Динамические массивы

Методы тестирования программ

Варшавский Павел Романович

к.т.н., доцент кафедры ПМ

Список литературы по курсу

1. Князев А.В. Основы языка C++. Учебное пособие. –М.: Издательство МЭИ, 2013 – 80 с. ISBN 978-5-7046-1425-8.

2. Князев А.В. Работа со сложными структурами данных на языке C++. Учебное пособие. –М.: Издательство МЭИ, 2015 – 48 с. ISBN 978-5-7046-1658-0.

3. Программирование. Сборник задач. Учебное пособие. –Санкт-Петербург: Лань, 2019 – 140 с. ISBN 978-5-8114-3857-0.

URL: https://e.lanbook.com/book/121485

ДИНАМИЧЕСКИЕ МАССИВЫ

МЕТОДЫ ТЕСТИРОВАНИЯ ПРОГРАММ

Динамическое выделение памяти необходимо для эффективного использования памяти компьютера.

При **статическом выделении памяти** для работы с массивом необходимо объявить массив и задать ему фиксированный размер (например, из 100 элементов – **int x[100];**).

В данном случае можно работать с массивом размером **не более 100 элементов** и **значительно снижается эффективность использования памяти**, если требуется например всего 10 элементов, так как в памяти выделяется место под 100 элементов.

Для динамического распределения памяти используются операции **new** и **delete**.

Операция **new** выделяет память из области свободной памяти.

```
int *x = new float [100];
```

Операция **delete** освобождает выделенную память.

```
delete [] x;
```

Выделяемая память, после её использования должна освобождаться, поэтому операции **new** и **delete** используются парами.

ПРИМЕР (задача 5.2 вариант №1**)**

Даны два массива $C_1, C_2, \ldots, C_n$ и $P_1, P_2, \ldots, P_n$. Если каждый элемент первого массива меньше суммы элементов второго, найти, при каких значениях i и j максимально значение выражения $C_i / (P_j + C_i^2)$.

```
void InputMas (int m, double *A);
double SumP (int m, double *B);
void Check (int m, double *A, double *B, bool &f);
void Findij (int m, double *A, double *B, int &k, int &q);
```

```
void main ()
{
    int n, i, j;
    bool fl;
    cout<<“Введите количество элементов в массивах”;
    cin>>n;
    double *C = new double [n];
    double *P = new double [n];
    InputMas (n, C);          // Подзадача 1.1
    InputMas (n, P);          // Подзадача 1.2
    Check (n, C, P, fl);      // Подзадача 2
    if (fl==true)
    {
        Findij (n, C, P, i, j);          // Подзадача 3
        cout<<i<<j<<endl;
    }
    else  cout<<“Условие не выполнено”<<endl;
    delete [] C;
    delete [] P;
}
```

Объявление двумерного динамического массива (матрицы)

```
// объявление двумерного динамического массива (4x5) из 20 элементов:
    float **A = new float* [4]; // 4 строки
    for (int i = 0; i < 4; i++)
        A[i] = new float [5]; // 5 столбцов
```

A – массив указателей на выделенный участок памяти под массив вещественных чисел (float).

float **A – указатель второго порядка, который ссылается на массив указателей float* [4], где размер массива равен четырём.

Освобождение памяти, отводимой под двумерный динамический массив (матрицу)

```
...
for (int i = 0; i < 4; i++)
        delete [] A[i];
  //освобождает память, выделенную для массива значений
```

```
        delete [] A;
//освобождает память, выделенную под массив указателей
```

Передача матрицы в подпрограмму

```
void InputMatr (int n, int m, float **A); // Прототип
                                          // функции
     InputMatr (4, 5, A); // Вызов функции в основной
                          // программе main()
```

МЕТОДЫ ТЕСТИРОВАНИЯ ПРОГРАММ

Тестирование является одним из наиболее устоявшихся способов обеспечения качества разработки программ.

С технической точки зрения **тестирование** заключается в выполнении программы на некотором множестве исходных данных и сверке получаемых результатов с заранее известными (эталонными).

Тестирование характеризуется достаточно большой трудоемкостью в процессе разработки программы (до 40%).

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Отладка (debug, debugging) – процесс поиска, локализации и исправления ошибок в программе.

Тестирование обеспечивает выявление (констатацию наличия) фактов расхождений с требованиями (ошибок).

Как правило, на фазе тестирования осуществляется и исправление идентифицированных ошибок, включающее локализацию ошибок, нахождение причин ошибок и соответствующую корректировку программы.

Реализация тестирования разделяется на три этапа:

– **Создание тестового набора** (test suite) путем ручной разработки или автоматической генерации для конкретной среды тестирования.

– **Прогон программы на тестах**, управляемый тестовым монитором (test driver) с получением протокола результатов тестирования (test log).

– **Оценка результатов выполнения программы на наборе тестов** с целью принятия решения о продолжении или остановке тестирования.

Основная проблема тестирования – определение достаточности множества тестов для истинности вывода о правильности реализации программы, а также нахождения множества тестов, обладающего этим свойством.

ВОСХОДЯЩЕЕ ТЕСТИРОВАНИЕ

Восходящий подход предполагает, что каждая подпрограмма тестируется отдельно на соответствие имеющимся спецификациям, а затем оттестированные подпрограммы собираются в более крупные (более высокий уровень иерархии), которые затем тестируют.

Данный подход обеспечивает полностью автономное тестирование, для которого просто генерировать тестовые последовательности.

Недостатки восходящего тестирования

Во-первых, при восходящем тестировании так же, как при восходящем проектировании, серьезные ошибки в спецификациях, алгоритмах и интерфейсе могут быть обнаружены только на завершающей стадии работы над проектом.

Во-вторых, для того, чтобы тестировать подпрограмму необходимо разработать специальные тестирующие программы, которые обеспечивают вызов подпрограмм с необходимыми параметрами. Причем эти тестирующие программы также могут содержать ошибки.

НИСХОДЯЩЕЕ ТЕСТИРОВАНИЕ

Автономно тестируется только основная программа.

При её тестировании все вызываемые подпрограммы заменяют «заглушками», которые в той или иной степени имитируют поведение вызываемых подпрограмм.

В отличие от тестирующих программ заглушки очень просты, например, они могут просто фиксировать, что им передано управление.

Часто заглушки просто возвращают какие-либо фиксированные данные.

Как только тестирование основной программы завершено, переходят к тестированию подпрограмм, вызываемых основной, с заменой соответствующих заглушек, а затем проводят их совместное тестирование. Далее последовательно переходят к следующим подпрограммам, пока не будет протестирована вся программа целиком.

Основной недостаток нисходящего тестирования – отсутствие автономного тестирования подпрограмм.

Основным достоинством данного метода является ранняя проверка основных решений и качественное многократное тестирование сопряжения подпрограмм.

Презентации лекций на сайте кафедры ПМ

http://www.appmat.ru

в разделе СТУДЕНТУ>УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ>МАТЕРИАЛЫ ПО БКП> МАТЕРИАЛЫ ДЛЯ ИЭТ

http://www.appmat.ru/материалы-для-иэт/