Язык программирования Си

Бикулов Д.А., Иваницкая Н.В., Иванов А.П.

Семинар 6. Указатели. Передача параметра в функцию по указателю. Функции работы с динамической памятью. Динамические массивы. Указатель на функцию.

1 Указатели

Указатель — переменная, в которой хранится адрес ячейки в памяти. В следующем примере создается переменная **value**, потом создается указатель **pointer**, в который записывается адрес ячейки памяти, в которой лежит начало блока данных переменной **value**. Показан вывод значения указателя, **разыменования** указателя (выводится значение переменной, содержащейся по адресу, записанному в указатель) и вывод первого элемента массива, на начало которого указывает указатель (массив из одной переменной в данном случае).

```
#include <iostream>
using namespace std;

int main()
{
    int value = 2;
    int *pointer = &value;

    cout << value << endl;
    cout << pointer << endl;
    cout << *pointer << endl;
    cout << pointer[0] << endl;

    return 0;
}
// Вывод:
// 2
// 0x7fff122b471c
// 2
// 2
```

2 Передача параметра в функцию по указателю

Передача параметра функции по указателю нужна, когда в качестве аргумента функции выступает массив или вы не хотите использовать ссылку. Указатель так же бывает константным и при попытке изменения переменной, на которую ссылается константный указатель, вы получите ошибку компиляции. Пример из раздела выше, но переписанный для использования указателей. Используется взятие адреса переменной & (амперсанд):

```
#include <iostream>
using namespace std;

int func(int *a) {
    *a = 2*(*a);
    int b = (*a)+1;
    return b;
}
```

```
int main()
{
    int value = 2;

    cout << value << endl;

    int result = func(&value);
    cout << value << endl;
    cout << result << endl;

    return 0;
}
// Вывод:
// 2
// 4
// 5
```

3 Функции работы с динамической памятью

Функция **malloc()** позволяет динамически выделять блок памяти под массив элементов. В качестве аргумента принимает длину блока в памяти, который надо выделить, в **байтах.** Динамически выделенную память необходимо очищать после работы с ней. Для этого в функцию **free()** необходимо передать указатель на выделенную ранее память. При выделении памяти также стоит проверять успешность выделения. Так, если в качестве указателя на выделенный объем памяти функцией **malloc()** возвращен **NULL**, то выделение памяти завершилось ошибкой. Пример выделения и освобождения памяти под динамический массив из 1000 элементов. На момент компиляции длину массива знать необязательно. Функция выделения памяти возвращает указатель типа **void ***, поэтому требуется ручное приведение типа к желаемому.

```
#include <iostream>
#include <stdlib.h>
using namespace std;

int main()
{
    int *array;

    size_t N = 1000;
    array = (int *)malloc(N * sizeof(int));
    if (array == NULL) {
        cout << "Allocation error" << endl;
        return 1;
    }

    array[5] = 34;
    cout << array[5] << endl;

    free(array);

    return 0;
}
```

Функция **realloc()** позволяет изменить размер выделенной памяти. При этом содержимое старой памяти сохраняется вплоть до последнего элемента в случае, если выделено больше памяти или вплоть до последнего, умещающегося в новый массив памяти. Пример использования показан ниже. Функция возвращает указатель типа **void ***, поэтому требуется ручное приведение типа к желаемому.

```
#include <iostream>
#include <stdlib.h>
using namespace std;

int main()
{
    int *array;

    size_t N = 1000;
    array = (int *)malloc(N * sizeof(int));
    if (array == NULL) {
        cout << "Allocation error" << endl;
        return 1;
    }

    array[5] = 34;
    array = (int *)realloc(array, 100);
    cout << array[5] << endl;

    free(array);

    return 0;
}
```

Пример. Функция, которая изменяет размер переданного динамического массива:

```
#include <iostream>
#include <stdlib.h>

using namespace std;

void myIntResize(int **a, size_t N) {
      *a = (int*)realloc(*a, N * sizeof(int));
}

int main()
{
    int *array;
    array = (int*)malloc(5 * sizeof(int));
    array[4] = 7;
    myIntResize(&array, 10);
    cout << array[4] << endl;
    free(array);
    return 0;
}
```

4 Указатель на функцию

```
#include <iostream>
using namespace std;

double myFunc(double x)
{
    return (10.0 * x + 1.2);
}

double mySum(double a, double b, double(*func)(double))
{
    return (func(a) + func(b));
}
```

```
int main()
{
    double m = 2.3;
    double n = 1.8;

    cout << myFunc(m) << endl;
    cout << myFunc(n) << endl;
    cout << mySum(m, n, myFunc) << endl;
    return 0;
}
// Вывод:
// 24.2
// 19.2
// 43.4
```

5 Контейнеры

5.1 Вектор

Вектор — это динамический массив, который сам следит за своим размером. Необходимо реализовать функции, эмулирующие работу вектора (приведены заголовки функций для массива чисел двойной точности):

