МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра «Информационные системы»

отчет

по лабораторной работе №3

по дисциплине «Программирование»

Тема: "Функции, рекурсивные функции и указатели"

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Землянский Д.А. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

Цель работы.

Создание, передача аргументов и рекурсивные функции.

Знакомство с указателями. Их понятие, операции и арифметика указателей.

Основные теоретические положения.

Использование функций позволяет:

1)      значительно упростить разработку сложных программ;

2)      сократить объем текста программы и генерируемого результирующего кода программы;

3)      значительно упростить отладку и модификацию программ;

4)      распределить работу над одной программой между различными исполнителями программистами.

Фактически разработка более-менее сложных программ практически невозможна без использования функций

***Передача данных по значению***

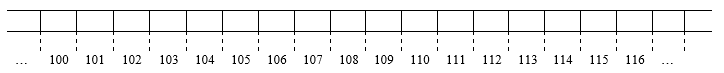
Механизм передачи данных через параметры функции очень прост. При вызове функции в определенной области памяти (в стеке программы) для каждого параметра функции создается переменная соответствующая типу данных параметра. В эти переменные копируются значения аргументов, использовавшихся при вызове функции. При выполнении кода функции эти копии значений аргументов могут использоваться для обработки, могут изменять свои значения, но эти изменения никак не затрагивают значений самих аргументов. Поэтому после завершения работы функции, значения аргументов, которые были использованы при вызове функции, останутся такими же, какими они были до вызова функции.

### Рекурсивное использование функций

Функции внутри своего тела могут вызывать сами себя. Такой вызов называется **рекурсией**. На основе рекурсии можно строить очень интересные алгоритмы обработки данных.

Указатели и ссылки являются одними из самых важных и достаточно сложных для понимания и использования средств языка программирования. Они ориентированы на прямую работу с памятью компьютера. С помощью этих средств реализуется работа с динамической памятью и динамическими объектами, возвращение из функций измененных данных и многое другое. К использованию указателей и ссылок мы будем неоднократно возвращаться в последующих разделах.

Все данные (переменные, константы и др.) хранятся в памяти. Память представляет собой непрерывную последовательность ячеек (байтов), каждая из которых имеет свой номер – адрес:



Указатели поддерживают ряд операций: присваивание, получение адреса указателя, получение значения по указателю, некоторые арифметические операции и операции сравнения.

### Присваивание

Указателю можно присвоить либо адрес объекта того же типа, либо значение другого указателя.

### Нулевые указатели

Нулевой указатель (null pointer) - это указатель, который не указывает ни на какой объект. Если мы не хотим, чтобы указатель указывал на какой-то конкретный адрес, то можно присвоить ему условное нулевое значение.

### Ссылки на указатели

Так как ссылка не является объектом, то нельзя определить указатель на ссылку, однако можно определить ссылку на указатель.

### Разыменование указателя

Операция разыменования указателя представляет выражение в виде \*имя\_указателя.

### Адрес указателя

Указатель хранит адрес переменной, и по этому адресу мы можем получить значение этой переменной. Но кроме того, указатель, как и любая переменная, сам имеет адрес, по которому он располагается в памяти. Этот адрес можно получить также через операцию &

### Операции сравнения

К указателям могут применяться операции сравнения >, >=, <, <=,==, !=. Операции сравнения применяются только к указателям одного типа и к значениям NULL и nullptr.

### Приведение типов

Иногда требуется присвоить указателю одного типа значение указателя другого типа. В этом случае следует выполнить операцию приведения типов с помощью операции (тип\_указателя \*)

К указателям можно применять некоторые арифметические операции. К таким операциям относятся:  **+**,**-**, **++**, **--**. Результаты выполнения этих операций по отношению к указателям существенно отличаются от результатов соответствующих арифметических операций, выполняющихся с обычными числовыми данными.

Постановка задачи.

1)    Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка N (6,8,10) случайными числами от 1 до  N\*N согласно схемам, приведенным на рисунках. Пользователь должен видеть процесс заполнения квадратичной матрицы.



