Лабораторная работа №5.

МАССИВЫ УКАЗАТЕЛЕЙ

1. Цель работы

Проверка понимания порядка построения динамических структур данных на основе массивов указателей, способа доступа к значениям компонентов, а также получение практических навыков по отладке программ, содержащих подобные структуры данных.

2. Динамические структуры данных с каталогом

Динамические структуры данных строятся с помощью стандартной операции new. Эта операция возвращает указатель, который будет определять местоположение динамического объекта и его тип. Порожденный таким образом объект обозначается в программе в виде двух составляющих: знака "*" и идентификатора указателя (рис. 1,*а*). Указатели, ссылающиеся на однотипные объекты, могут быть объединены в массив. Если все элементы этого массива будут участвовать в порождении динамических объектов, то получится структура, изображенная на рис. 1,*б*. В этой структуре можно выделить два типа объектов: каталог и динамические компоненты.

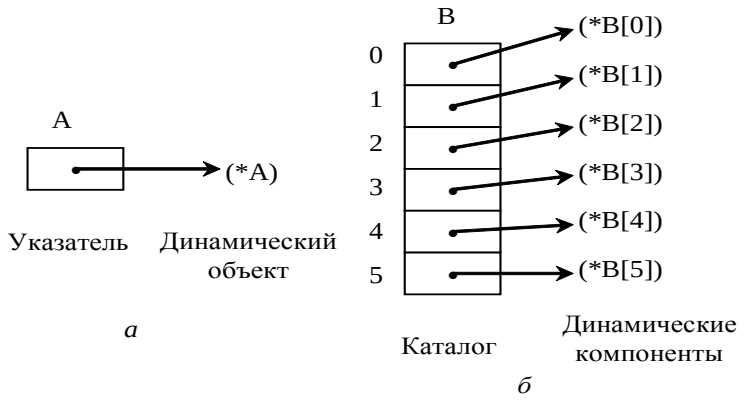

Рис. 1. Каталог динамических данных

Каталог является статическим объектом и описывается в виде массива указателей. Число элементов каталога и тип компонента, ссылка на который хранится в каталоге, фиксируется при описании переменных программы. Число динамических компонентов может меняться в процессе выполнения программы. Обозначение динамических компонентов состоит из и знака "*", идентификатора каталога и индекса элемента каталога. Например, динамический компонент, на который ссылается второй элемент каталога, будет иметь обозначение *B[1]. При таком способе обозначения динамических компонентов процесс их построения и обработки можно строить на основе операторов цикла. Ниже приведено описание на языке C++ процедуры **DM**, выполняющей построение структуры данных, изображенной на рис. 1,*б*. Порожденные этой процедурой динамические компоненты могут хранить целые числа. При этом следует подчеркнуть, что значения в компонентах не определены, для них отведена только память.

```
const int N=6;
int * B[N]; // Каталог

void DM () {
    for (int I=0; I<N; I++)
         B[I] = new int;  // Порождение компонентов
}
```

Занести значение в динамический компонент можно путём выполнения операции ввода или присваивания: *B[I]=I. Заметим, что в соответствии с приоритетами операций сначала компиля-

тор вычисляет значение выражения B[I] как элемент каталога, хранящий адрес динамического компонента. Далее с помощью операции разыменовывания (*) вычисляется значение по полученному адресу памяти B[I].

Элементы каталога при выполнении программы должны хранить либо ссылку на динамический компонент, либо значение nullptr, указывающее, что ссылка на компонент отсутствует. Пример структуры данных, где часть элементов каталога имеет значение nullptr, приведен на рис. 2,*а*.

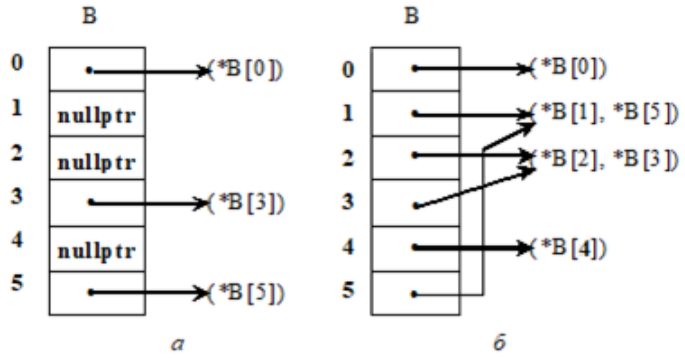

Рис. 2. Каталог с одинаковыми данными

Использование каталога особенно целесообразно для представления индексированных множеств[1], компоненты которых занимают большой объём памяти и хранят одинаковые значения. В этом случае значение можно хранить в одном компоненте, а все соответствующие элементы каталога будут ссылаться на него (см. рис. 2,*б*). Такой способ представления позволяет уменьшить затраты памяти для хранения компонентов индексированного множества.

