**Задания к контрольной работе (ВАРИАНТ №5)**

В рамках данной контрольной работы предлагается осуществить формирование требований, проектирование, реализацию и тестирование программы. Типовые темы заданий приводятся ниже для десяти вариантов. Возможен выбор темы, отличающейся от типовой. Выбор темы из приводимого ниже списка альтернативных тем производится без согласования с преподавателем. При согласовании с преподавателем допускается выбор собственной темы, отличающейся от типовой или альтернативной из списка.

**Общий для всех вариантов и тем план выполнения работ разбит на следующие этапы:**

1. Формирование требований. Представьте себя заказчиком и потенциальным пользователем программы, которая решала бы задачу Вашего варианта. Сформируйте требования пользователя к ней. Можно использовать как неформальное описание на естественном языке (русском, английском), так и какой-либо формальный язык для спецификаций. В любом случае важен не объем текста (это может быть, например, всего 0.5-2 страницы), а содержательное описание функций с обоснованием их выбора.

2. Проектирование

2.1. Выберите и обоснуйте аппаратную платформу, ОС, язык программирования и компилятор для него (или некоторую интегрированную среду разработчика). При необходимости, также можно выбрать какую-либо заемную библиотеку функций. Выбор производится, исходя из сформированных в задании 1 требований, имеющихся у Вас аппаратных ресурсов, знания языков программирования и имеющегося набора программного обеспечения. В данном разделе желательно избегать подробное изложение достоинств выбираемых сред, которые не относятся непосредственно к обоснованию их выбора (достаточно указания ссылок на печатные или электронные материалы).

2.2. Нарисуйте в виде блок-схемы алгоритм работы программы.

2.3. Если использован объектно-ориентированный подход, перечислите все классы, их атрибуты и методы, отношение наследования (в виде текста на естественном языке или текста на выбранном языке программирования, поддерживающем объектно-ориентированное программирование, или в виде диаграммы классов).

2.4. Нарисуйте диаграмму с набором модулей, на которые будет разбита программа. Стрелками для каждого модуля укажите, какие модули используются данным модулем.

3. Реализация

3.1. Программирование

По результатам проектирования реализуйте программу на выбранной платформе и на выбранном языке.

В процессе реализации для устранения ошибок пользуйтесь отладчиком. Для этого соберите программу в отладочном варианте. Если Вы выбрали GNU C/C++, то командная строка для сборки может быть такой: *gcc –g –o program.bin program.c.* Запустите собранный бинарный файл в отладчике. Пример для GCC и GDB приведен в гл. 6.

Результат выполнения задания 3 – все тексты программы и использовавшаяся для ее компиляции команда. Тексты программы оформляются в соответствии с рекомендациями по стилю, которые приведены в гл. 4 конспекта лекций данного курса.

**3.2. Документирование – разработка модели.** Возьмите за основу модель документации из приложения 2. Предложите измененную версию модели, которая на Ваш взгляд была бы удобна для описания программ, подобных построенной в задании 3. Для каждого добавленного, измененного и исключенного пункта объясните причину, почему это было необходимо сделать.

**3.3. Документирование.** На основе полученной в предыдущем пункте модели документации, составьте документацию для программы, реализованной в этом задании (п. 3.1).

4. Оценка производительности. Проведите измерение времени выполнения получившейся в задании 4 программы зависимости от объема данных. Если Вы выбрали в качестве языка программирования C/C++, то для замеров можно воспользоваться функциями из файла *benchmark.1.c* (или *benchmark.2.c*). При этом функция main будет иметь такой вид:

*main(){*

 *benchmark\_start();*

 *…*

 *рrintf(“duration = %d\n”, benchmark\_stop());*

*}*

Сделайте это для нескольких вариантов, изменив флаги для компилятора, которые отвечают за уровни оптимизации. Можно попробовать и флаги, включающие и выключающие отдельные виды оптимизации. Если Вы выбрали GNU C/C++, то минимальный рекомендуемый набор вариантов для тестирования таков:

*gcc –o program.bin program.c*

*gcc –O2 –o program.bin program.c*

*gcc –O3 –o program.bin program.c*

*gcc –O4 –o program.bin program.c*

Постройте графики, где по ось X задана в соответствии с Вашим вариантом, а ось Y – время выполнения программы в миллисекундах.

