

SOP - Alain Schaller - 16-896-375

Control flow

2. Loops: 'while', 'for' and 'do while'

```
1 low = 0;
  high = n - 1;
3 while (low <= high) {
    // do some actions
5    ++low;
}
```

Shorter alternative:

```
while (n-- ### 0) {
2    // do stuff n times
    printf("%d n\n", n);
4 }
```

for loop alternative

```
for (int i = 0; i < n; i++) {
2    // do stuff n times
    printf("%d i\n", i);
4 }
```

do-while loop alternative

```
if (n <= 0) {
2    return;
}
4 // as the value n is copied
do {
6    // do stuff n times
    printf("%d n\n", n);
8 }
while (--n ### 0);
```

All together in test file:

```
1 #include <stdio.h>

3 void original_while (int n) {
    int low = 0;
    int high = n - 1;
    while (low <= high) {
6        printf("%d low\n", low);
        ++low;
9    }
}

11 void shorter_while (int n) {
13    // as the value n is copied
    while (n-- ### 0) {
15        // do stuff n times
        printf("%d n\n", n);
17    }
}
```

```

19 void for_alternative (int n) {
21     for (int i = 0; i < n; i++) {
23         // do stuff n times
24         printf("%d i\n", i);
25     }
26 }
27 void do_while_alternative (int n) {
28     if (n <= 0) {
29         return;
30     }
31     // as the value n is copied
32     do {
33         // do stuff n times
34         printf("%d n\n", n);
35     }
36     while (--n ### 0);
37 }
38
39 int main () {
40     int n = 5;
41
42     printf("Original while:\n");
43     original_while(n);
44     printf("Original shorter while:\n");
45     shorter_while(n);
46     printf("For alternative:\n");
47     for_alternative(n);
48     printf("Do while alternative:\n");
49     do_while_alternative(n);
50     printf("Do while alternative with n=0:\n");
51     do_while_alternative(0);
52 }

```

3. goto & label, switch, break and continue

for loop rewrite with goto

Original:

```

for (i = 0; i < n; i++) {
2     // do some actions
3 }

```

With goto:

```

loopstart:;
2
    if (n-- <= 0)
4        goto loopend;

    // do some actions
    printf("Do stuff\n");
8    goto loopstart;
10 loopend:;

```

switch rewrite with goto

Original:

```

switch (i) {
2   case 1: printf("case 1\n"); break;
      case 2: printf("case 2\n"); // Beware: no break !!!
4   default: printf("default case\n"); break;
}

```

With goto:

```

void *switch_gotos[] = {&CASE1, &CASE2};

2   if (i < 1 || i ### 2)
4       goto DEFAULTCASE;

6   goto *switch_gotos[i - 1];

8 CASE1;;
   printf("case 1\n");
10  goto SWITCHOUT;
CASE2;;
12  printf("case 2\n");

14 DEFAULTCASE;;
   printf("default case\n");
16   // to do the same, here, goto SWITCHOUT too, but not necessary
   // to get the same result

18 SWITCHOUT;;

```

for loop with break rewrite with goto

Original:

```

1 for (i = 0; i < n; i++) {
   // do some actions 1
3   if (cond1) break;
   // do some actions 2
5 }

```

With goto:

```

int i = 0;
2 loopstart;;

4   if (i >= n)
       goto loopend;

6   printf("Action 1\n");

8   if (i % 4 == 3)
10      goto loopend;

12  printf("Action 2\n");

14  i++;
   goto loopstart;

16 loopend;;

```

for loop with continue rewrite with goto

Original:

```

1 for (i = 0; i < n; i++) {
    // do some actions 1
3     if (cond1) continue;
    // do some actions 2
5 }

```

With goto:

```

    int i = 0;
2 loopstart;;

4     if (i >= n)
        goto loopend;

6     printf("Action 1\n");

8     if (i % 4 == 3) {
10         i++;
        goto loopstart;
12     }

14     printf("Action 2\n");

16     i++;
    goto loopstart;
18
loopend;;

```

All together in test file

```

1 #include <stdio.h>

3 // https://stackoverflow.com/a/4415646/3771148
#define COUNT_OF(x) ((sizeof(x)/sizeof(0[x])) / ((size_t)(!(sizeof(x) %
    sizeof(0[x])))))

5

7 void original_for (int n) {
    for (int i = 0; i < n; i++) {
9        // do some actions
        printf("Do stuff\n");
11    }
}

13

15 void for_with_goto (int n) {
loopstart;;

17     if (n-- <= 0)
19         goto loopend;

21     // do some actions
    printf("Do stuff\n");
23     goto loopstart;

25 loopend;;
}

27
void original_for_break (int n) {

