***
- ***realloc (от англ. reallocation, перераспределение памяти).***
- ***free*** *(англ. free, освободить)*

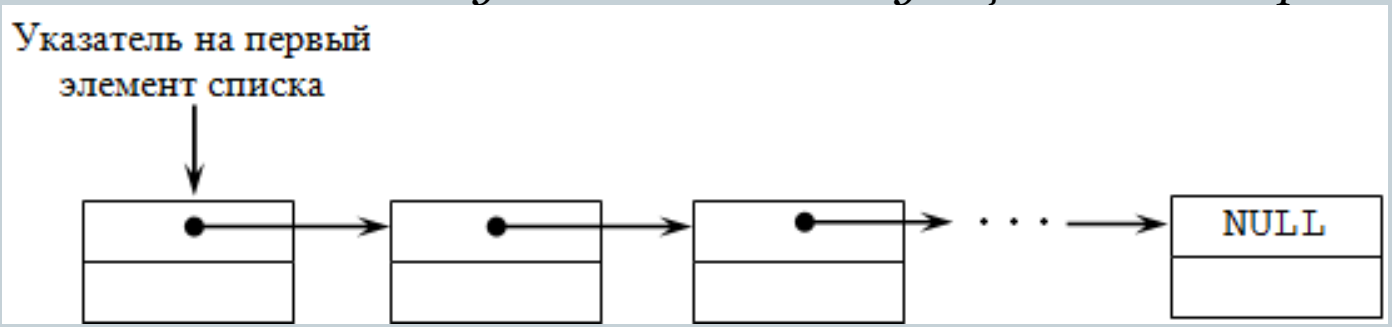
Эти функции имеют следующие описания:

```
#include <stdlib.h>

void *malloc (size_t size);
void *calloc (size_t num, size_t size);
void *realloc(void *block, size_t size);
void free(void *block);
```

Однонаправленные (односвязные) списки

Однонаправленный (односвязный) список – *это структура данных, представляющая собой последовательность элементов, в каждом из которых хранится значение и указатель на следующий элемент списка. В последнем элементе указатель на следующий элемент равен* NULL.



Пример простейшего элемента такого списка выглядит следующим образом:

```
struct Node {
              int key;//информационное поле
              Node*next;//адресное поле
            };
```

Однонаправленные (односвязные) списки

Каждый элемент списка содержит ключ, который идентифицирует этот элемент. Ключ обычно бывает либо целым числом, либо строкой.

Основными операциями, осуществляемыми с однонаправленными списками, являются:

- *создание списка;*
- *печать (просмотр) списка;*
- *вставка элемента в список;*
- *удаление элемента из списка;*
- *поиск элемента в списке*
- *проверка пустоты списка;*
- *удаление списка.*

Основные операции, осуществляемые с однонаправленными списками

- *создание списка:*

```
//создание однонаправленного списка (добавления в конец)
void Make_Single_List(int n,Single_List** Head){
  if (n > 0) {
    (*Head) = new Single_List();
    //выделяем память под новый элемент
    cout << "Введите значение ";
    cin >> (*Head)->Data;
    //вводим значение информационного поля
    (*Head)->Next=NULL;//обнуление адресного поля
     Make_Single_List(n-1,&((*Head)->Next));
  }
```

