Тема 3. Ссылочные схемы хранения данных

Однонаправленные ссылки, стек

В предыдущих непрерывных реализациях стека, дека, очереди последовательные связи между элементами, выражаемые отношением следующий–предыдущий, естественным образом реализовывались путем размещения в соседних ячейках массива (с поправкой на технический прием кольцевого буфера). В этом случае позиция соседнего элемента могла быть вычислена по позиции текущего элемента.

Однако расплатой за непрерывный способ хранения явилось размещение в массиве и как следствие — ограниченный фиксированный объем памяти для реализации.

Можно отказаться от непрерывного размещения и выделять память для каждого отдельного элемента независимо (пока операционная система нам позволяет). Но тогда возникнет проблема как определять следующий или предыдущий, поскольку их позиции, вообще говоря, непредсказуемы, и мы их не можем вычислить. Ответ прост — раз не можем вычислить, то надо запомнить. Это приводит к понятию ссылочной реализации.

Иллюстрация: ссылочная цепочка из нескольких узловых элементов

В данном случае узловой элемент можно описать, например, такой структурой

```
template <class T>
struct ListNode
{
   T value;
   ListNode *next;
};
```

Иллюстрация: однонаправленная ссылочная цепочка

Реализация: StackList.zip — стек на базе однонаправленной ссылочной реализации
List.h — внешний дружественный класс узла
List_1.h — внутренний приватный класс узла

Итак, что было рассмотрено:
— ссылочные однонаправленные реализации линейных цепочек (на примере стека)
— внешний дружественный класс ListNode
— внутренний приватный класс ListNode
— перегрузка оператора вывода
— частные реализации шаблонных функций (например, Print для int стека)
— понятие typename (при операторе :: разрешения контекста)
Все это проиллюстрировано в коде StackList.zip