**Выполнение этапов 1, 2.1, 2.2., 3.1. и 3.3. - обязательно.** Выполнение остальных этапов желательно. Важность их выполнения варьируется в зависимости от выбранной темы.

**Типовая тема заданий**

**Вариант 5.** Симулятор клеточного автомата с игрой Конуэйя ”Жизнь”

Автомат для игры ”Жизнь” можно представить двумерным массивом 1-битовых значений. Нуль интерпретируется как ”мертвая” клетка, один – как ”живая”. Новое значение каждой клетки вычисляется как функция от клеток ее окрестности 3 на 3. Вычисляется сумма восьми соседей клетки. Для мертвой клетки значение остается нулем для всех значений получившейся суммы, исключая – 3, что называется рождением клетки. Для живой клетки происходит переход в 0, если сумма меньше 2 (гибель от одиночества) или больше 3 (гибель от перенаселенности). В остальных случаях клетка остается в состоянии 1.

На каждом шаге моделирования новые значения клеток сначала вычисляются и сохраняются в некотором буфере, а потом одновременно записываются в массив. Такое синхронное исполнение можно реализовать, например, имея две копии массива. На четных шагах идет запись в одну копию, а на нечетных – в другую. Или можно, например, использовать списки для организации буфера.

В минимальном виде симулятор – это утилита, которой в качестве параметров передается имя входного файла c исходной конфигурацией массива и число шагов , имя выходного файла с результирующей строкой, имя файла с правилами. Утилита 1) считывает входные параметры и проверяет их корректность, 2) открывает исходный файл и файл с правилами, обрабатывая возможные ошибочные ситуации, 3) производит поиск и применение подстановок, 4) записывает результат в результирующий файл и завершает работу.

**Альтернативные темы заданий**

(не требуют согласования с преподавателем, любая из них может выбираться самостоятельно вместо стандартной темы для Вашего варианта)

**Тема 1.** Интерпретатор языка LISP. Минимальный набор функций - car, cdr, cons, defun, cond, print (печать на консоль), read (чтение с консоли), загрузка фрагмента программы из файла.

**Тема 2.** Симулятор нормальных алгоритмов Маркова – реализует обработку текстовой строки (обрабатываемая строка) по набору правил. Число правил – произвольное, правила упорядочены. Каждое правило определяет текстовую подстановку и состоит из двух строк. Первая (левая) строка определяет заменяемую подстроку. Вторая (правая) строка задает, на какую она заменяется. Исполнение происходит до неприменимости, т.е. того момента, когда в обрабатываемой строке нет ни одной подстроки, которая совпадает с одной из строк из левых частей правил. Применение правила заключается в поиске вхождения его левой части в обрабатываемую строку и замена соответствующей подстроки на правую часть правила. Если таких вхождений несколько, то заменяется только первое вхождение. Если вхождений нет, то правило – неприменимо. Применение правил идет следующим образом. Берется первое правило и пытается примениться. Если оно неприменимо, берется следующее правило. И так происходит либо до выхода по неприменимости, либо до первого применимого правила. После того, как применимое правило сработало, все начинается сначала – с первого правила.

Пример:

Обрабатываемая строка:

AAAAEEabbbb

Правила:

AE -> ggg

A -> b

bg -> X

Промежуточные и конечный результаты будут такими:

1. AAAgggEabbbb

2. bAAgggEabbbb

3. bbAgggEabbbb

4. bbbgggEabbbb

5. bbXggEabbbb

В минимальном виде симулятор – это утилита, которой в качестве параметров передается имя входного файла c исходной строкой, имя выходного файла с результирующей строкой, имя файла с правилами. Утилита 1) считывает входные параметры и проверяет их корректность, 2) открывает исходный файл и файл с правилами, обрабатывая возможные ошибочные ситуации, 3) производит поиск и применение подстановок, 4) записывает результат в результирующий файл и завершает работу.

Набор правил, синтаксис их записи и вид исходной строки выбирается по собственному усмотрению.

Для задания 4 ось X – размер входного файла с обрабатываемой строкой. Все измерения нужно проводить с одним набором правил.

