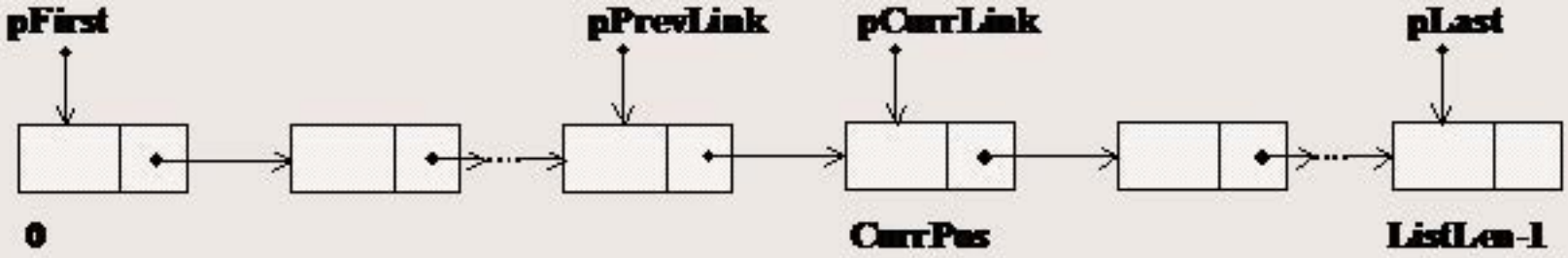
2. Проектирование структуры списка



- pFirst – указатель на первое звено списка
- pLast – указатель на последнее звено списка
- pCurrLink – указатель на текущее звено списка
- pPrevLink – указатель на звено, предшествующее текущему
- CurrPos – номер текущего звена
- ListLen – количество звеньев в списке

☑ Для повышения общности схемы реализации будем использовать вместо величины NULL константу pStop для фиксации ситуаций, в которых указатель не содержит адрес какого-либо звена списка

3. Операции для работы со списком…

- Информационные методы 
  - IsEmpty – проверить, не является ли список пустым
  - GetListLength – получить количество звеньев списка
- Методы доступа к значениям в списке 
  - GetDatValue – получить указатель на значение из звена списка (обращение возможно только в первому (FIRST), текущему (CURRENT) или последнему звеньям списка (LAST); желаемы вариант доступа задается через параметр метода)

Методы навигации по списку (*итератор*)

- Reset – установить текущую позицию на первое звено
- GoNext – переместить текущую позицию на звено вправо
- IsListEnded – проверка завершения списка (под ситуацией завершения списка понимается состояние после применения GoNext для текущей позиции, установленной на последнем звене списка, т.е. когда pPrevLink=pLast, pCurrLink=pStop)
- GetCurrentPos – получить номер текущего звена
- SetCurrentPos – установить текущую позицию на звено с заданным номером (прямой доступ к звеньям !?)

3. Операции для работы со списком…

- Вставка звеньев 
  - InsFirst – вставить звено перед первым звеном списка
  - InsLast – вставить звено после последнего звена
  - InsCurrent – вставить звено перед текущим звеном списка

☑ При выполнении операций вставки звеньев следует учитывать, что список может быть пустым

☑ После выполнения вставки необходимо обеспечить корректность значений указателей первого, текущего и последнего звеньев списка

☑ При корректировке указателей следует учитывать возможность различного положения текущей позиции списка
- текущая позиция является первым звеном (pCurrLink=pFirst),
- текущая позиция является вторым звеном (pPrevLink=pFirst),
- текущая позиция находится внутри списка,
- текущая позиция является последним звеном (pCurrLink=pLast),
- текущая позиция выходит за пределы списка (pPrevLink=pLast)

Удаление звеньев 

- DelFirst – удалить первое звено списка
- DelCurrent – удалить текущее звено
- DelList – удалить список

4. Обеспечение удобного интерфейса…

- Надежность преобразование типа
  - ☑ Указатель производного типа может автоматически приводиться к указателю на базовый тип
  - ☑ Для обратного приведения – от базового типа к производному необходимо явное преобразования типа

```
PTDatValue pValue;
PTMonom pMonom;
pMonom = PTMonom(pValue);
```

  ⇒ такое преобразование типа не является безопасным

  - ☑ Преобразование и использованием информацией о типе времени исполнения

```
pMonom = dynamic_cast<PTMonom>(pValue);
```

