Глава 5

Приложение

5.1 Прожиточный минимум.

Процесс разработки программы на ЭВМ состоит из следующих стандартных шагов. Первое, с помощью текстового редактора (не текстового процессора) набираем программу, например, в файле `task-2_11.c`. Второе, компилируем `task-2_11.c` и получаем файл `task-2_11.o`. Если компилятор выдает сообщения о синтаксических ошибках или предупреждения, то исправляем код, обычно начиная с первого замечания, и компилируем заново. Третье, собираем (линкуем) `task-2_11.o` с библиотечными функциями и получаем `task-2_11.e`. Четвертое, запускаем итоговый файл `./task-2_11.e`.

А далее приступаем к тестированию и поиску алгоритмических и программных ошибок.

5.1.1 Установка полезных программ.

Если на доступной Вам ЭВМ установлена ОС типа Linux со стандартным набором пакетов, то в Вашем распоряжении имеется консольный редактор vim, продвинутый редактор типа kwriter, позволяющие выполнить первый шаг, компилятор gcc/g++ и утилита make для выполнения второго и третьего шагов, утилита gdb для отладки кода и поиска ошибок, а также программа построения графиков gnuplot для визуализации полученных результатов.

При работе в ОС Windows разумно установить пакет Cygwin — бесплатно распространяемый эмулятор Linux, требующий минимальных ресурсов ЭВМ и усилий при освоении. Инсталляция Cygwin состоит из следующих шагов (подробнее см. документацию на сайте): заходим на сайт http://www.cygwin.com, спускаемся по ссылке Install Cygwin и в зависимости от архитектуры Вашего компьютера скачиваем файл-загрузчик (install-32 или install-64). Запускаем его либо с правами администратора (что обычно доставляет меньше хлопот), либо переименовываем и запускаем из командной строки cmd с ключом - - no-admin под обычным пользователем.

Выбираем директорию для установки (обычно c:/cygwin), режим установки из “интернета”, ссылку для скачивания и указываем набор интересующих нас пакетов, отметив их галочками. На первом этапе достаточно установить:

Admin:Install;
Base: Install;
Devel: gcc-core, gcc-fortran, gcc-g++, gdb, make, mcpp;
Doc: cygwin-doc, man;
Editors: mc, vim, vim-clang-format, vim-common, vim-doc;
Graphics: gnuplot.
Net: openssh, ssh-pageant

В результате в c:/cygwin создается набор стандартных поддиректорий ОС Linux, а Вашей домашней (рабочей) с точки зрения Cygwin директорией является C:/cygwin/home/“ВашеWindowsИмя”. Запустив из-под Windows пакетный файл c:/cygwin/Cygwin.bat, получаем окно-эмулятор ОС Linux и возможность выполнять все установленные Linux-пакеты. Далее стоит создать поддиректорию типа C:/cygwin/home/“ВашеWindowsИмя”/Cprog для сохранения и компиляции C-программ. Отметим, что редактировать файлы в директории Cprog удобнее из ОС Windows стандартным редактором типа Notepad++ (бесплатное по, http://notepad-plus-plus.org, по ссылке download выбираем Notepad++ zip package, распаковываем в стандартную папку, запускаем Notepad++.exe), а компилировать набранные C-файлы и запускать полученный в результате исполнимый код необходимо из окна Cygwin, перейдя предварительно в директорию Cprog. При этом стоит убедиться, что Вы осуществляете компиляцию и редактирование одних и тех же файлов (например, выполнив для этого в окне Cygwin в рабочей директории C:/cygwin/home/user/Cprog команду `cat task-2_11.c`).

Использование сред разработки типа Visual Studio Express Edition C/C++, Code::Blocks или Dev-C++, позволяющих объединить все шаги, на наш взгляд мешает изучению языка C, получению базовых навыков поиска синтаксических и логических ошибок в своей программе и поэтому неразумно на первом этапе обучения. Назначение подобных программных продуктов — облегчить жизнь профессионального программиста, а поэтому ими стоит пользоваться тем, кто полностью освоил язык C и хочет сосредоточить все свое внимание на разработке больших и сложных проектов.

Отметим, что в интернете имеется огромное число C/C++ онлайн компиляторов, от элементарных типа http://ideone.com, до профессиональных типа http://rextester.com, а также облачных приложений типа https://c9.io. Однако, их использование создает неискушенному пользователю еще больше проблем, чем среда разработки, а поэтому стоит взвесить все за и против при их выборе.