Кроме основного каталога, используемого для построения динамических компонентов, в структуру данных могут быть введены дополнительные каталоги. С их помощью можно изменить порядок следования динамических компонентов, по сравнению с тем порядком, который установлен в основном каталоге, или выделить во множестве порожденных компонентов подмножества компонентов, обладающих определенными значениями. Например, на рис. 3 изображена структура данных, состоящая из трех каталогов.

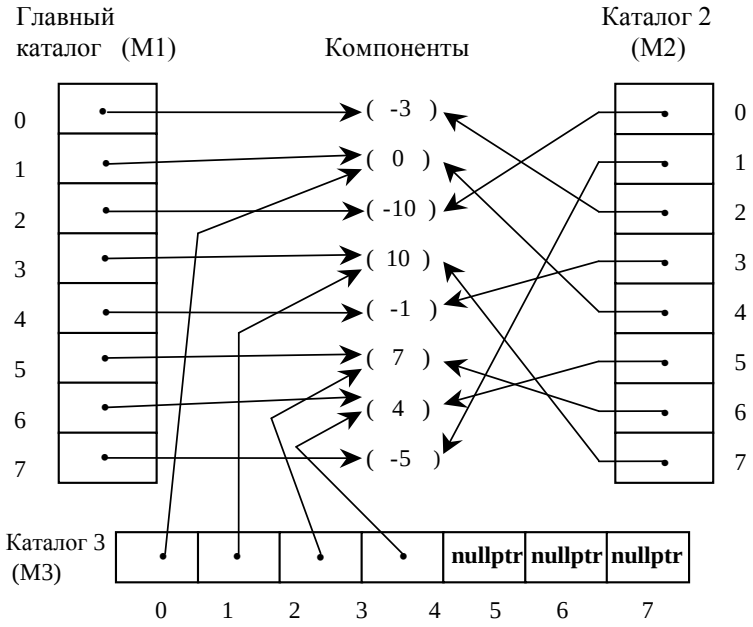

Рис. 3. Структура с тремя каталогами

[1] Индексированное множество – множество, элементами которого помечены (индексированы) элементы другого множества.

Главный каталог (M1) определяет порядок порождения динамических компонентов. Порядок следования ссылок во втором каталоге (M2) сначала такой же, как в главном каталоге, а затем он изменяется таким образом, чтобы отражать возрастающую последовательность значений динамических компонентов. В третьем каталоге (M3) имеются ссылки только на те компоненты, которые хранят целые неотрицательные числа. Процесс построения структуры данных, изображенный на рис. 3, можно описать в виде следующей программы.

```
// программа STRUCTURE
typedef int * Link;
Link M1[8], M2[8], M3[8];  // Каталоги
int i, j;
Link P;

int main() {
     // Построение компонентов и ввод их значений
     for (i=0; i<8; i++) {
         M1[i] = new int;
         cin >> *M1[i];
         M2[i] = M1[i];
     }
     // Вывод компонентов через каталог M1
     cout << "M1";
     for (i=0; i<8; i++)
         cout << setw(9) << *M1[i];
    cout << endl;

     // Упорядочивание компонентов по возрастанию
     for (j=0; j<7; j++)
         for (i=j+1; i<8; i++)
              if (*M2[i] < *M2[j]) {
                 P=M2[j];
                 M2[j]=M2[i];
                 M2[i]=P;
               }
     // Вывод компонентов через каталог M2
     cout << "M2";
     for (i=0; i<8; i++)
         cout << setw(9) << *M2[i];
    cout << endl;

     // Выделение неотрицательных компонентов
     j=0;
     for (i=0; i<8; i++)
         if (*M1[i] >= 0) {
            M3[j]=M1[i];
            j=j+1;
         }
    while (j<8) {
           M3[j]=nullptr;
           j=j+1;
    }

     // Вывод компонентов через каталог M3
     cout << "M3";
     for (i=0; i<8; i++)
         if (M3[i]!=nullptr)
              cout << setw(9) << *M3[i];
         else cout << setw(9) << "nullptr";
     cout << endl;

   return 0;
}
```

При работе со структурой данных, в которой имеются каталоги, рекомендуется придерживаться следующих правил:

1. Не оставлять без значений элементы каталога. Каждый элемент каталога должен хранить либо значение nullptr, либо ссылку на динамический компонент. Ссылка будет занесена, если элемент каталога будет параметром процедуры new или получит значение в результате присваивания от другой ссылки.