- *печать (просмотр) списка:*

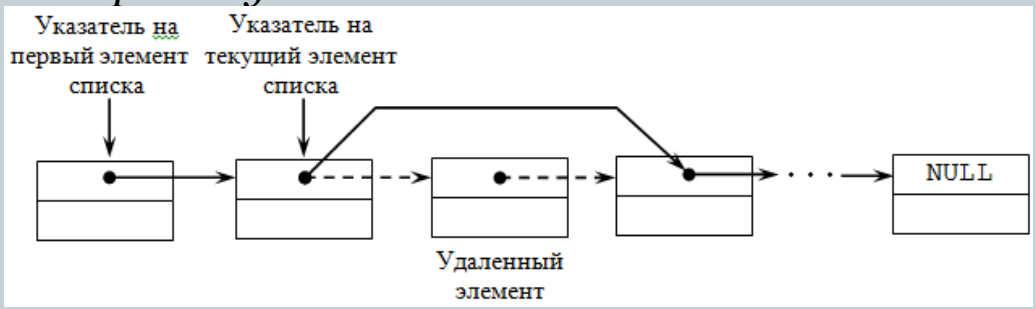
```
//печать однонаправленного списка
void Print_Single_List(Single_List* Head) {
   if (Head != NULL) {
     cout << Head->Data << "\t";
     Print_Single_List(Head->Next);
     //переход к следующему элементу
   }
   else cout << "\n";
}
```

Основные операции, осуществляемые с однонаправленными списками

- *Удаление элемента списка:*

Операция удаления элемента однонаправленного списка осуществляет удаление элемента, на который установлен указатель текущего элемента. После удаления указатель текущего элемента устанавливается на предшествующий элемент списка или на новое начало списка, если удаляется первый.

Алгоритмы удаления первого и последующих элементов списка отличаются друг от друга. Поэтому в функции, реализующей данную операцию, осуществляется проверка, какой элемент удаляется. Далее реализуется соответствующий алгоритм удаления

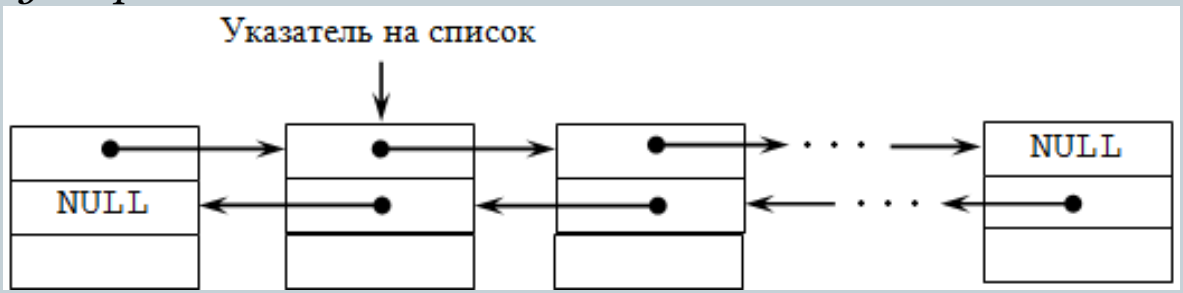



```
/*удаление элемента с заданным номером из однонаправленного списка*/
Single_List* Delete_Item_Single_List(Single_List* Head,
      int Number){
  Single_List *ptr;//вспомогательный указатель
  Single_List *Current = Head;
  for (int i = 1; i < Number && Current != NULL; i++)
    Current = Current->Next;
  if (Current != NULL){//проверка на корректность
    if (Current == Head){//удаляем первый элемент
      Head = Head->Next;
      delete(Current);
      Current = Head;
    }
    else {//удаляем непервый элемент
      ptr = Head;
      while (ptr->Next != Current)
        ptr = ptr->Next;
      ptr->Next = Current->Next;
      delete(Current);
      Current=ptr;
    }
  }
  return Head;
```

Двунаправленные (двусвязные) списки

Двунаправленный (двусвязный) список *– это структура данных, состоящая из последовательности элементов, каждый из которых содержит информационную часть и два указателя на соседние элементы. При этом два соседних элемента должны содержать взаимные ссылки друг на друга.*

В таком списке каждый элемент (кроме первого и последнего) связан с предыдущим и следующим за ним элементами. Каждый элемент двунаправленного списка имеет два поля с указателями: одно поле содержит ссылку на следующий элемент, другое поле – ссылку на предыдущий элемент и третье поле – информационное. Наличие ссылок на следующее звено и на предыдущее позволяет двигаться по списку от каждого звена в любом направлении: от звена к концу списка или от звена к началу списка, поэтому такой список называют двунаправленным.