5.1.2 Консольный режим: shell, vim, gcc,make

```
whoami          выдать имя пользователя      (узнать «кто я?»)
pwd             выдать полное имя текущей директории  (узнать «где я?»)
ls              выдать содержимое текущей директории  (узнать «что тут?»)
ls -a           выдать полное содержимое текущей директории
ls -l           выдать подробное содержимое текущей директории
clear           очистить экран


mkdir Cprog     создание директории с именем Cprog
rmdir Cprog     удаление пустой директории с именем CProg
cd ./A          переход из текущей директории в поддиректорию A
cd ..           переход в директорию предыдущего уровня
cd              переход из текущей директории в домашнюю директорию

cp a.c  b.c     копирование файла a.c  в  файл b.c
mv a.c  b.c     переименование файла a.c  в  файл b.c
rm a.c          удаление файла a.c
cat a.c         выдать на экран содержимое файла a.c
less a.c        постраничная выдача  на экран содержимого файла a.c
             ('пробел' - следующая страница, 'Enter' - следующая строка)
od   a.exe     выдать на экран коды символов файла a.exe
file a.exe      определить и выдать на экран тип файла a.exe

gcc a.c -c      компиляция файла a.c (создается файл a.o)
gcc a.o -lm     линковка файла a.o с подключением математической
    -o a.exe    библиотеки и созданием a.exe
./a.exe         запуск  файла a.exe из текущей директории
^C              аварийное завершение запущенного приложения
^Z              приостановить выполнение текущей команды
fg              продолжить выполнение приостановленной команды

man "команда"   постраничная выдача справки о "команде"
ps              выдать список текущих процессов
top             выдавать список текущих процессов в режиме
                реального времени (выход по клавише 'q')
kill -9 "PID"   аварийное завершение процесса с указанным PID
kill -9 -1      аварийное завершение всех доступных процессов
```

Базовые команды текстового редактора vi/vim.

Далее считается, что включена английская раскладка клавиатуры. При входе в файл редактор по умолчанию устанавливается в командный режим (нажатие клавиши воспринимается как команда). Переход из командного режима в режим командной строки осуществляется нажатием $Shift+:$ (здесь и далее знак + означает одновременное нажатие указанных клавиш). В результате появляется приглашение ':' в нижней строке окна редактирования. Для выхода из режима командной строки нажмите клавишу Enter.

```
vim a.c          вызов редактора из консольного режима для
                      редактирования a.c,
                      файл открывается в командном режиме
Shift+:            переход из командного режима в
                                              режим командной строки
:'Enter'          переход из режима командной строки в командный режим.
```

Режим командной строки. Для выполнения команды необходимо в строке приглашения ':' набрать нужную комбинацию и нажать клавишу Enter.

```
:q!                   выход из файла без сохранения изменений
:w                    сохранение на диск внесенных изменений
:wq                   сохранение и выход
:'стрелка вверх/вниз'  листание по набранному ранее списку команд
:set number           включить нумерацию строк
:set nonumber         выключить нумерацию строк
:n                    переход на строку с номером n
:/words               поиск последовательности words
:split b.c            открытие в редакторе второго окна с файлом b.c
:help                 открытие в редакторе второго окна с файлом help
```

Командный режим.

```
Ctrl'+ww      переход между открытыми окнами редактирования (если они есть)
'стрелка'     стандартные перемещения по тексту
PgUp, PgDn    сместиться  на страницу   вверх/вниз
Home, End     сместиться  в начало/конец строки
x             удалить текущий символ (над курсором)
X             удалить предыдущий символ (перед курсором)
J             склеить текущую и следующую строки
r<символ>     заменить символ над курсором на указанный
dd            удалить строку
o         вставить пустую строку за текущей строкой
                    и перейти в режим редактирования
O         вставить пустую строку на место текущей  строки
                    и перейти в режим редактирования
u             отменить последнее действие
.             повторить последнее действие
v          начать посимвольное выделение текста "стрелками"
```

```
V         начать построчное выделение текста "стрелками"
Ctrl+V    начать блочное выделение текста
y         запомнить выделенный текст в стандартный буфер
d         запомнить выделенный текст в стандартный буфер
                                                и стереть текст
p         вспомнить за курсором содержимое буфера
"ayy      очистить буфер с именем a и запомнить в него текущую строку
"Ayy      добавить текущую строку  в конец буфера a
"Ap       вставить содержимое буфера a с текущей позиции

i        переход в режим редактирования с текущей позиции курсора
I        переход в режим редактирования с начала строки
a        переход в режим редактирования со следующей позиции
A        переход в режим редактирования с конца текущей строки
```