**Тема 3.** Симулятор нормальных алгоритмов Маркова на бинарных строках. Вариант аналогичен теме 2. Но: 1) вместо текстовых строк в качестве преобразуемой строки берутся произвольные бинарные (т.е. такие, где байты могут иметь любые значения от 0 до 255), 2) в правилах вместо символов используются шестнадцатеричные коды.

Пример:

Обрабатываемая строка в шестнадцатеричном представлении:

00 ff ff fe 80 80 79 32

Правила:

ff ff -> fe fe fe

fe fe fe fe -> 00

00 00 -> 01

01 -> 02

Промежуточные и конечный результаты будут такими:

1. 00 fe fe fe fe 80 80 79 32

2. 00 00 80 80 79 32

3. 01 80 80 79 32

 4. 02 80 80 79 32

**Тема 4.** Интерпретатор языка REFAL.

**Тема 5.** Локальная фильтрация изображений. Минимальный набор функций - загрузка изображения в формате bmp truecolor, сохранение изображения в формате bmp truecolor, задание значений коэффициентов локального фильтра, проведение фильтрации загруженного изображения. Желательный дополнительный набор функций: показ изображения, откат к предыдущему изображению, создание библиотеки фильтров, занесение фильтра в библиотеку фильтров, выбор фильтра из библиотеки для проведения фильтрации.

Файлы для выполнения Контрольной работы (.С):

benchmark.1.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 FILE: benchmark.c

 DESCRIPTION:

 Benchmark routines implementation in A.S.R.

 VERSION:

 Jun 14, 1999

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <sys/time.h>

#include <unistd.h>

struct timeval tv1, tv2, dtv;

struct timezone tz;

/\*+ benchmark\_start

PROTOTYPE:

 void benchmark\_start(void);

DESCRIPTION:

 Runs a time counter for delay measurement.

PARAMETERS:

 none

RETURNS:

 nothing

-\*/

void benchmark\_start(void){

 gettimeofday(&tv1, &tz);

} /\* benchmark\_start \*/

/\*+ benchmark\_stop

PROTOTYPE:

 long benchmark\_stop(void);

DESCRIPTION:

 Stops the time counter for delay measurement.

PARAMETERS:

 none

RETURNS:

 number of milliseconds between start and stop counter calls.

-\*/

long benchmark\_stop(void){

 gettimeofday(&tv2, &tz);

 dtv.tv\_sec = tv2.tv\_sec - tv1.tv\_sec;

 dtv.tv\_usec = tv2.tv\_usec - tv1.tv\_usec;

 if(dtv.tv\_usec < 0){

 dtv.tv\_sec--;

 dtv.tv\_usec += 1000000;

 }

 return dtv.tv\_sec \* 1000 + dtv.tv\_usec / 1000;

} /\* benchmark\_stop \*/

/\* end of file - benchmark.c \*/

benchmark.2.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 FILE: benchmark.2.c

 DESCRIPTION:

 Benchmark routines implementation in A.S.R.

 VERSION:

 .1 - 19990614 - system clock based

 .2 - 20021015 - process times based

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <sys/times.h>

#include <unistd.h>

struct tms tmsBegin;

struct tms tmsEnd;

/\*+ benchmark\_start

PROTOTYPE:

 void benchmark\_start(void);

DESCRIPTION:

 Runs a time counter for delay measurement.

PARAMETERS:

 none

RETURNS:

 nothing

-\*/

void benchmark\_start(void){

 times(&tmsBegin);

} /\* benchmark\_start \*/

/\*+ benchmark\_stop

PROTOTYPE:

 long benchmark\_stop(void);

DESCRIPTION:

 Stops the time counter for delay measurement.

PARAMETERS:

 none

RETURNS:

 number of milliseconds between start and stop counter calls.

-\*/

long benchmark\_stop(void){

 times(&tmsEnd);

 return ((tmsEnd.tms\_utime - tmsBegin.tms\_utime) +

 (tmsEnd.tms\_stime - tmsBegin.tms\_stime)) \* 10;

} /\* benchmark\_stop \*/

/\* end of file - benchmark.2.c \*/