2. Проверять значения указателей на nullptr, если не все элементы каталога участвовали в порождении динамических компонентов.

Для проверки понимания способов построения и обработки динамических структур данных, созданных с помощью массивов указателей, предлагается самостоятельно проанализировать приведенную ниже программу при указанных далее исходных входных данных, дать графическую иллюстрацию структуры данных, полученной в результате ее работы, и исправить логическую ошибку при работе с динамической памятью.

```
// TEST25
#include <iostream>

typedef char * U;
U P1[27], P2[9], P3[59];
int M, K, L;

int main() {
  for (K=0; K<27; K=K+1) {
    P1[K] = new char;
    std::cin >> *P1[K];
  }

  for (K=0; K<9; K=K+1) {
    P2[K] = new char;
    std::cin >> *P2[K];
  }

  M=0;
  for (K=0; K<27; K=K+1)
    if (*P1[K] != 'P') {
      P3[M]=P1[K];
      M=M+1;
    }
    else
      for (L=0; L<9; L=L+1) {
        P3[M]=P2[L];
        M=M+1;
      };
  for (L=0; L<59; L=L+1)
    std::cout << *P3[L];
  std::cout << endl;

  return 0;
}
```

Исходные данные для анализа программы TEST25:

```
S=SQRT(P*(P-A)*(P-B)*(P-C))
(A+B+C)/2
```

3. Многоуровневые структуры данных с каталогами

Указатели, объединенные в каталог, могут хранить ссылку как на простые, так и на структурные объекты (запись, массив). Если в качестве таких объектов выступают другие каталоги,

то получаются сложные, иерархические структуры данных, у которых переход от уровня к уровню осуществляется с помощью ссылок. На рис. 4,*а* изображена трехуровневая структура данных. На ее первом уровне располагается статический каталог (D), на втором уровне размещаются динамические каталоги*D[0], *D[1], *D[2], а на третьем уровне находятся динамические компоненты, в которых содержатся значения. Для построения этой структуры данных в программе необходимо описать:

- тип компонент, хранящихся в каталоге;
- тип, определяющий структуру динамического каталога, элементы которого могут ссылаться на компоненты, хранящие значения;
- каталог D в виде массива указателей, элементы которого могут ссылаться на динамические каталоги.