Следует помнить, что в базовой конфигурации vi в режиме редактирования допускается только набор текста. Это значит, что запрещено использовать управляющие клавиши, в том числе ‘стрелки’. Для перемещения по тексту сначала потребуется перейти в командный режим.

```
Esc           переход в из режима редактирования в командный режим
```

За внешние настроки редактора типа подсветка синтаксиса и настройки режимов редактирования отвечает конфигурационный файл `/.vimrc` который полезно создать в домашней директории. За основу рекомендуется взять файл `./usr/share/vim/vimrc_example.vim`, сохранив его под указанным именем.

Компилляция.

```
gcc a.c -c      компиляция файла a.c (создается файл a.o)
gcc a.o -lm  -o a.exe   линковка файла a.o
                        с подключением математической
                        библиотеки и созданием a.exe
./a.exe          запуск программы a.exe
./a.exe  < inputdata
                 запуск a.exe  в режиме потокового ввода
                 начальных данных из файла inputdata
./a.exe  >  outputdata
                 запуск a.exe с выводом результатов
                 работы программы в файл outputdata
```

Для отслеживания различных проблемных моментов компилляцию рекомендуется проводить с ключами `-fsanitize=undefined -Wall -Wextra`.

Команда make. Разумная лень — двигатель прогресса: существенно легче один раз освоить написание make-файлов и далее этим пользоваться, чем каждый раз пошагово выполнять процедуру компиляции, сборки, запуска, рискуя что-то пропустить. Даже на однофайловой программе процесс разработки и тестирования ускоряется не менее, чем в два раза. Пусть требуется собрать исполнимый файл $m.e$ из файлов $./main.c$ и

$./fun.c$, содержащих соответственно функцию $main()$ и набор вспомогательных функций. При этом описание глобальных прототипов вынесено в $./com.h$ и его содержимое включается в $./main.c$ и $./fun.c$ с помощью команды `#include''./com.h'`. Создадим файл $./makefile$ следующей структуры:

```
m.e: <tab> main.o fun.o com.h
 <tab>              gcc main.o fun.o -lm -o m.e
main.o: <tab> main.c  com.h
  <tab>              gcc main.c -c
fun.o:    <tab>  fun.c com.h
  <tab>               gcc  fun.c  -c
clean:
  <tab>               rm -f m.e main.o fun.o *.rez
```

По сути, в $makefile$ записано в специальном формате пошаговое дерево зависимостей для цели-выршины $./m.e$ от исходного кода $./main.c$, $./fun.c$, $./com.h$ и правила перехода по его ветвям. Система, при выполнении команд $make$, анализирует дерево зависимостей и сравнивает даты изменения каждой вершины дерева и порождающих ее объектов. Если нарушается естественная упорядоченность по времени, то каждая такая ветвь пересобирается. В данном случае изменение $./com.h$ приведет к полной пересборке дерева, а изменение $./main.c$ потребует выполнения только двух первых команд. При выполнении $make\ clean$ будет вызвана команда, следующую за строкой, помеченой ссылкой $clean$.

Что общего между магами и программистами – и те другие шепчут под нос непонятные слова, делают загадочные пассы руками и ... в результате не могут внятно объяснить, почему же все это работает.

Отладчик gdb. Стандартная схема работы с ртладчиком выглядит следующим образом: исходные с-файлы компиллируем с ключом -g без оптимизации (в результате в main.o сохраняется таблица соответствия со строками main.c); линкуем и запускаем gdb, указав имя исполнимого файла в качестве параметра. На приглашение gdb> набираем команду run, нажимаем Enter и вводим (если требуется) данные с клавиатуры.

```
 gcc  main.c  -Wall  -g -c -O0   Компилляция с запретом
              на оптимизацию кода и ключом -g
 gcc main.o -o main.e  сборка
 gdb ./main.e          запуск GnuDeBuger
gdb>run
```