  ⇒ преобразование типа выполняется только если тип находится выше по иерархии наследования от объекта, указываемого pVal (иначе pMonom=NULL)

- Владение (создание и удаление) значениями
  - ☑ Использование нескольких указателей на объект усложняет контроль за его временем жизни
  - ☑ Возможное решение – передача объектов *по значению*
    - при записи в список указателя на объект в списке запоминается указатель на копию объекта-значения,
    - при получении указателя из списка создается еще одна копия объекта-значения и как результат передается указатель на новую созданную копию
  - ⇒ Возможность создания копий обеспечивает метод GetCopy

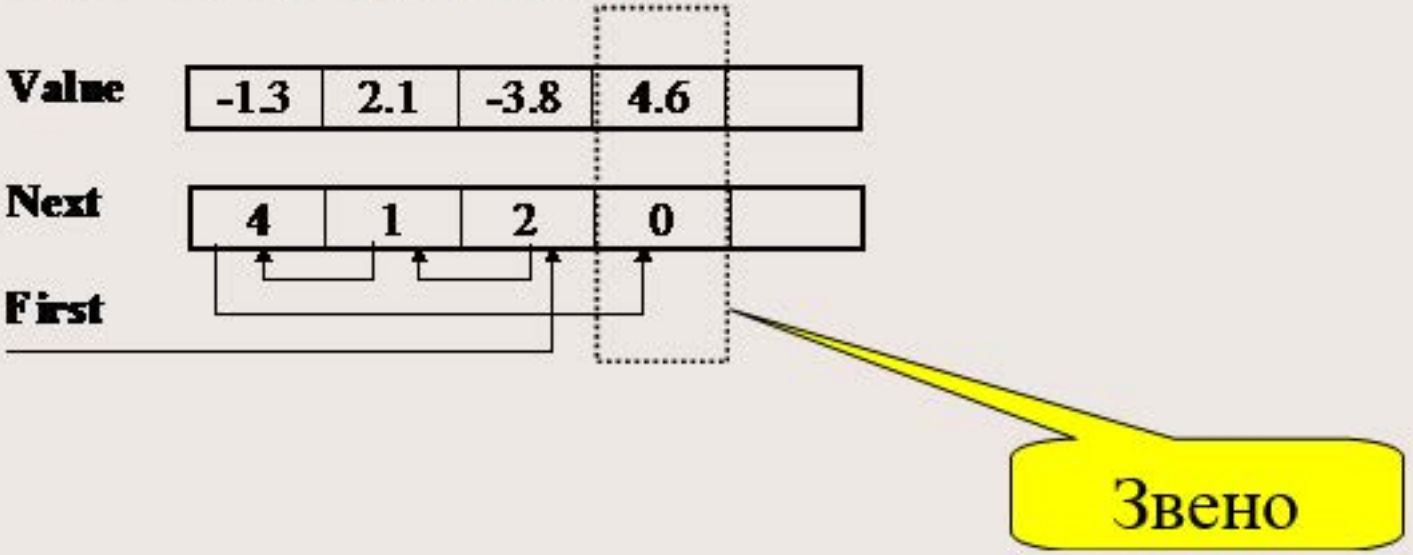
Обеспечение удобного API

- ☑ Работа с объектами является более удобным по сравнению с использованием указателей

⇒ Все перечисленные моменты (безопасное преобразование типов, создание копий, использование объектов) могут быть учтены при помощи создания шаблона класса-переходника (*proxy*) TList к классу TDatList

2. Реализация списков на языке высокого уровня…

Подход 1. Для имитации звеньев могут быть использованы два массива, один из которых используется для хранения значений, другой – для хранения индексов следующих элементов. В этом случае, звено есть элементы массивов с одинаковым индексом, адрес (имя) звена – индекс массивов.



Подход 2. С использованием ООП звено может быть представлено в виде объекта. Образ памяти, выделенной для хранения структур данных, в этом случае будет представлять массив звеньев-объектов.

```
class TLink {
  public:
    int  Value; // значение
    int  Next;  // индекс следующего звена
  protected:
    TLink();
};
TLink Mem[MemLimit];
```