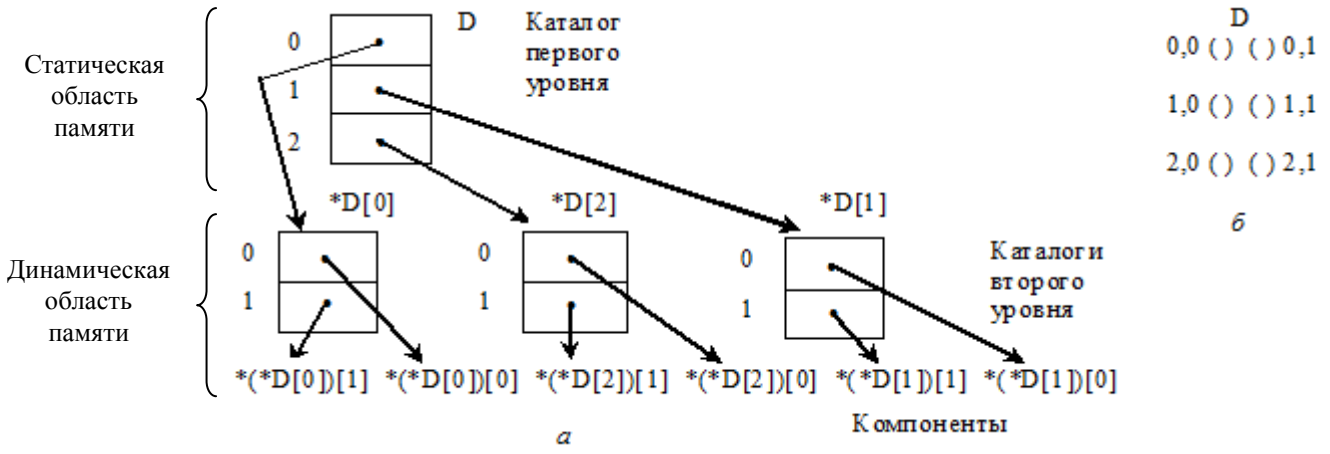

Рис. 4. Трехуровневая структура данных

Процесс построения многоуровневой структуры данных выполняется последовательно сверху вниз. Первый уровень строится компилятором по описанию статического каталога. В процессе выполнения программы с помощью операции new сначала создаются элементы второго уровня, затем третьего и т.д. С помощью операции new сначала выделяется память для хранениея элементом каталога, расположенные на более высоком уровне, чем порождаемый динамический компонент. Чтобы создать элементы третьего уровня приведенной иерархической структуры данных, необходимо включить в программу следующую операцию:

(*D[I])[J] = new float; .

Но для того, чтобы создать элемент второго уровня, являющийся массивом, память для размещения элементов которого выделяется во время выполнения программы, а не во время компиляции (*динамический массив*), необходимо включить в программу следующую операцию:

D[I] = new *имя_типа* [*количество_элементов*]; .

Как отмечалось ранее, при использовании каталога обозначение динамически порождаемых объектов состоит из знака "*", имени каталога и индекса элемента каталога. Такой способ обозначения применим как к динамическим компонентам, хранящим значения, так и к динамическим каталогам, участвующим в построении многоуровневой структуры данных. Динамические каталоги, расположенные на втором уровне иерархии, будут обозначаться в виде знака "*", имени каталога первого уровня иерархии и индекса элемента этого каталога (*D[I]). Поскольку каталоги второго уровня являются основой для построения третьего уровня иерархии, то обозначение каталога второго уровня (*D[I])[J] войдет в обозначение компонентов третьего уровня (*(*D[I])[J]) и так далее от уровня к уровню. Из этого следует, что обозначение компонента, расположенного на последнем уровне иерархии и хранящего значение, определяется значениями индексов элементов всех каталогов в соответствии с их иерархией. Такое же соответствие между индексами и компонентами, хранящими значения, имеет место в обычных массивах. Например, если компоненты обычного двухмерного массива имеют обозначение D[I][J], то компоненты трехуровневой структуры данных будут иметь обозначение *(*D[I])[J]. Эта особенность позволяет рассматривать многоуровневые структуры данных, построенные на основе ка-

талогов, как массивы с динамически порождаемыми компонентами. Отличительными чертами таких массивов являются:

1. Конфигурация массива определяется не с помощью описания переменных, а путем алгоритма порождения динамических компонентов.

2. Массив может иметь произвольную конфигурацию, т.е. не для всех индексов порождаются компоненты, хранящие значения.

3. Число компонентов массива может меняться в процессе выполнения программы.

Для примера приведем описание программы построения иерархической структуры данных, изображенной на рис. 4,*а*.

```
// program DIMST1
#include <iostream>
#include <iomanip>
using namespace std;
typedef float * DMU[2];  // Каталог 2-го уровня
DMU * D[3];      // Каталог 1-го уровня
int i, j;

int main() {
  for (i=0; i<3; i=i+1) {
    D[i] = new DMU[1]; // Порождение i-ого каталога 2-го уровня
    for (j=0; j<2; j=j+1) {
      // Порождение и ввод значения в компонент
      (*D[i])[j] = new float;
       cin >> *(*D[i])[j];
    };
  };
  // Вывод значений компонентов
  for (i=0; i<3; i=i+1) {
    for (j=0; j<2; j=j+1)
      cout << setw(7) << *(*D[i])[j] << "  ";
    cout << endl;
   }
  return 0;
}
```

Эта программа создает в памяти ЭВМ трехуровневую структуру данных, которую можно рассматривать как двухмерный массив, состоящий их трех строк и двух столбцов (рис. 4,*б*). В описании каталога первого уровня определяется число строк массива, а в описании каталога второго уровня – число столбцов массива. Поскольку все элементы каталогов первого и второго уровней участвуют в построении структуры данных, то в результате выполнения программы получается прямоугольный массив. Компоненты этого массива, хранящие значения, обозначаются в виде *(*D[I])[J].