2)    Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами:



3)    Используя арифметику указателей, сортирует элементы любой сортировкой.

4)    Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.

Приложение А

Полный код программы

#include <iostream>

#include <cmath>

void printMatrix(const int \* const \* matrix, const int matrixSize) {

for (int i = 0; i < matrixSize; ++i) {

for (int j = 0; j < matrixSize; ++j) {

std::cout << matrix[i][j] << "\t";

}

std::cout << std::endl;

}

std::cout << "----------------------------------------------" << std::endl;

}

void spiralFill(int \*\*& matrix, const int matrixSize) {

int currentRowStart = 0, currentColumnStart = 0;

int currentRowEnd = matrixSize, currentColumnEnd = matrixSize;

while (currentRowStart < currentRowEnd && currentColumnStart < currentColumnEnd) {

for (int i = currentColumnStart; i < currentColumnEnd; ++i) {

matrix[currentRowStart][i] = std::rand() % (matrixSize \* matrixSize);

std::cout << "Added element with indexes [" << currentRowStart << ";" << i << "]" << std::endl;

printMatrix(matrix, matrixSize);

}

currentRowStart++;

for (int i = currentRowStart; i < currentRowEnd; ++i) {

matrix[i][currentColumnEnd - 1] = std::rand() % (matrixSize \* matrixSize);

std::cout << "Added element with indexes [" << i << ";" << currentColumnEnd - 1 << "]" << std::endl;

printMatrix(matrix, matrixSize);

}

currentColumnEnd--;

if (currentRowStart < currentRowEnd) {

for (int i = currentColumnEnd - 1; i >= currentColumnStart; --i) {

matrix[currentRowEnd - 1][i] = std::rand() % (matrixSize \* matrixSize);

std::cout << "Added element with indexes [" << currentRowEnd - 1 << ";" << i << "]" << std::endl;

printMatrix(matrix, matrixSize);

}

currentRowEnd--;

}

if (currentColumnStart < currentColumnEnd) {

for (int i = currentRowEnd - 1; i >= currentRowStart; --i) {

matrix[i][currentColumnStart] = std::rand() % (matrixSize \* matrixSize);

std::cout << "Added element with indexes [" << i << ";" << currentColumnStart << "]" << std::endl;

printMatrix(matrix, matrixSize);

}

currentColumnStart++;

}

}

}

void snakeFill(int \*\*& matrix, const int matrixSize) {

int currentColumn = 0;

for (int i = 0; i < matrixSize; ++i) {

if (currentColumn % 2 == 0) {

for (int j = 0; j < matrixSize; ++j) {

matrix[j][currentColumn] = std::rand() % static\_cast<int>(std::pow(matrixSize, 2));

std::cout << "Added element with indexes [" << j << ";" << currentColumn << "]" << std::endl;

printMatrix(matrix, matrixSize);

}

}

else {

for (int j = matrixSize - 1; j >= 0; --j) {

matrix[j][currentColumn] = std::rand() % static\_cast<int>(std::pow(matrixSize, 2));

std::cout << "Added element with indexes [" << j << ";" << currentColumn << "]" << std::endl;

printMatrix(matrix, matrixSize);

}

}

++currentColumn;

}

}

void initSquareMatrix(int \*\*& matrix, const int matrixSize) {

matrix = new int \*[matrixSize];

for (int i = 0; i < matrixSize; ++i) {

matrix[i] = new int[matrixSize];

}

for (int i = 0; i < matrixSize; ++i) {

for (int j = 0; j < matrixSize; ++j) {

matrix[i][j] = 0;