1. Структура звена списка

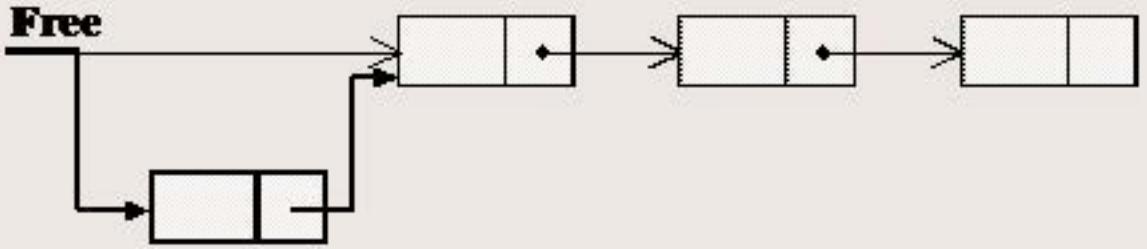
Звено списка представляется в виде объекта класса TLink

```
class TLink {
  public:
    int  Value; // значение
    int  Next;  // индекс следующего звена
  protected:
    TLink();
};
```

3. Организация списка свободных звеньев

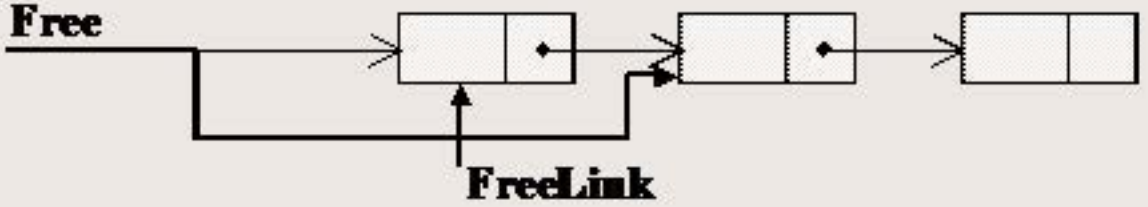
☑ Все свободные звенья объединяются в один список свободных звеньев. Звенья этого списка используются при необходимости свободной памяти, в этот список звенья должны возвращаться после освобождения.

Вставка звена в список свободных звеньев



Новое звено включается в начало списка свободных

Выборка звена из списка свободных звеньев

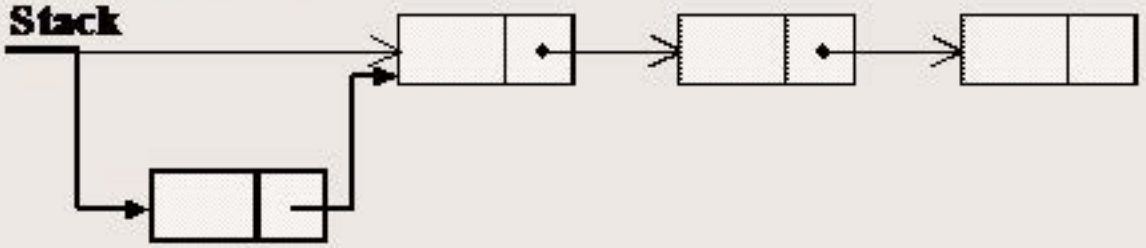


Для выделения используется первое звено списка свободных

4. Структура хранения стека

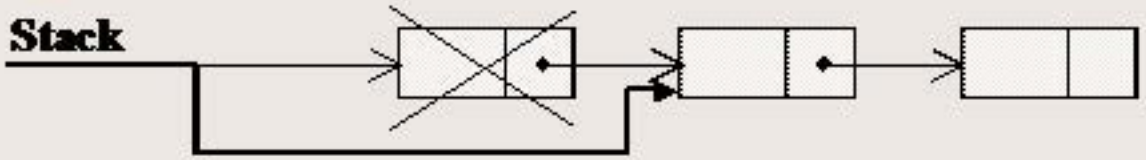
☑ Структура хранения стека – линейный список (начало списка – вершина стека).

Вставка в стек



Звено для нового значения берется в списке свободных

Выборка из стека



Исключаемое звено из стека должно оказаться в списке свободных

☑ Схема работы со стеком и со списком свободных совпадают
↳ Список свободных звеньев есть стек.