В случае "падения" программы на экран выдается информация о типе проблемы и соответсвующая строка с-кода. Далее, можно распечатать содержимое переменных и попытаться выявить ошибку. Если сбой происходит внутри стандартных библиотечных функций, то полезно распечатать стек вызовов функций с помощью команды backtrace, и перемещаясь по хранящимся в стеке фреймам выяснить (с помощью указанных далее механизмов) причины сбоя программы. Отметим, что большинство команд gdb допускают одно/двухбуквенные сокращения.

```
quit              выйти из отладчика
help              обзор команд
run               запуск загруженного исполнимого файла
Ctr+C           прервать выполнение
continue       продолжить выполнение программы

print i            печатать содержимого
print  mas[i]           переменной,
print &mas[i]          адреса переменной
ptype i              печать типа переменной
x/4x &i            побайтовый вывод содержимого памяти:
                      4 байта в heX формате (на процессорах x86
        байты хранятся в порядке от младшего к старшему)
set var i   3      задать новое значение

list                 печать части листинга программы
list 20, 35
list main.c:f1

backtrace        печать стека вызовов программ
backtrace 5     указанной части стека
backtrace -10
up; down     переход по фреймам стека
frame 2       перейти к указанному фрейму
info frame   информация о текущем фрейме
```

```
info args     показать аргументы в текущем фрейме
info locals   показать локальные переменные в текущем фрейме:
```

Последовательность действий при пошаговой отладке:

```
break main   задать точку останова в начале функции main
break main.c:23     в 23-й строке
break main.c:fun    в начале функции fun

step      выполнить шаг вперед
step 3      или несколько шагов вперед
stepi        одну инструкцию следующей строки кода

next
next 5    в отличие от step функция отрабатывается как один шаг
until 100  выполнить программу до указанной строчки
finish продолжить выполнение до возвращения из текущей функции

установка точек останова   (breackpoint):
break 33       на строку 33
break 33 if i == 12       если переменная  i равна 12
watch   i          на изменение содержимого переменной
rwatch  i          на считывание содержимого переменной
awatch i          на считывание/изменение переменной
hbreak    5   аппаратная точка останова на 5-ой строке
info break  листинг имеющихся точек останова
del  1      удаление точки останова по номеру
d            удаление всех точек останова
disable 1 временное отключение брейкпоинта
enable 1 включение брейкпоинта
```

Запущенная под Linux задача

```
#include<stdio.h>
#include<stdlib.h>
int main(){
int *a=NULL;
int i,n=1024;
a=(int *)malloc(n*sizeof(int));
i=1024;
while(1){i--; a[i]=1000000; if(i<0)break;}
free(a);
a=NULL;
return 0;
}
```

“падает” с сообщением типа core dumped, но проблема “прячется”, если удалить строку free(a). В данном случае аварийный останов это происходит

в результате порчи информации, хранящейся в памяти перед выделенным блоком, и необходимой для корректного высвобождения памяти при вызове функции `free()`. Следующая последовательность команд позволяет выявить проблему.

```
(gdb) list    // печать с-кода
(gdb) break 7 // установка breakpoint
                     // после выделения памяти
(gdb) run
(gdb) print  a
$1 = (int *) 0x601010  // адрес начала выделенного блока
(gdb) watch *( (int *) 0x601010 -1) // установка preakpoint
                                    // на изменение содержимого
(gdb) continue
Hardware watchpoint 2: *( (int *) 0x601010-1)
Old value = 0
New value = 1000000
0x0000000000400551 in main () at free.c:8
8       while(1){i--; a[i]=1000000; if(i<0)break;}
(gdb) print i
$2 = -1
(gdb) exit
```

Если системные настройки обеспечивают создание после падения программы соответствующего core-файла, то gdp можно запустить `gdb ./mai.exe core`; также вожможно подкючение к уже запущенному процессу `gdb ./mai.exe pid`.

5.1.3 Визуализация результатов: gnuplot.

Умение наглядно представить полученные результаты является необходимым не только на заключительном этапе решения задачи, но и в процессе отладки и верификации программы. Для этих целей прекрасно подходит бесплатно распространяемый пакет `gnuplot`, который можно установить под Windows (просто развернув архивный файл), и который входит в базовый набор всех стандартных версий Linux. По данной программе имеется огромное количество как полезных примеров в интернете, так и полноценная документация, см. [7], поэтому мы проведем лишь минимальный обзор. Идеология пакета gnuplot —- консольный режим. Команды можно набрать либо непосредственно в командной строке, либо вызвать (в случае оконной реализации) нажатием соответствующих кнопок. Часто используемую последоватедьность команд удобно набрать в некотором текстовом файле `Filename` (комментарии начинаются с символа #), и при необходимости загружать командой

`load "Path/FileName"`.

При желании можно сменить текущую дерикторию командой

`cd 'Path'` и выполнить `load "FileName"`.