}

}

}

void shuffleMatrix(int \*\*& matrix, const int matrixSize, const int shuffleType) {

int \*\* topLeft, \*\* topRight, \*\* bottomLeft, \*\* bottomRight;

initSquareMatrix(topLeft, matrixSize / 2);

initSquareMatrix(topRight, matrixSize / 2);

initSquareMatrix(bottomLeft, matrixSize / 2);

initSquareMatrix(bottomRight, matrixSize / 2);

for (int i = 0; i < matrixSize / 2; ++i)

for (int j = 0; j < matrixSize / 2; ++j)

topLeft[i][j] = matrix[i][j];

for (int i = 0; i < matrixSize / 2; ++i)

for (int j = matrixSize / 2; j < matrixSize; ++j)

topRight[i][j - matrixSize / 2] = matrix[i][j];

for (int i = matrixSize / 2; i < matrixSize; ++i)

for (int j = 0; j < matrixSize / 2; ++j)

bottomLeft[i - matrixSize / 2][j] = matrix[i][j];

for (int i = matrixSize / 2; i < matrixSize; ++i)

for (int j = matrixSize / 2; j < matrixSize; ++j)

bottomRight[i - matrixSize / 2][j - matrixSize / 2] = matrix[i][j];

if (shuffleType == 1) {

for (int i = 0; i < matrixSize / 2; ++i)

for (int j = 0; j < matrixSize / 2; ++j)

matrix[i][j] = bottomLeft[i][j];

for (int i = 0; i < matrixSize / 2; ++i)

for (int j = matrixSize / 2; j < matrixSize; ++j)

matrix[i][j] = topLeft[i][j - matrixSize / 2];

for (int i = matrixSize / 2; i < matrixSize; ++i)

for (int j = 0; j < matrixSize / 2; ++j)

matrix[i][j] = bottomRight[i - matrixSize / 2][j];

for (int i = matrixSize / 2; i < matrixSize; ++i)

for (int j = matrixSize / 2; j < matrixSize; ++j)

matrix[i][j] = topRight[i - matrixSize / 2][j - matrixSize / 2];

}

else if (shuffleType == 2) {

for (int i = 0; i < matrixSize / 2; ++i)

for (int j = 0; j < matrixSize / 2; ++j)

matrix[i][j] = bottomRight[i][j];

for (int i = 0; i < matrixSize / 2; ++i)

for (int j = matrixSize / 2; j < matrixSize; ++j)

matrix[i][j] = bottomLeft[i][j - matrixSize / 2];

for (int i = matrixSize / 2; i < matrixSize; ++i)

for (int j = 0; j < matrixSize / 2; ++j)

matrix[i][j] = topRight[i - matrixSize / 2][j];

for (int i = matrixSize / 2; i < matrixSize; ++i)

for (int j = matrixSize / 2; j < matrixSize; ++j)

matrix[i][j] = topLeft[i - matrixSize / 2][j - matrixSize / 2];

}

else if (shuffleType == 3) {

for (int i = 0; i < matrixSize / 2; ++i)

for (int j = 0; j < matrixSize / 2; ++j)

matrix[i][j] = bottomLeft[i][j];

for (int i = 0; i < matrixSize / 2; ++i)

for (int j = matrixSize / 2; j < matrixSize; ++j)

matrix[i][j] = bottomRight[i][j - matrixSize / 2];

for (int i = matrixSize / 2; i < matrixSize; ++i)

for (int j = 0; j < matrixSize / 2; ++j)

matrix[i][j] = topLeft[i - matrixSize / 2][j];

for (int i = matrixSize / 2; i < matrixSize; ++i)

for (int j = matrixSize / 2; j < matrixSize; ++j)

matrix[i][j] = topRight[i - matrixSize / 2][j - matrixSize / 2];

}

else if (shuffleType == 4) {

for (int i = 0; i < matrixSize / 2; ++i)

for (int j = 0; j < matrixSize / 2; ++j)

matrix[i][j] = topRight[i][j];

for (int i = 0; i < matrixSize / 2; ++i)

for (int j = matrixSize / 2; j < matrixSize; ++j)

matrix[i][j] = topLeft[i][j - matrixSize / 2];

for (int i = matrixSize / 2; i < matrixSize; ++i)

for (int j = 0; j < matrixSize / 2; ++j)

matrix[i][j] = bottomRight[i - matrixSize / 2][j];

for (int i = matrixSize / 2; i < matrixSize; ++i)

for (int j = matrixSize / 2; j < matrixSize; ++j)

matrix[i][j] = bottomLeft[i - matrixSize / 2][j - matrixSize / 2];