Двунаправленные (двусвязные) списки

Пример простейшего элемента такого списка выглядит следующим образом:

```
struct list {
              type elem ;
              list *next, *pred ;
             }
list *headlist ;
```

Основными операциями, осуществляемыми с двунаправленными списками, являются:

- *создание списка;*
- *печать (просмотр) списка;*
- *вставка элемента в список;*
- *удаление элемента из списка;*
- *поиск элемента в списке;*
- *проверка пустоты списка;*
- *удаление списка.*

Основные операции, осуществляемые с двунаправленными списками

- *создание списка:*

```
//создание двунаправленного списка (добавления в конец)
void Make_Double_List(int n,Double_List** Head,
         Double_List* Prior){
  if (n > 0) {
    (*Head) = new Double_List();
     //выделяем память под новый элемент
    cout << "Введите значение ";
    cin >> (*Head)->Data;
    //вводим значение информационного поля
    (*Head)->Prior = Prior;
    (*Head)->Next=NULL;//обнуление адресного поля
    Make_Double_List(n-1,&((*Head)->Next),(*Head));
  }
  else (*Head) = NULL;
```

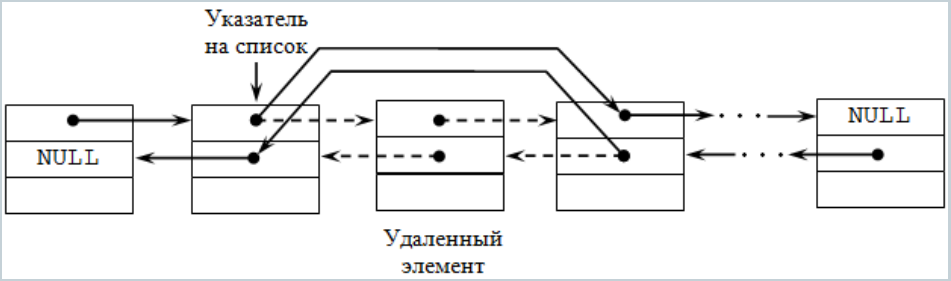
- *печать (просмотр) списка:*

```
//печать двунаправленного списка
void Print_Double_List(Double_List* Head) {
  if (Head != NULL) {
    cout << Head->Data << "\t";
    Print_Double_List(Head->Next);
    //переход к следующему элементу
  }
  else cout << "\n";
}
```

Основные операции, осуществляемые с двунаправленными списками

- *Удаление элемента списка:*

Из динамических структур можно удалять элементы, так как для этого достаточно изменить значения адресных полей. Операция удаления элемента из двунаправленного списка осуществляется во многом аналогично удалению из однонаправленного списка.




```
/*удаление элемента с заданным номером из двунаправленного списка*/
Double_List* Delete_Item_Double_List(Double_List* Head,
      int Number){
  Double_List *ptr;//вспомогательный указатель
  Double_List *Current = Head;
  for (int i = 1; i < Number && Current != NULL; i++)
    Current = Current->Next;
  if (Current != NULL){//проверка на корректность
    if (Current->Prior == NULL){//удаляем первый элемент
      Head = Head->Next;
      delete(Current);
      Head->Prior = NULL;
      Current = Head;
    }
    else {//удаляем непервый элемент
      if (Current->Next == NULL) {
      //удаляем последний элемент
        Current->Prior->Next = NULL;
        delete(Current);
        Current = Head;
      }
      else {//удаляем непервый и непоследний элемент
        ptr = Current->Next;
        Current->Prior->Next =Current->Next;
        Current->Next->Prior =Current->Prior;
        delete(Current);
        Current = ptr;
      }
    }
  }
  return Head;
```