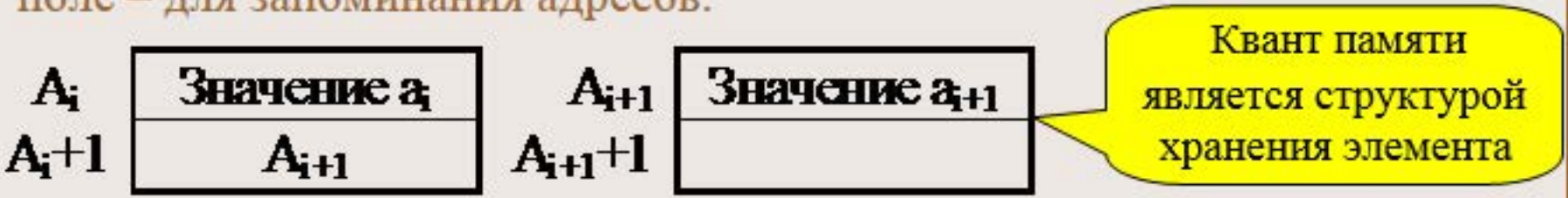
1. Представление отношений следования с использованием указателей. Понятие линейного списка...

- ☑ Необходимость перепаковки для обеспечения динамического распределения памяти возникает в силу принятого способа реализации отношения следования - следующий элемент структуры располагается в следующем элементе памяти (с адресом, большим на 1)
- ☑ Устранение перепаковки возможно только при кардинальном изменении способа реализации основных отношений – необходимо допустить размещение следующих элементов структуры в произвольных элементах памяти (там, где имеется свободные области памяти)
- Возможность такого подхода может быть обеспечена запоминанием для каждого текущего элемента структуры адреса памяти, где хранится следующий элемент
- Интерпретация содержимого элемента памяти (значение или адрес следующего элемента) в самом простом варианте может быть обеспечена фиксированным форматом используемых участков памяти

Определение 1.16. Под *квантом памяти* понимается последовательность элементов памяти с последовательно-возрастающими адресами. *Именем* (адресом) этой группы считается адрес первого слова кванта. Элементы кванта называются *полями*.

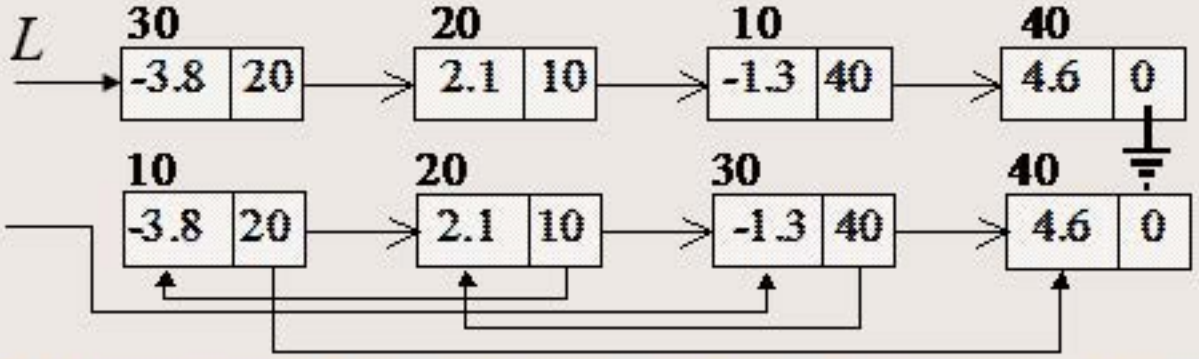
Определение 1.17. В общем случае, набор элементов памяти, связанных с одним именем, называют *звеном*.

☑ Далее будут использоваться двухэлементные звенья памяти, в которых первое поле будет использоваться для хранения значений, а второе поле – для запоминания адресов.



Определение 1.18. Способ задания отношения следования, в котором фиксация месторасположения следующего элемента производится путем запоминания соответствующего адреса памяти, называется *сцеплением* (пары, хранящие a_i и a_{i+1}, сцеплены адресными указателями).

☑ Для изображения структуры хранения с использованием сцепления звенья памяти рисуются в виде прямоугольников, а сцепление звеньев показывается в виде стрелок.



Логический порядок звеньев

Физический порядок звеньев

☑ Индикация последнего звена в списке обычно производится записью в поле адреса некоторого *барьера* – фиктивного (неадресного) значения (как правило 0 или -1).

☑ Для доступа к звеньям списка должен быть известен адрес первого звена списка. Указатель, в котором этот адрес запоминается, называется *переменной связи*.

Определение 1.19. Структура хранения данного типа (звенья, сцепление, барьер, переменная связи) называется *линейным* или *односвязным списком*.