Ссылочный характер связей в многоуровневой структуре данных позволяет порождать компоненты в любом порядке и уничтожать ранее порождаемые компоненты. Например, требуется создать трехуровневую структуру только с таким набором компонентов *(*D[0])[0], *(*D[2])[1] (рис. 5,*а*, *б*).

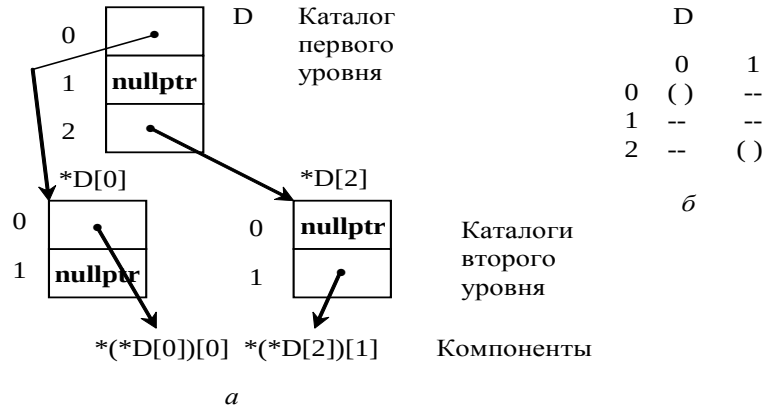

Рис. 5. Трехуровневая структура с двумя компонентами

Это построение выполняется следующей программой:

```
// program DIMST2;
#include <iostream>
#include <iomanip>
using namespace std;
typedef float * DMU[2];  // Каталог 2-го уровня
DMU * D[3];  // Каталог 1-го уровня
int i, j, k;

int main()
{
  D[0] = new DMU[1];   // Первая строка
  (*D[0])[0] = new float;   cin >> *(*D[0])[0];
  (*D[0])[1] = nullptr;
  D[1] = nullptr;      // Второй строки нет
  D[2] = new DMU[1];   // Третья строка
  (*D[2])[0] = nullptr;
  (*D[2])[1] = new float;   cin >> *(*D[2])[1];

  // Вывод
  cout << setiosflags(ios::fixed | ios::showpoint)
       << setprecision(5) << left;

  for (i=0; i<3; i=i+1)
     if (D[i] == nullptr)               // Каталога нет
        cout << "null1\n";
     else {
       for (j=0; j<2; j=j+1)
            if ((*D[i])[j] == nullptr)  // Компонента нет
                cout << setw(13) << "null2";
            else cout << setw(13) << *(*D[i])[j];
            cout << endl;
     }
  return 0;
}
```

Удалить компоненты из структуры можно путем присвоения значения nullptr указателю, в котором хранится ссылка на этот компонент. После этого компонент становится недоступным и теряется, но продолжает занимать память. Значительно грамотнее будет сначала освободить память от этого компонента, а затем присвоить указателю nullptr в знак отсутствия компонента.

Освобождение памяти, занятой компонентом, когда работа с ним завершена, реализуется с помощью операции delete. Память, которую вы возвращаете, или освобождаете, затем может быть повторно использована другими частями программы. Операция delete применяется с указателем на блок памяти, который был выделен операцией new:

```
int * ps = new int; // выделить память с помощью операции new
...                 // использовать память
delete ps;          // по завершении освободить память
                    // с помощью операции delete
```

Это освобождает память, на которую указывает ps, но не удаляет сам указатель ps. При этом возможно повторно использовать ps – например, чтобы указать на другой выделенный new блок памяти.

При работе с динамическим массивом необходимо также по завершении работы с ним освободить память с помощью операции delete []:

```
short * ps = new short [500] ;
delete [] ps;
```

Вы всегда должны обеспечивать сбалансированное применение new и delete, либо new[] и delete[]; в противном случае вы рискуете столкнуться с таким явлением, как *утечка памяти*, т.е. ситуацией, когда память выделена, но более не может быть использована. Если утечки памяти слишком велики, то попытка программы выделить очередной блок может привести к ее аварийному завершению.

При освобождении динамической памяти надо помнить, что:

– Нельзя использовать delete для освобождения той памяти, которая не была выделена с помощью new (*статические переменные*, которые известны на момент компиляции программы).

– Не использовать delete для освобождения одного и того же блока памяти дважды (результат таких попыток не определен).

– Использовать delete [], если применялась операция new[] для размещения массива.

– Использовать delete без скобок, если применялась операция new для размещения отдельного элемента.