- добавление элемента в конец вектора:

```
void push_back(double **array, size_t &size, size_t &capacity,
               const double &element);
```

- удаление элемента из конца вектора:

```
void pop_back(double **array, size_t &size);
```

- вставка элемента на позицию **`ind`**:

```
void insert(double **array, size_t &size, size_t &capacity,
            size_t ind, const double &element);
```

- получение элемента на позиции **`ind`**:

```
double at(double **array, size_t ind);
```

- изменение размера вектора:

```
void resize(double **array, size_t &size, size_t &capacity,
            size_t new_size);
```

- очистка памяти:

```
void destruct(double **array);
```

5.2 Очередь

Очередь — это линейный список элементов, предоставляющий тип доступа к своим элементам LILO (Last In Last Out): **последний** добавленный элемент будет **последним** отданным. То есть, если в очередь положить список чисел `1 2 3 4`, то, забирая эти числа из очереди, вы получите `1 2 3 4`. Необходимо реализовать функции, эмулирующие работу очереди (приведены заголовки функций для массива целых чисел):

- добавить элемент в очередь:

```
void push(int **queue, size_t &size, size_t &capacity,
          const int &element);
```

- получить элемент из очереди:

```
int top(int **queue, size_t &size);
```

- удалить элемент из очереди:

```
void pop(int **queue, size_t &size);
```

- очистка памяти:

```
void destruct(int **queue);
```

5.3 Стек

Стек — это линейный список элементов, предоставляющий тип доступа к своим элементам LIFO (Last In First Out): **последний** добавленный элемент будет **первым** отданным. То есть, если в стек положить список чисел `1 2 3 4`, то, забирая эти числа из стека, вы получите `4 3 2 1`. Необходимо реализовать функции, эмулирующие работу стека (приведены заголовки функций для массива чисел одинарной точности):

- добавить элемент в стек:

```
void push(int **stack, size_t &size, size_t &capacity,
          const float &element);
```

- получить элемент из стека:

```
float top(int **stack, size_t &size);
```

- удалить элемент из стека:

```
void pop(int **stack, size_t &size);
```

- очистка памяти:

```
void destruct(int **stack);
```

Типовое задание: написать программу, разбитую на два модуля компиляции, в одном – набор функций, реализующих динамический контейнер (например, вектор) со всей описанной выше функциональностью, а также с функцией вывода элементов контейнера (для контейнеров, хранящих тип **`char`** – без пробелов), во втором модуле – тестовая программа к реализованному контейнеру.

1. Вариант

Реализовать **стек** для элементов типа **`double`** со стратегией увеличения ёмкости на константу (константа задается через **`#define`**).

2. Вариант

Реализовать **стек** для элементов типа **`int`** со стратегией удвоения ёмкости.

3. Вариант

Реализовать **вектор** для элементов типа **`int`** со стратегией увеличения ёмкости на константу (константа задается через **`#define`**).

4. Вариант

Реализовать **вектор** для элементов типа **`char`** со стратегией удвоения ёмкости.

5. Вариант

При помощи одного динамического массива реализовать **очередь** для элементов типа **`char`** со стратегией увеличения ёмкости на константу (константа задается через **`#define`**).

6. Вариант

При помощи одного динамического массива реализовать **очередь** для элементов типа **`double`** со стратегией удвоения ёмкости.

7. Вариант

Реализовать **отсортированный вектор** для элементов типа **`double`** со стратегией удвоения ёмкости, который при вставке элементов сохраняет свою сортировку.

8. Вариант

Реализовать **двунаправленную очередь** для элементов типа **`double`** со стратегией удвоения ёмкости. Элементы в двунаправленной очереди можно добавлять как в начало, так и в конец.

9. Вариант

Реализовать сортировку массива целых чисел без его изменения: создается дополнительный массив указателей на элементы исходного массива. Дальше производится сортировка массива указателей, без изменения исходного массива.

10. Вариант

Есть два массива: массив «ключей» и массив «значений». Также есть третий массив, в котором попарно хранятся указатели на соответствующие элементы этих двух массивов. Необходимо реализовать функцию сортировки пар в третьем массиве по возрастанию ключей, а также функцию добавления новой пары ключ-значение ко всей коллекции (с изменением всех трех массивов).

11. Вариант

Есть два массива: массив «ключей» и массив «значений». Также есть третий массив, в котором попарно хранятся указатели на соответствующие элементы этих двух массивов. Необходимо реализовать функцию удаления пар ключ-значение с дубликатами ключей: для двух пар с одинаковыми ключами, оставляется та пара, у которой значение больше.

12. Вариант

Реализовать **двумерную матрицу**, для которой реализованы функции взятия элемента по двум индексам (номер строки и номер столбца) и возможностью вставить как новый столбец, так и новую строку. Матрицу реализовать как массив указателей на массивы, хранящие ее строки.

13. Вариант

Реализовать **двумерную матрицу**, для которой реализованы функции взятия элемента по двум индексам (номер строки и номер столбца) и возможностью вставить как новый столбец, так и новую строку. Матрицу реализовать посредством одного динамического массива.