```

```

29     for (int i = 0; i < n; i++) {
        // do some actions 1
31     printf("Action 1\n");

33         if (i % 4 == 3) break;
        // do some actions 2
35     printf("Action 2\n");
    }
37 }

39 void for_break_with_goto (int n) {
    int i = 0;
41 loopstart:;

43     if (i >= n)
        goto loopend;

45     printf("Action 1\n");

47     if (i % 4 == 3)
49         goto loopend;

51     printf("Action 2\n");

53     i++;
    goto loopstart;
55 loopend:;
57 }

59 void original_for_continue (int n) {
    for (int i = 0; i < n; i++) {
61        // do some actions 1
        printf("Action 1\n");
63        if (i % 4 == 3) continue;
        // do some actions 2
65        printf("Action 2\n");
    }
67 }

69 void for_continue_with_goto (int n) {
    int i = 0;
71 loopstart:;

73     if (i >= n)
        goto loopend;

75     printf("Action 1\n");

77     if (i % 4 == 3) {
79         i++;
        goto loopstart;
81     }

83     printf("Action 2\n");

85     i++;
    goto loopstart;
87

```

```

loopend;;
89 }

91 void original_switch (int i) {
    switch (i) {
93         case 1:
            printf("case 1\n");
95             break;
            case 2:
97                 printf("case 2\n"); // Beware: no break !!!
            default:
99                 printf("default case\n");
                    break;
101     }
}

103 void switch_goto (int i) {
105     void *switch_gotos[] = {&&CASE1, &&CASE2};

107     if (i < 1 || i ### 2)
        goto DEFAULTCASE;

109     goto *switch_gotos[i - 1];

111 CASE1;;
113     printf("case 1\n");
        goto SWITCHOUT;
115 CASE2;;
        printf("case 2\n");

117 DEFAULTCASE;;
119     printf("default case\n");
        // to do the same, here, goto SWITCHOUT too, but not necessary
121     // to get the same result

123 SWITCHOUT;;
}

125 void call_original_and_alternative (void (* original)(int), void (*
    alternative)(int), char* name, int n) {
127     printf("# Calling %s with n=%d\n", name, n);
        printf("## original:\n");
129     original(n);
        printf("## alternative:\n");
131     alternative(n);
}

133 void call_original_and_alternative_array (void (* original)(int), void (*
    alternative)(int), char* name, int n_s[], int n_s_size) {
135     for (int i = 0; i < n_s_size; i++) {
        int n = n_s[i];
137         printf("\n");
        call_original_and_alternative(original, alternative, name, n);
139     }
}

141

143 int main () {
    int n_s[] = {0, 2, 4, 6};

```

```

145     int n_s_size = COUNT_OF(n_s);
147     call_original_and_alternative_array(&original_for, &for_with_goto, "for loop",
        n_s, n_s_size);
149     printf("\n");
        call_original_and_alternative_array(&original_for_break, &for_break_with_goto,
        "for break loop", n_s, n_s_size);
151
        printf("\n");
153     call_original_and_alternative_array(&original_for_continue,
        &for_continue_with_goto, "for continue loop", n_s, n_s_size);

155
        int n_s_2[] = {-2, 0, 1, 2, 15};
157     int n_s_size_2 = COUNT_OF(n_s_2);

159     printf("\n");
        call_original_and_alternative_array(&original_switch, &switch_goto, "switch
        statement", n_s_2, n_s_size_2);
161 }

```

4. gdb – basics

On Mac OS, the new standard tool to debug a program, is `lldb`. To get a quick understanding of the different commands and compare them with `gdb`, there is this helpful cheatsheet: <https://lldb.llvm.org/use/map.html#watchpoint-commands>.