– Помнить о том, что применение delete к нулевому указателю является безопасным (при этом ничего не происходит).

4. Индивидуальные задания

Для получения практических навыков по использованию массивов указателей при построении многоуровневых структур данных каждому студенту предлагается самостоятельно разработать программу, в которой были бы отражены следующие виды обработки:

1. Создание многоуровневой структуры данных заданной конфигурации и ввод значений в компоненты структуры.
2. Добавление нового компонента в заданное место структуры и ввод в него значения.
3. Удаление указанного компонента из структуры и освобождение памяти.
4. Изменение значения для заданного компонента.
5. Уничтожение всех динамических компонентов и проверка освобождения памяти.
6. Вывод значений и порядковых номеров (не индексов!) всех имеющихся в структуре компонентов после любой ее модификации (5 раз).

Конфигурации многоуровневых структур данных представлены на рис. 6, 7. Они задаются в виде не полностью заполненных прямоугольных матриц, в которых указаны: начальный вид конфигурации многоуровневой структуры, место расположения нового компонента, какой компонент должен быть удален и в каком компоненте должно быть изменено значение. Индексы элементов матрицы и порядковые номера компонентов многоуровневой структуры должны соответствовать друг другу.

Тип компонента иерархической структуры данных выбирается следующим образом:

для матриц с номерами 1, 5, 9, 13, 17, 21, 25, 29 - целый;

для матриц с номерами 2, 6, 10, 14, 18, 22, 26, 30 - вещественный;

для матриц с номерами 3, 7, 11, 15, 19, 23, 27 - символьный;

для матриц с номерами 4, 8, 12, 16, 20, 24, 28 - булевский.

Элементы матрицы имеют следующие обозначения:

( ) - элемент входит в начальную конфигурацию;

(+) - элемент, который добавляется в структуру;
(-) - элемент входит в начальную конфигурацию, а затем удаляется;
(*) - элемент входит в начальную конфигурацию и меняет свое значение;
пустота - отсутствие элемента в конфигурации (память не выделяется).