2. Сравнение непрерывной и списковой структур хранения

	Непрерывная память	Списки
1	Перепаковка для динамического распределения памяти	Динамическое распределение памяти эффективно реализуется при помощи списка свободных звеньев
2	В структуре хранения хранятся только данные	В структуре хранения хранятся данные и указатели
3	К элементам структуры данных обеспечивается прямой доступ	К элементам структуры данных обеспечивается последовательный доступ

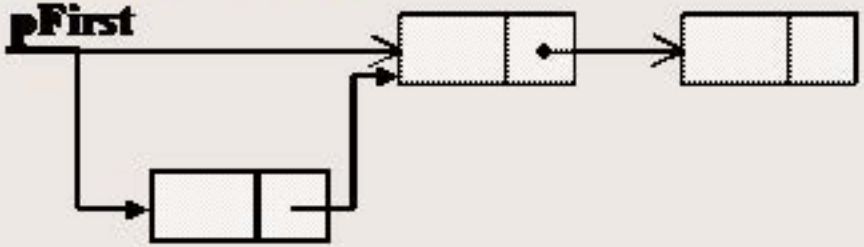
☑ Среда выполнения обеспечивает *динамически-распределяемую область памяти*:

– звено –

– выделение звена – pTemp = new TDatLink()

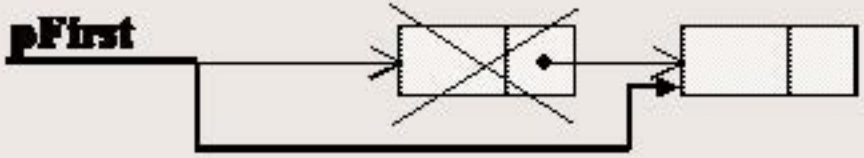
– освобождение звена – delete pTemp

Вставка в стек




```
PTDatLink pTemp;
pTemp = new TDatLink();
pTemp->SetDatValue(Val);
pTemp->SetNextLink(pFirst);
pFirst = pTemp;
```

Выборка из стека

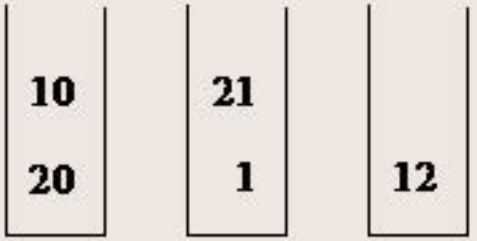



```
PTDatLink pTemp = pFirst;
Val = pFirst->GetDatValue();
pFirst = pFirst->GetNextLink();
delete pTemp;
```

4. Пример использования стеков: поразрядная сортировка…

Пример 1.9. Упорядочение данных методом поразрядной сортировки (на примере набора значений **20 12 1 21 10**)

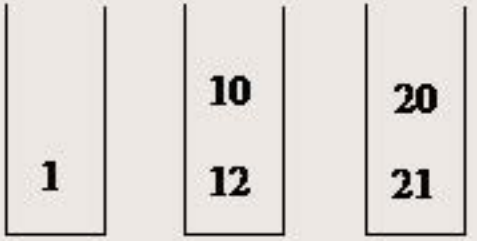
- Разложить значения по стекам (номер стека - младшая цифра)



- Собрать значения из стеков (от старшего стека)

12 21 1 10 20

- Разложить значения по стекам (номер стека - старшая цифра)



- Собрать значения из стеков (от младшего стека)

1 10 12 20 21

6. Стандартная библиотека шаблонов алгоритмического языка C++…

☑ В стандарте языка C++ предусматривается наличие в среде программирования *стандартной библиотеки шаблонов* (*Standard Template Library, STL*)

Основные понятия библиотеки STL

I. Библиотека включает в свой состав большое количество ***контейнеров***, представляющих собой структуры данных, в которых могут храниться объекты. В числе имеющихся контейнеров

- vector<T> - вектор переменного размера,
- list<T> - двусвязный список,
- queue<T> - очередь,
- stack<T> - стек,
- deque<T> - дек,
- priority_queue<T> - приоритетная очередь,
- set<T> - множество,
- multiset<T> - множество с повторением элементов,
- map<key,val> - ассоциативный массив (таблица),
- multimap<key,val> - ассоциативный массив с повторением ключей

II. Для быстрого и эффективного построения вычислительных процедур, библиотека обеспечивает ***итераторы*** для всех видов контейнеров, которые представляют унифицированный механизм последовательного доступа к элементам контейнеров.

