

Лабораторная работа №14

Журавлев Георгий Иванович

Цель работы

приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями.

Ход работы.

1. В домашнем каталоге создал подкаталог `~/work/os/lab_prog`.

```
g_zhuravlev@g ~$ mkdir -p ~/work/os/lab_prog
```

2. Создал в нём файлы: `CalcHeader.h`, `Calculator.cpp`, `operation.cpp`, `main.cpp`.

```
g_zhuravlev@g ~/work/os/lab_prog$ ls -l
total 20
-rw-rw-r-- 1 g_zhuravlev g_zhuravlev 169 мая 7 14:06 CalcHeader.h
-rw-rw-r-- 1 g_zhuravlev g_zhuravlev 1851 мая 7 14:26 Calculator.cpp
-rw-rw-r-- 1 g_zhuravlev g_zhuravlev 112 мая 7 12:27 main.cpp
-rw-rw-r-- 1 g_zhuravlev g_zhuravlev 381 мая 7 17:13 Makefile
-rw-rw-r-- 1 g_zhuravlev g_zhuravlev 184 мая 7 12:27 operation.cpp
```

2.1. Header file.

```
1  #include<string>
2  #ifndef CalcHeader_H
3  #define CalcHeader_H
4  void OperationTypes();
5  std::string operation();
6  double Calculate(int variable_number);
7  int NumofVar();
8  #endif
```

2.2. Calculator.cpp.

```
#include <iostream>
#include<cmath>
#include<string>
#include "CalcHeader.h"
using namespace std;

void OperationTypes() {
    cout << "There are only few possible operations with ONE or TWO variables: \n";
    cout << "Basic: + ; - ; * ; / ; pow.\nTrigonometric: sin; cos; tan.\n";
    cout << "Please, use only this operators, otherwise it won't work.\n";
}

double Calculate(int variable_number) {
    double a=0;
    double b=0;
    double result;
    string operationCalc;
    if (variable_number == 2) {
        cout << "first variable: "; cin >> a;
        operationCalc = operation();
        cout << "second variable: "; cin >> b;
        if (operationCalc == "+") {
            result = a + b;
        }
        else if (operationCalc == "-") {
            result = a - b;
        }
        else if (operationCalc == "*") {
            result = a * b;
        }
        else if (operationCalc == "/") {
            if (b == 0) {
                cout << "that can't be done(doesn't exist)\n";
                return 1;
            }
            else {
                result = a / b;
            }
        }
    }
}
```

```

39     }
40     else {
41         cout << "wrong option, operator is undefined ";
42         return 1;
43     }
44 }
45
46 else if(variable_number==1){
47     cout << "variable: "; cin >> a;
48     operationCalc = operation();
49     if (operationCalc == "sin") {
50         result = sin(a);
51     }
52     else if (operationCalc == "cos") {
53         result = cos(a);
54     }
55     else if (operationCalc == "tan") {
56         if (cos(a) == 0) {
57             cout << "that can't be done(doesn't exist)\n";
58             return 1;
59         }
60         else {
61             result = tan(a);
62         }
63     }
64     else if (operationCalc == "pow") {
65         int power;
66         cout << "to power of: "; cin >> power;
67         result = pow(a,power);
68     }
69 }
70 else {
71     cout << "wrong option, operator is undefined ";
72     return 1;
73 }
74
75 else {
76     cout << "wrong option, only ONE or TWO variables";
77     return 1;
78 }
79
80
81 if (variable_number == 2) {
82     cout << a << " " << operationCalc << " " << b << " = " << result;
83 }
84 else{
85     cout << operationCalc << "(" << a << ")" = " << result;
86 }
87 return 0;
88 }
89 int NumofVar() {
90     int VarNum;
91     cout << "how many variables do you need?( 1 or 2): "; cin >> VarNum;
92     return VarNum;
93 }

```

2.3. operation.cpp.

```
1  #include <iostream>
2  #include <string>
3  #include "CalcHeader.h"
4  using namespace std;
5  string operation() {
6      string operation;
7      cout << "operator: "; cin >> operation;
8      return operation;
9  }
```

2.4. main.cpp.

```
1  #include <iostream>
2  #include "CalcHeader.h"
3  int main()
4  {
5      OperationTypes();
6      Calculate(NumofVar());
7      return 0;
8  }
```

3.Выполнил компиляцию программы посредством g++:

```
g_zhuravlev@g ~/work/os/lab_prog g++ -c Calculator.cpp 1 1129 07:45:28
g_zhuravlev@g ~/work/os/lab_prog g++ -c main.cpp ✓ 1130 07:45:35
g_zhuravlev@g ~/work/os/lab_prog g++ -c operation.cpp ✓ 1131 07:45:42
g_zhuravlev@g ~/work/os/lab_prog g++ calculator.o main.o operation.o -o calculator
g++: error: calculator.o: No such file or directory
g_zhuravlev@g ~/work/os/lab_prog g++ Calculator.o main.o operation.o -o calculator
g_zhuravlev@g ~/work/os/lab_prog ✓ 1134 07:46:43
```