}

else {

std::cout << "Error: Unknown shuffle type. Matrix was not changed" << std::endl;

}

}

void sortMatrix(int \*\*& matrix, const int matrixSize) {

int \* masFromMatrix = new int[matrixSize \* matrixSize];

for (int i = 0; i < matrixSize; ++i)

for (int j = 0; j < matrixSize; ++j)

masFromMatrix[matrixSize \* i + j] = matrix[i][j];

for (int i = 1; i < matrixSize \* matrixSize; i++) {

int z = masFromMatrix[i];

int x = i - 1;

while (x >= 0 && masFromMatrix[x] > z) {

masFromMatrix[x + 1] = masFromMatrix[x];

masFromMatrix[x] = z;

x--;

}

}

for (int i = 0; i < matrixSize; ++i)

for (int j = 0; j < matrixSize; ++j)

matrix[i][j] = masFromMatrix[matrixSize \* i + j];

}

void processMatrix(int \*\*& matrix, const int matrixSize,

const int processType, const int processNum) {

for (int i = 0; i < matrixSize; ++i)

for (int j = 0; j < matrixSize; ++j) {

if (processType == 1)

matrix[i][j] += processNum;

if (processType == 2)

matrix[i][j] -= processNum;

if (processType == 3)

matrix[i][j] /= processNum;

if (processType == 4)

matrix[i][j] \*= processNum;

}

}

void Lab3() {

int matrixSize = 0;

std::cout << "Input matrix size: ";

std::cin >> matrixSize;

matrixSize >>= 1;

matrixSize <<= 1;

int \*\* matrix;

initSquareMatrix(matrix, matrixSize);

std::cout << "Source matrix:" << std::endl;

printMatrix(matrix, matrixSize);

std::cout << "For spiral fill input 1" << std::endl

<< "For shake fill input 2" << std::endl

<< "Choose fill type:";

int fillType = 0;

std::cin >> fillType;

if (fillType == 1) {

spiralFill(matrix, matrixSize);

}

else if (fillType == 2) {

snakeFill(matrix, matrixSize);

}

else {

std::cout << "Error: Unknown fill type" << std::endl;

return;

}

std::cout << "For circle shuffle input 1 " << std::endl

<< "For diagonal shuffle input 2 " << std::endl

<< "For vertical shuffle input 3 " << std::endl

<< "For horizontal shuffle input 4 " << std::endl

<< "Choose shuffle type:";

int shuffleType = 0;

std::cin >> shuffleType;

shuffleMatrix(matrix, matrixSize, shuffleType);

std::cout << "Matrix after shuffle:" << std::endl;

printMatrix(matrix, matrixSize);

sortMatrix(matrix, matrixSize);

std::cout << "Matrix after sort:" << std::endl;

printMatrix(matrix, matrixSize);

std::cout << "Input process type:";

int processType = 0;

std::cin >> processType;

std::cout << "Input process number:";

int processNum = 0;

std::cin >> processNum;

processMatrix(matrix, matrixSize, processType, processNum);

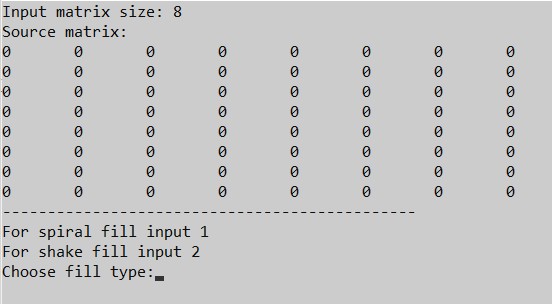
std::cout << "Matrix after process:" << std::endl;

printMatrix(matrix, matrixSize);

}

Приложение Б

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ



Предлагает выбрать спиральное или змейкой заполнение