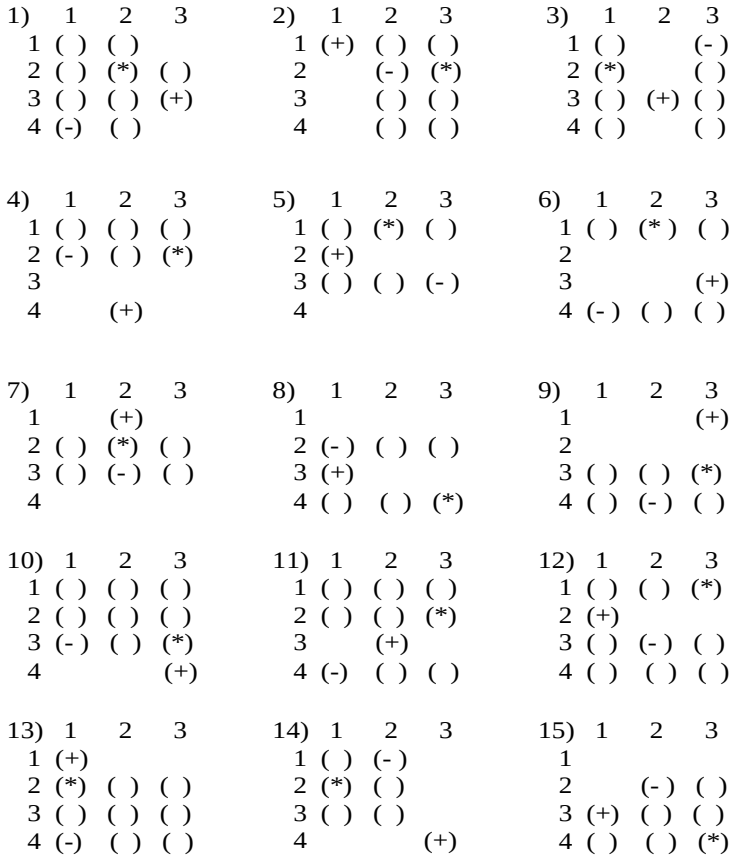

Рис.6. Конфигурации структур данных

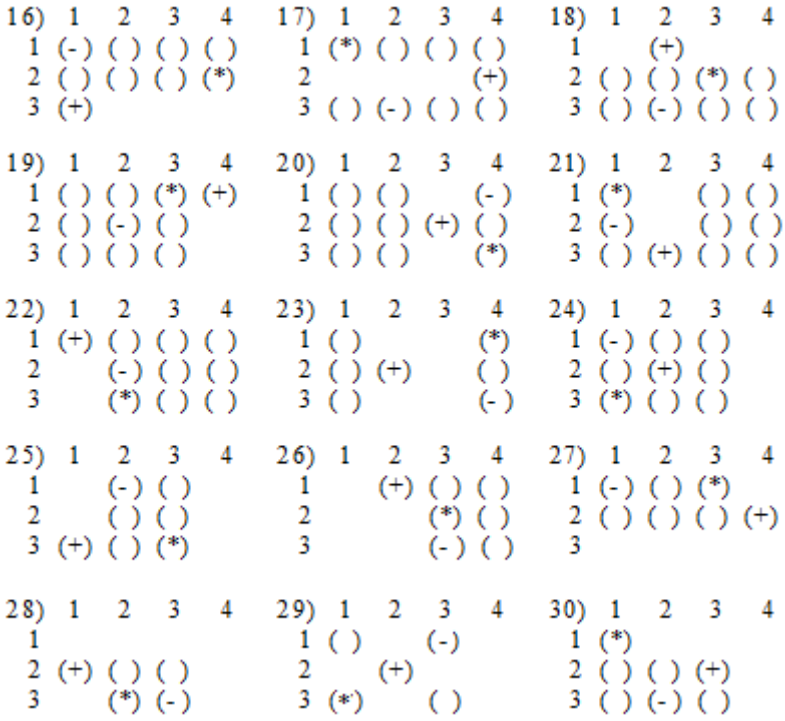

Рис. 7. Конфигурации структур данных

5. Подготовка к работе

1. Изучить описание работы, структуры данных и работу с ними. Разобраться в приведенных в описании примерах программ.

2. Нарисовать структуру данных, полученную при анализе программы TEST25 (по аналогии с рис. 3) по окончании её работы.

3. Для указанной в индивидуальном задании структуры данных продумать способ ее обработки и изобразить структуры каталогов (по аналогии с рис. 4), отражающие различные этапы его обработки (создание, удаление и добавление компонета), с указанием местоположения объектов (статическая/динамическая область памяти).

4. Разработать программу в соответствии с заданием.

5. Подготовить в письменной форме заготовку отчета по работе.

6. Порядок выполнения работы

1. Ввести переменные и операторы для создания структуры данных, ввода значений компонент и вывода содержимого структуры. Проверить работоспособность. Вывод содержимого структуры организовать с нумерацией строк и компонентов структуры.

2. Добавить добавление нового компонента. Проверить.

3. Добавить изменение компонента. Проверить.

4. Добавить удаление компонента. Проверить.

5. Добавить уничтожение всех динамических переменных из структуры. Проверить.

6. Записать в отчет результаты проверки программы.

7. Содержание отчета

1. Текст исправленной программы TEST25 с указанием исправленных ошибок, значения входных данных и изображение структуры данных, полученной в результате ее работы.

2. Индивидуальное задание на составление программы построения многоуровневой структуры данных.

3. Графическое изображение многоуровневой структуры данных индивидуального задания-по примеру рис. 4.

4. Материалы в соответсвии с подготовкой к работе (п.2 и 3).

5. Текст программы на языке C++, отражающий указанные в задании виды обработки.

6. Результаты выполнения работы.

8. Пример программы

Задание. Разработать программу для обработки не полностью определенного двумерного массива булевских данных с помощью двухуровневой структуры с указателями:

	1	2	3	4
1	( )		(-)	
2		(+)		
3	(*)		( )	

Программа должна иметь структуру:

1. Создание структуры данных и ввод булевых значений.

2. Добавление нового (+) компонента ввод в него значения.

3. Удаление компонента (*) из структуры и освобождение памяти.

4. Изменение значения для компонента (-).

5. Уничтожение всех динамических компонентов.