4. Создал Makefile.

```
1  CC = g++
2  CFLAGS = -g -c
3  CompFlags= -g -o
4
5  calculate: Calculator.o main.o operation.o
6          $(CC) Calculator.o main.o operation.o
7          $(CompFlags) calculate
8
9  Calculator.o: Calculator.cpp CalcHeader.h
10         $(CC) $(CFLAGS) Calculator.cpp
11
12  main.o: main.cpp CalcHeader.h
13         $(CC) $(CFLAGS) main.cpp
14
15  operation.o: operation.cpp CalcHeader.h
16         $(CC) $(CFLAGS) operation.cpp
17
18  clean:
19         rm -rf *.o calculate
```

5. Собрал Makefile.

```
g_zhuravlev@g ~/work/os/lab_prog > make -f Makefile
g++ -g -c Calculator.cpp
g++ -g -c main.cpp
g++ -g -c operation.cpp
g++ Calculator.o main.o operation.o -g -o calculate
g_zhuravlev@g ~/work/os/lab_prog >
```

В нём указаны :

1. CC - тип компилятора.
2. CFLAGS - опции, с которыми мы компилируем основной файл.
3. CompFlags - опции, с которыми мы компилируем остальные файлы.
4. Компиляция каждого зависящего файла .o
5. Сборщик всей программы "calculate"
6. Команда clean, для быстрого удаления всех файлов.

5. Провёл отладку с помощью GNUdebbuger(gdb).

5.1. Запустил отладчик GDB, загрузив в него программу для отладки.

5.2. Для запуска программы внутри отладчика ввел команду run.

```
g_zhuravlev@g /work/os/lab_prog gdb ./calculate
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calculate...
(gdb) run
Starting program: /home/g_zhuravlev/work/os/lab_prog/calculate
There are only few possible operations with ONE or TWO variables:
Basic: + ; - ; * ; / ; pow.
Trigonometric: sin; cos; tan.
Please, use only this operators, otherwise it won't work.
how many variables do you need?( 1 or 2): 1
variable: 7
operator: pow
to power of: 2
pow(7) = 49[Inferior 1 (process 127091) exited normally]
(gdb)
```

5.3. Использовала команду list.

```
(gdb) list
1      #include <iostream>
2      #include "CalcHeader.h"
3      int main()
4      {
5          OperationTypes();
6          Calculate(NumofVar());
7          return 0;
8      }
(gdb)
```

5.4. Для просмотра строк с 2 по 5 основного файла использовал list 2,5.

```
(gdb) list 2,5
2      #include "CalcHeader.h"
3      int main()
4      {
5          OperationTypes();
(gdb)
```

5.5. Для просмотра определённых строк не основного файла использовал list с параметрами: list Calculate

```

(gdb) list Calculator.cpp: 7,45
7      void OperationTypes() {
8          cout << "There are only few possible operations with ONE or TWO variables: \n";
9          cout << "Basic: + ; - ; * ; / ; pow.\nTrigonometric: sin; cos; tan.\n";
10         cout << "Please, use only this operators, otherwise it won't work.\n";
11     }
12
13     double Calculate(int variable_number) {
14         double a=0;
15         double b=0;
16         double result;
17         string operationCalc;
18         if (variable_number == 2) {
19             cout << "first variable: "; cin >> a;
20             operationCalc = operation();
21             cout << "second variable: "; cin >> b;
22             if (operationCalc == "+") {
23                 result = a + b;
24             }
25             else if (operationCalc == "-") {
26                 result = a - b;
27             }
28             else if (operationCalc == "*") {
29                 result = a * b;
30             }
31             else if (operationCalc == "/" ) {
32                 if (b == 0) {
33                     cout << "that can't be done(doesn't exist)\n";
34                     return 1;
35                 }
36                 else {
37                     result = a / b;
38                 }
39             }
40             else {
41                 cout << "wrong option, operator is undefined ";
42                 return 1;
43             }
44         }
45     }

```

5.6. Установил точку останова в файле Calculator.cpp на строке номер 18

5.7. Вывел информацию об имеющихся в проекте точках останова: info breakpoints

```

(gdb) break 18
Breakpoint 1 at 0x5555555553f6: file Calculator.cpp, line 18.
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint       keep y   0x00005555555553f6 in Calculate(int) at Calculator.cpp:18

```

5.8. Запустил программу внутри отладчика и убедился, что программа остановится в момент прохождения

```

(gdb) run
Starting program: /home/g_zhuravlev/work/os/lab_prog/calculate
There are only few possible operations with ONE or TWO variables:
Basic: + ; - ; * ; / ; pow.
Trigonometric: sin; cos; tan.
Please, use only this operators, otherwise it won't work.
how many variables do you need?( 1 or 2): 2
Breakpoint 1, Calculate (variable_number=2) at Calculator.cpp:18
18         if (variable_number == 2) {
(gdb)