При этом имя текущей директории можно определить командой `pwd`. Отметим, что строка `"Path/FileName"` не должна содержать символы национальных алфавитов. Также бывает удобно запомнить в буфере обмена набранные в файле команды и далее вставить содержимое буфера в командной строке `gnuplot`. Все выполненные в текущем сеансе команды последовательно сохраняются в буфере и могут быть вызваны нажатием клавиш “стрелка вверх”, “стрелка вниз”.

Для построения графиков по сути имеется две команды: `plot` для работы с функциями одной переменной и `splot` для функций, зависящих от двух переменных. Остальные команды пакета отвечают за стилевую настройку режима визуализации. Приведем скромный минимум команд пакета `gnuplot` без подробных пояснений, так как их назначение становится понятным после первого использования.

Для построения аналитически заданных функций можно использовать следующие конструкции:

```
plot sin(2*x)
set xrange[-1.:2.]
set yrange[-1.5:1.5]
replot
reset
set grid
plot sin(2*x)  linetype 5 linewidth 3,
         x*x linetype 2 linewidth 7
unset grid
plot sin(2*x) with points
                       title "Graph"
exit
```

Если имеется файл a.dat с двумя столбцами, в первом из которых содержатся значения дискретной координаты x, а во втором — соответствующие им значения y, и файл b.dat из нескольких столбцов, для которого координаты x содержатся в третьем, а y — в четвертом столбцах, то для заданных таким образом табличных функций можно использовать следующие команды:

```
cd 'C:/cygwin/home/user/Cprog'
pwd
plot 'a.dat'
plot 'b.dat' using 3:4
plot 'a.dat' with lines, 'b.dat' using 3:4 with dots
set style data linespoints
replot
```

Для построения таблично заданного векторного поля необходимо сохранить в некотором файле `v.dat` четыре столбца, первые два — для координаты точки на плоскости, вторые два — для длин горизонтальной и вертикальной компонент вектора. А далее набрать следующую команду:

```
plot 'v.dat' using 1:2:3:4   with vectors head filled lt 2
```

Для построения графиков аналитически заданных функций от двух переменных применяются команда `splot`:

```
splot x*y
splot sin(x)*sin(y), x*x+y*y
```

При этом можно использовать указанные ранее команды настройки.

Для таблично заданной функции достаточно подготовить, например, файл `c.dat` с тремя столбцами, в которых хранятся координаты x_i, y_j и соответствующие значения $f(x_i, y_j)$ и использовать команду

```
splot `c.dat'
```

Поверхности выглядят более наглядно, если сформировать файл `d.dat` с прямоугольной матрицей значений $f(x_i, y_j)$ и использовать следующую конструкцию

```
splot 'd.dat' matrix with lines
```

В этом случае координаты точек по умолчанию имеют целочисленные значения, соответсвующие позиции элемента в матрице. При этом элемент `(0,0)` располагается в левом нижнему углу экрана.

Для детализации получаемой в результате трехмерной картинки бывают полезны следующие модификации формата вывода:

```
set contour base; set contour surface; set format z  ''%.2f'';
set cntrparam levels 7;
set clabel "%.2f''; unset surface;
set view 0, 360; set xrange[0:64];
set xtics ("0" 0, "X" 32, "1" 64) offset 0,0;
unset xtics; set key  87,55; set hidden; set pm3d
```

Если требуется визуазиализировать динамику зависящей от времени дискретной функции, то в процессе расчета можно, например, создать файлы с именами $a.001$, $a.002$, $a.003$ одного из указанных форматов, соответствующие моментам времени $1, 2, 3$. Далее, например, в файл `dyn_a` записать (также автоматически в процессе расчета) строки

```
plot `a.001'; pause 2
plot `a.002'; pause 2
plot `a.003'; pause 2
```

и в gnuplot набирать команду

```
load `dyn_a'
```

Для сохранения построенных графиков в отдельных файлах, например, в форматах .png или .eps, удобно поступать следующим образом:

```
set size ratio 1
set terminal png; set out "c.png";
splot 'c.dat' matrix with lines;
set terminal postscript "Times-Roman" 30 lw 3;
set out "c.eps";
splot 'c.dat' matrix with lines;
set term windows; # set term X11 при работе в Linux-приложении
set out
```

Для более детального ознакомления с возможностями gnuplot можно следующим образом загрузить подготовленные разработчиками пакета командные демофайлы:

```
cd '~\gnuplot\demo'
load  'airfoil.dem'
load  'all.dem'
```

А также самостоятельно разобрать синтаксис каждой конструкции, вызвав встроенную команду `help`. Например,

```
help set contour
```