6. Вывод значений, номеров строк и колонок для всех имеющихся в структуре компонентов после любой ее модификации (5 раз).

```
// program Ptr_Arr25;
#include <iostream>
#include <iomanip>
using namespace std;
typedef
  bool * Arr4Ptr[4];   // Массив из 4-х указателей - компоненты строки

Arr4Ptr *Arr3Ptr[3];   // Массив из трех указателей - строки
short int Row, Col, Val;

void OutArr();   // Вывод не полностью определенного массива

int main() {
  cout << "Содержимое структуры по умолчанию\n";
  OutArr();

  cout << "\nСоздание структуры и ввод данных\n";
  cout << "Булевские значения ввести как 0 и 1: \n";
  for (Row=0; Row<3; Row=Row+1)
    if (Row == 1) // 2-я строка пуста
      Arr3Ptr[Row] = nullptr;
    else {
      cout << setw(3) << "Ряд " << Row << ". "; // Номер строки
      // Создаем массив из 4-х указателей}
      Arr3Ptr[Row] = new Arr4Ptr [1];
      for (Col=0; Col<4; Col=Col+1)
        if (Col%2!=0) // Столбцы 2 и 4 пусты
          (*Arr3Ptr[Row])[Col] = nullptr;
        else {        // Col = 0, 2}
          // Создать компонент для булева значения
          (*Arr3Ptr[Row])[Col] = new bool;
          cin >> Val;   // Вводим значения 0 или 1
          // Преобразуем в тип Boolean
          *(*Arr3Ptr[Row])[Col] = Val;
        };
    };
  cout << "\nСодержимое памяти после создания"
       << " структуры и ввода данных\n";
  OutArr();

  cout << "\nВведите компонет [1][1]: ";
  // Создаем массив из 4-х указателей для строки 2
  Arr3Ptr[1] = new Arr4Ptr [1];
  for (Col=0; Col<4; Col=Col+1)
    if (Col==1) {
      (*Arr3Ptr[1])[Col] = new bool; // Создаем компонент
      // Вводим значение и неявно преобразуем его к типу bool
      cin >> Val;
      *(*Arr3Ptr[1])[Col] = Val;
    }
    else // Col = 0,2,3
      // Остальные три компонента отсутствуют
      (*Arr3Ptr[1])[Col] = nullptr;
  cout << "Содержимое памяти после добавления"
       << " компонента\n";
  OutArr();

 cout << "\nУдаляем компонент [0][2]\n";
  // Освободить память от компонента
  delete (*Arr3Ptr[0])[2];
  // Указатель обнулить в знак отсутствия компонента
  (*Arr3Ptr[0])[2] = nullptr;
  cout << "Содержимое памяти после удаления "
```

```
        << "компонента\n";
  OutArr();

  cout << "\nМеняем компонет [2][0] \n";
  // Просто вводим новое значение и преобразуем тип
  cin >> Val;
  *(*Arr3Ptr[2])[0] = Val;
  cout << "Содержимое памяти после изменения"
       << " компонента\n";
  OutArr();

  cout << "\nОсвобождаем всю занятую динамическую память\n";
  // Для всех строк
  for (Row=0; Row<3; Row++)
    // Если строка содержит компоненты
    if (Arr3Ptr[Row] != nullptr) {
      // Для всех компонентов строки
      for (Col=0; Col<4; Col++)
        // Если компонент имеется
        if ((*Arr3Ptr[Row])[Col] != nullptr) {
          // Уничтожить его
           delete (*Arr3Ptr[Row])[Col];
          // И установить признак его отсутствия
          (*Arr3Ptr[Row])[Col] = nullptr;
        };
      // Освободить память от массива из 4-х указателей
      delete [] Arr3Ptr[Row];
      Arr3Ptr[Row] = nullptr; // Ряд стал пуст
    };
  cout << "\nВывод для проверки - всё ли освободили?\n";
  OutArr();

  cin.get();
  return 0;
}

 // Вывод не полностью определенного массива
void OutArr() {
  char Row, Col;
  cout << setw(6) << "29)  ";  // Номер варианта задания
  // Вывод номеров компонентов строки - номера столбцов
  for (Col=0; Col<4; Col++)
      cout << setw(6) << int(Col);
  cout << endl;
  // Для всех строк
  for (Row=0; Row<3; Row++) {
    cout << setw(6) << int(Row);  // Номер строки}
    // Если компонентов в строке нет
    if (Arr3Ptr[Row] == nullptr)
      cout << setw(10) << "Ряд пуст";
    else
      // Для всех компонентов строки
      for (Col=0; Col<4; Col+=1)
        if ((*Arr3Ptr[Row])[Col] == nullptr)
          cout << setw(6) << "null";  // Компонент отсутствует
        else
          // Значение компонента
          cout << setw(6) << *(*Arr3Ptr[Row])[Col];
    cout << endl;
  };
  return;
}
```