```

5.8.1. Использовал backtrace.

```
(gdb) backtrace
#0 Calculate (variable_number=2) at Calculator.cpp:18
#1 0x0000555555555a61 in main () at main.cpp:6
```

- 5.9. Посмотрел, чему равно на этом этапе значение переменной a, введя: print a.
- 5.10. Сравнил с результатом вывода на экран после использования команды: display a.
- 5.11. Убрал точки останова: info breakpoints -> delete 1.

```
(gdb) print a
$1 = 0
(gdb) display a
1: a = 0
(gdb) info breakpoints
Num      Type      Disp Enb Address          What
1        breakpoint keep y  0x0000555555555f6 in Calculate(int) at Calculator.cpp:18
breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb)

(gdb) n
second variable: 5
22                                     if (operationCalc == "+") {
1: a = 5
2: operationCalc = "+"
3: b = 5
```

6. С помощью утилиты cppcheck проанализировал коды файлов(всю директорию).

```
g_zhuravlev@ /work/os/lab_prog : cppcheck --work/os/lab_prog
Checking /home/g_zhuravlev/work/os/lab_prog/Calculator.cpp ...
1/3 files checked 86% done
Checking /home/g_zhuravlev/work/os/lab_prog/main.cpp ...
2/3 files checked 91% done
Checking /home/g_zhuravlev/work/os/lab_prog/operation.cpp ...
3/3 files checked 100% done
```

Вывод.

Благодаря этой лабараторной работе я приобрел простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C++ калькулятора с простейшими функциями.

Контрольные вопросы.

1. Информацию об этих программах можно получить с помощью функций -help и man.
2. Разработка приложений в UNIX:

1. создание исходного кода программы;(файл с необходимым расширением(и кодом)).
 2. сохранение различных вариантов исходников;
 3. анализ исходников; необходимо отслеживать изменения исходного кода.
 4. компиляция исходников и построение исполняемого модуля;
 5. тестирование и отладка; (проверка кода на наличие ошибок)
 6. сохранение всех изменений, выполняемых при тестировании и отладке.
 7. Загрузка версии исходников в систему контроля версий GIT.
3. Суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта. (п
4. Основное назначение этого компилятора заключается в компиляции всей программы и получении исполн
5. Утилита make нужна для исполнения команд из Makefile(-ов). (компиляции, очистки и тп).

6. Пример:

```
CC = compiler
CFLAGS = compiler flags( like -c or -g)
.....( and so on)
```

```
target1: dependencies ( for example: report: dependentFile.o ... dependentFile.o)
target2: dependencies
...
targetn: dependencies
<tab>(necessary) $(CC) dependentFile.o $(CFLAGS) report.
```

пример из лабараторной работы:

```
CC = g++
CFLAGS = -g -c
CompFlags= -g -o
```

```
calculate: Calculator.o main.o operation.o
    $(CC) Calculator.o main.o operation.o $(CompFlags) calculate
```

```
Calculator.o: Calculator.cpp CalcHeader.h
    $(CC) $(CFLAGS) Calculator.cpp
```

```

main.o: main.cpp CalcHeader.h
$(CC) $(CFLAGS) main.cpp

operation.o: operation.cpp CalcHeader.h
$(CC) $(CFLAGS) operation.cpp

clean:
rm -rf *.o calculate

```

7. Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем, переменных, а также внутренних регистров микроконтроллера и напряжений на выводах этой микросхемы.

8.

1. backtrace - вывод на экран путь к текущей точке останова.
2. break - установить точку останова (в качестве параметра может быть указан номер строки или название функции)
3. clear - удалить все точки останова в функции
4. continue - продолжить выполнение программы
5. delete (n) - удалить точку останова
6. display - добавить выражение в список выражений, значения которых отображаются при достижении
7. finish - выполнить программу до момента выхода из функции
8. info breakpoints - вывести на экран список используемых точек останова
9. info watchpoints - вывести на экран список используемых контрольных выражений
10. list - вывести на экран исходный код (в качестве параметра может быть указано название файла и ч
11. next - выполнить программу пошагово, но без выполнения вызываемых в программе функций
12. print - вывести значение указываемого в качестве параметра выражения
13. run - запуск программы на выполнение
14. set[variable] - установить новое значение переменной
15. step - пошаговое выполнение программы
16. watch - установить контрольное выражение, при изменении значения которого программа будет ост

9.

- 1) Запустил Makefile(компиляция и сборка);
- 2) Начал отладку(run);
- 3) Вывел содержимое основного файла и Calculator.cpp;
- 4) Установил точку останова в Calculator.cpp.
- 5) Продолжил выполнение(run);
- 6) Посмотрел используемые функции(на данный момент времени backtrace);
- 7) Использовал команды print & display;
- 8) Удалил точку останова;
- 9) Закончил отладку(ошибок не было найдено);

10. Реакция была нормальной(ошибок не было)

11. cppcheck; splint; cscope;

12.

- 1) Проверка корректности задания аргументов всех использованных в программе функций, а также ти
- 2) Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффе
- 3) Общая оценка мобильности пользовательской программы.