14. Вариант

Реализовать **двумерную матрицу**, для которой реализованы функции взятия элемента по двум индексам (номер строки и номер столбца) и возможностью удалить как указанный столбец, так и указанную строку. Матрицу реализовать как массив указателей на массивы, хранящие ее строки.

15. Вариант

Реализовать **двумерную матрицу**, для которой реализованы функции взятия элемента по двум индексам (номер строки и номер столбца) и возможностью удалить как указанный столбец, так и указанную строку. Матрицу реализовать посредством одного динамического массива.

16. Вариант

Реализовать функцию, которая перемножает две матрицы и возвращает результат в создаваемой третьей матрице. Матрицы реализовать как массивы указателей на массивы, хранящие их строки. Для всех матриц реализовать функции взятия элемента по двум индексам (номер строки и номер столбца).

17. Вариант

Реализовать четыре функции, представляющие собой четыре арифметических действия с другими неизвестными функциями, представленными указателями, а также функцию, реализующую суперпозицию двух неизвестных функций, представленных указателями. В качестве элементарных функций использовать: **`sin(), cos(), log(), exp(), sqrt()`**. По запросу пользователя – построить сложную функцию, реализующую запрошенное функциональное выражение и посчитать значение получившейся функции в заданной точке.

18. Вариант

Дан одномерный массив. Реализовать функцию, которая строит и возвращает гистограмму значений из этого массива в массиве с заданным количеством ячеек, а также параллельный массив указателей на массивы элементов, попавших в каждую из ячеек гистограммы.

19. Вариант

Реализовать **очередь** на основе двух динамических массивов. Реализовать функции добавления в конец, добавления в начало и взятия элемента по индексу из этой пары массивов. Будем считать, что первый из двух массивов хранит начало очереди в обратном порядке – то есть, последний элемент массива является головным элементом очереди, предпоследний вторым и т.д. Элементом очереди, следующим за нулевым элементом первого массива является нулевой элемент второго массива, следом – первый элемент второго массива и т.д. до последнего элемента, второго массива, который является последним элементом очереди. Каждый из массивов содержит заранее выделенный запас элементов, который позволит быстро добавлять и удалять элементы в начало и в конец очереди. При взятии элемента по индексу рассчитывается «виртуальный» индекс на основе первого и второго массива: если требуемый «виртуальный» индекс **`i`** больше размера первого массива, то возвращается (**$i-dim_1$**) элемент второго массива. Если же меньше, то возвращается (**dim_1-i**) элемент первого массива, где **dim_1** – количество элементов в первом массиве.

20. Вариант

Работа с **разреженными матрицами**. Дана двумерная вещественная матрица, многие элементы которой являются нулями. Написать функцию для преобразования такой матрицы в упакованный набор: массив значений, массив указателей, переводящий двумерные индексы к позиции в массиве значений, при этом элементы матрицы, по модулю меньшие 10^{-10}, не сохраняются в упакованный набор. Также написать функцию, которая возвращает **`[i][j]`**-элемент матрицы и функцию печати такой матрицы в обычном двумерном виде. При заполнении матрицы воспользоваться генератором случайных чисел – сделать так, чтобы примерно каждый третий элемент матрицы был равен нулю.

21. Вариант

Реализовать работу с **квадратными верхними треугольными матрицами**: создание, удаление, умножение, взятие **`[i][j]`**-элемента матрицы и вычисление определителя. Нулевые элементы из нижней треугольной области не хранить.

22. Вариант

Реализовать работу с **ленточными матрицами**: создание, удаление, умножение, взятие **`[i][j]`**-элемента матрицы. Нулевые элементы из верхней и нижней треугольных областей не хранить.

23. Вариант

Реализовать сортированную очередь из строк символов. Пользователь вводит текстовые строки, которые добавляются в массив указателей, отсортированный по алфавиту. Сначала пользователь добавляет пять строк подряд, затем после каждой добавляемой строки печатается и удаляется одна строка из головы очереди. Программа завершается после ввода слова «**`end`**».

24. Вариант

Сортировка слиянием. Дан массив, длина которого равна степени числа 2. Разделить массив пополам на два новых массива. Слить эти два массива в исходный массив, соблюдая порядок сортировки в парах «элемент из первого массива»-«элемент из второго массива». Получившийся массив опять разделить пополам и слить два массива в исходный, соблюдая порядок сортировки для четверок элементов. Далее повторить операции для восьмерок элементов и т.д., пока весь массив не окажется отсортированным. В конце – удалить служебные массивы.

25. Вариант

Реализовать **двумерную матрицу** произвольной размерности, хранящую отдельные биты (так называемый bitmap) в виде битового вектора. Биты хранить упакованными по восемь бит – в байтах. Реализовать функцию, которая возвращает `[i][j]`-бит из матрицы в виде булевского значения. Реализовать функции вставки новых строки и колонки бит.