Общая схема:

- <класс-контейнер>::iterator Iter; - объявление итератора,
- Iter = <объект-контейнер>.begin(); - установка на первый элемент,
- Iter != <объект-контейнер>.end(); - проверка на завершение,
- ++Iter – переход к следующему элементу

В зависимости от типа контейнера, итератор может обеспечивать прямой доступ, быть одно- или двух- направленным, предназначенным только для чтения или записи и др.

III. Библиотека содержит для контейнеров большое количество реализованных ***обобщенных алгоритмов***. В числе таких алгоритмов:

- for_each() - вызвать функцию для каждого элемента,
- find() - найти первое вхождение элемента,
- find_if() - найти первое соответствие условию,
- count() - подсчитать число вхождений элемента,
- count_if() - подсчитать число соответствий условию,
- replace() - заменить элемент новым значением,
- copy() - скопировать элементы,
- unique_copy() - скопировать только различные элементы,
- sort() - отсортировать элементы,
- merge() - объединить отсортированные последовательности

и др.

Пример 1.10. Преобразование выражений из инфиксной формы в польскую запись

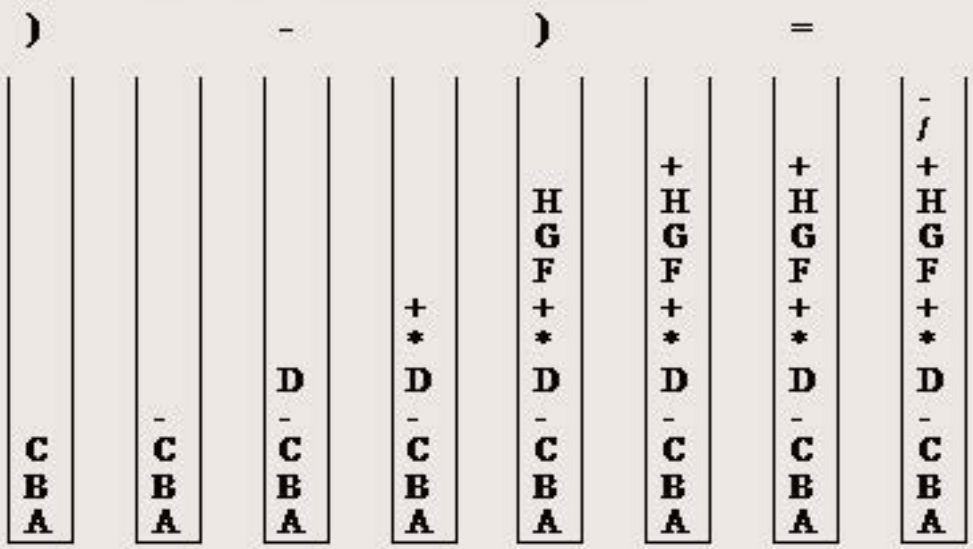
Алгоритм

1. Для операций вводится приоритет
   '*' '/' (3), '+' '-' (2), '(' (1), '=' (0)
2. Для хранения данных используется 2 стека (1 – для результата, 2 – для операций)
3. Исходное выражение просматривается слева направо
4. Операнды по мере их появления помещаются в стек 1
5. Символы операций и левые скобки помещаются в стек 2
6. При появлении правой скобки последовательно изымаются элементы из стека 2 и переносятся в стек 1. Данные действия продолжаются либо до опустошения стека 2 либо до попадания в стеке 2 на левую скобку
7. Если текущая операция, выделенная при обходе выражения, имеет меньший (более низкий) приоритет, чем операция на вершине стека 2, то такие операции из стека 2 переписываются в стек 1

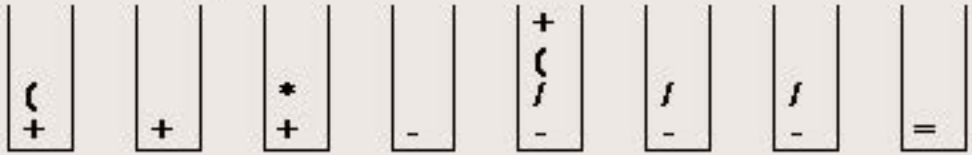
Пример 1.10. Преобразование выражений из инфиксной формы в польскую запись

Пусть выражение имеет вид A+(B-C)*D-F/(G+H)=

Стек 1 – выражение в постфиксной записи



Стек 2 - операции



программа, приложение