Задания по программированию студентам группы 9201 ФИТ НГУ. Второй семестр

Александр Геннадьевич Фенстер

2010 г.

Для получения зачёта во втором семестре студентам необходимо и достаточно выполнить следующие действия:

1. На практических занятиях в терминальном классе сдать перечисленные ниже задания. Сдача задач по электронной почте и другими «удалёнными» способами допускается лишь в виде исключения (болезнь и т. п.).

2. Сдать устный зачёт в конце семестра. Некоторые студенты могут быть от него освобождены.

Студенты, сдающие задания позже указанного срока, могут получить дополнительную задачу за каждую просроченную неделю.

На сайте `http://9201.fenster.name` будет выкладываться таблица с информацией о сданных задачах.

Зачёт во втором семестре будет не дифференцированным, т. е. вариантов оценки два: «зачёт» или «незачёт», однако условная оценка «за работу в семестре» будет учитываться на экзамене.

В этом семестре вам необходимо научиться писать программы на C, состоящие из нескольких файлов, компилирующихся по отдельности. К каждой программе необходимо будет приложить `Makefile` для сборки её в UNIX-окружении и файл `.vcproj` для компиляции в Microsoft Visual Studio 2005. Программа должна компилироваться без ошибок и предупреждений (warnings) как при помощи установленной на сервере версии `gcc`, так и компилятором `cl.exe` из Visual Studio (в связи с тем, что стандартные функции `scanf`, `strcpy` и многие другие в MSVS 2005 внезапно стали deprecated, вам придётся приложить некоторые усилия для достижения этой цели). Подробнее об этом мы поговорим на занятиях.

Задание 1. Вычисления на стеке

Необходимо написать программу-калькулятор, несколько напоминающую стандартную программу `bc`. На вход программе подаются строки одного из двух видов:

1. присваивание: `идентификатор=выражение`
2. выражение для вычисления: `выражение`

Получив на вход строку первого типа, программа должна вычислить значение выражения и сохранить его значение в переменной с указанным именем. Получив на вход строку второго типа, программа должна вычислить и напечатать значение введённого выражения. Пустые строки игнорируются. Программа завершает работу при нажатии Ctrl+D (символ конца файла).

В качестве идентификатора (имени переменной) может использоваться строка произвольной длины, состоящая из латинских букв, цифр и знака подчёркивания и начинающаяся не с цифры. Выражение — это строка, состоящая из имён переменных, чисел, знаков операций и скобок, корректная с точки зрения математики. Калькулятор должен работать с числами типа `int`, деление считается целочисленным. Ниже приведено более строгое описание входных данных в виде формы Бэкуса-Наура:

```
        цифра = '0' | '1' | ... | '9'
        буква = 'a' | 'b' | ... | 'z'
идентификатор = буква {буква | цифра | '_'}
        число = ['+' | '-'] цифра {цифра}
      операнд = идентификатор | число
    операция = '+' | '-' | '*' | '/'
   выражение = операнд | '(' выражение ')' |
                         выражение операция выражение
присваивание = идентификатор '=' выражение
   программа = {присваивание '\n' | выражение '\n' | '\n'}
```

Здесь квадратные скобки означают «встречается 0 или 1 раз», а фигурные — «встречается 0 или более раз».

Для разбора выражения используйте разобранный на семинарах алгоритм. Не нужно писать полноценный синтаксический разбор выражения и строить дерево разбора: основной целью задания является не разбор выражения, а реализация функций работы со стеком, хранящим произвольные данные, и их использование.

Программа должна содержать как минимум следующие файлы:

1. Реализация стека для хранения произвольных данных (`void *`) при помощи односвязного списка. Поместите заголовки основных функций в файл `stack.h`, а их реализацию — в `stack.c`. Заголовки функций должны выглядеть примерно так:

   ```
   void push(stack *st, void *data); // вставка
   void *top(stack *st);       // верхушка стека
   void pop(stack *st);        // удаление
   int empty(stack *st);       // проверка на пустоту
   ```

   Могут также присутствовать функции для инициализации и уничтожения стека. Напишите небольшую тестовую программу, проверяющую написанные функции.

2. Реализация структуры данных для хранения значений переменных (например, хэш-таблицы). Поместите заголовки функций в файл `vars.h`, а реализацию — в `vars.c`. Заголовки этого набора функций должны выглядеть примерно так:

   ```
   void set_value(char *name, int value); // присваивание
   int get_value(char *name); // значение переменной
   void cleanup(void); // очистка памяти в конце работы
   ```

3. Основная программа (например, `main.c`).

4. Вспомогательные файлы для сборки: `Makefile`, файлы проекта для Microsoft Visual Studio.

В программе будут использоваться два стека: стек для хранения операций и скобок в первой части алгоритма и стек для хранения чисел во второй части. Вы должны сделать одну реализацию стека, хранящего произвольные данные (`void *`), и использовать её для обоих стеков. Вероятно, вам придётся потратить некоторое время на то, чтобы понять, как правильно добавлять данные в такой стек. Помните также о необходимости освобождения всей выделенной памяти.

В связи с большим объёмом задания его необходимо сдавать по частям в порядке усложнения: сначала стек, затем функции для работы с переменными, а затем основной алгоритм. **Программу такого объёма практически нереально сдать за один раз!**

Срок сдачи задания: первая контрольная неделя.

Задание 2. Длинная арифметика

Модифицируйте калькулятор из предыдущего задания так, чтобы он принимал целые числа произвольного размера. Достаточно удобно хранить число переведённым в систему счисления по основанию, например, 10000. В этом случае цифрами числа будут являться числа от 0 до 9999 включительно, а для вывода числа не нужно производить практически никаких действий: например, число 912340567 будет представлено в системе по основанию 10000 в виде трёхзначного числа $9‘1234‘567_{10000}$ и для печати нужно просто вывести по порядку все его цифры, «добивая» все, кроме первой, до четырёх знаков (`"%04d"`).

При таком способе хранения число можно будет хранить в программе в виде следующей структуры:

```
struct long_int
{
     int *digits;  /* цифры, удобнее в обратном порядке */
     int length;   /* количество цифр */
     int sign;     /* знак числа: 1 или -1 */
};
```

Вам нужно будет создать отдельную библиотеку для выполнения операций с такими числами. Описание структуры `long_int` поместите в файл `long.h`. Также там будут находиться заголовки функций, например, такие:

```
struct long_int add(struct long_int a, struct long_int b);
struct long_int sub(struct long_int a, struct long_int b);
struct long_int mul(struct long_int a, struct long_int b);
struct long_int div(struct long_int a, struct long_int b);
void print(struct long_int a);
void free_int(struct long_int a);
```

и любые другие на ваше усмотрение. Вспомогательные функции, которые нужны только вам и не предназначены для пользователей библиотекой, не нужно вносить в заголовочный файл. Более того, их нужно сделать статическими.

Обратите особое внимание на работу с памятью: при выполнении нескольких арифметических операций (например, при реализации умножения) вам придётся создавать временные переменные. Не забывайте освобождать всю неиспользуемую память. В других языках для этого есть специальные механизмы (деструкторы и `auto_ptr` в C++, сборка мусора в Java), в C же приходится помнить об этом постоянно.

Срок сдачи задания: вторая контрольная неделя.

Задание 3. Архиватор Хаффмана

Реализуйте архиватор и деархиватор, работающие по методу Хаффмана. Программа должна иметь примерно такой интерфейс:

```
$ ./huffman archive.huf file.txt        # архивация
$ ./huffman -d archive.huf unpacked.txt # деархивация
```

или любой другой на ваше усмотрение (но программой должно быть удобно пользоваться из командной строки).

Для выполнения архивации и деархивации вам понадобится уметь делать как минимум следующие действия:

1. Обработка параметров командной строки.
2. Чтение и запись данных в файл.
3. Построение дерева Хаффмана и кодов Хаффмана.
4. Сохранение дерева или списка кодов в файл и чтение из файла.
5. Работа с битами целого числа.

Постарайтесь разбить проект на файлы наиболее логичным образом. Старайтесь не использовать глобальные переменные и сведите к минимуму зависимости компонентов программы друг от друга.

Для проверки корректности архивации и деархивации попробуйте запаковать и распаковать двоичный файл (например, `/bin/bash`) и при помощи программы `diff` проверить работу своей программы.

Срок сдачи задания: зачётная неделя.