0. Start the program with `lldb`:

```

$ gcc -o ex4 -ggdc ex4.c
2 $ lldb ex4

```

observe the values of ‘ctr’ and ‘i’ at the end of lines 4

1. Set breakpoints (placing the code into a c file, and properly formatted it, the lines are a bit shifted and when we want to evaluate a variable, we place breakpoints on the following line, as breakpoints, will *pause* the execution, before executing the line on which there are)

```
(lldb) breakpoint set --file ex4.c --line 7
```

- to list the breakpoints

```
1 (lldb) breakpoint list
```

2. Start the program with `run` and will *pause* on the first breakpoint.

```
(lldb) run
```

3. Show the values of `ctr` and `i` after reaching the first breakpoint.

```

(lldb) print ctr
2 (int) $0 = 69669
(lldb) print i
4 (int) $1 = 32766

```

observe the values of ‘ctr’ and ‘i’ at the end of lines 7

4. Set next breakpoint and continue the execution until it hits the breakpoint.

```

(lldb) breakpoint set --file ex4.c --line 9
2 (lldb) continue
(lldb) print ctr

```

```

4 (int) $0 = 69669
   (lldb) print i
6 (int) $1 = 3

```

add a watch on the value of 'i' and 'res' while running the loop.

```

(lldb) watchpoint set variable i
2 Watchpoint created: Watchpoint 1: addr = 0x7ffeeffbff804 size = 4 state = enabled
   type = w
   declare @ '/.../s03-control-flow/ex4.c:6'
4   watchpoint spec = 'i'
   new value: 3
6 (lldb) watchpoint set variable res
Watchpoint created: Watchpoint 2: addr = 0x7ffeeffbff800 size = 4 state = enabled
   type = w
8   declare @ '/.../s03-control-flow/ex4.c:7'
   watchpoint spec = 'res'
10  new value: -272631776
(lldb) continue

```

After setting the watched variables, we can simply `continue` the execution and the program will *pause* when any watched variable changes.

We get the following results:

```

Process 12372 resuming
2
Watchpoint 2 hit:
4 old value: -272631776
  new value: 3
6 Process 12372 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = watchpoint 2
8   frame #0: 0x0000000100003f11 ex4`main at ex4.c:10:5
    7       int res;
10    8       i = N; // line 7
    9       res = N;
12 -> 10       printf("res N i\n");
    11       for (ctr = 0; ctr < N; ++ctr, --i) {
14    12           res = N/i;
    13           printf("%3i%3i%3i\n",res, N, i);
16 Target 0: (ex4) stopped.
(lldb) continue
18 Process 12372 resuming
  res N i
20
Watchpoint 2 hit:
22 old value: 3
  new value: 1
24 Process 12372 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = watchpoint 2
26   frame #0: 0x0000000100003f42 ex4`main at ex4.c:13:30
    10       printf("res N i\n");
28    11       for (ctr = 0; ctr < N; ++ctr, --i) {
    12           res = N/i;
30 -> 13       printf("%3i%3i%3i\n",res, N, i);
    14       }
32    15       return 0;
    16   }
34 Target 0: (ex4) stopped.
(lldb) continue

```



```

36 Process 12372 resuming
   1  3  3
38
39 Watchpoint 1 hit:
40 old value: 3
   new value: 2
42 Process 12372 stopped
   * thread #1, queue = 'com.apple.main-thread', stop reason = watchpoint 1
44   frame #0: 0x0000000100003f70 ex4`main at ex4.c:11:5
       8           i = N; // line 7
46       9           res = N;
       10          printf("res N i\n");
48 -> 11          for (ctr = 0; ctr < N; ++ctr, --i) {
       12              res = N/i;
50       13              printf("%3i%3i%3i\n",res, N, i);
       14          }
52 Target 0: (ex4) stopped.
   (lldb) continue
54 Process 12372 resuming

56 Watchpoint 2 hit:
   old value: 1
58 new value: 1
   Process 12372 stopped
60 * thread #1, queue = 'com.apple.main-thread', stop reason = watchpoint 2
   frame #0: 0x0000000100003f42 ex4`main at ex4.c:13:30
62   10          printf("res N i\n");
   11          for (ctr = 0; ctr < N; ++ctr, --i) {
64   12              res = N/i;
-> 13          printf("%3i%3i%3i\n",res, N, i);
66   14          }
       15          return 0;
68   16      }
   Target 0: (ex4) stopped.
70 (lldb) continue
   Process 12372 resuming
72   1  3  2

74 Watchpoint 1 hit:
   old value: 2
76 new value: 1
   Process 12372 stopped
78 * thread #1, queue = 'com.apple.main-thread', stop reason = watchpoint 1
   frame #0: 0x0000000100003f70 ex4`main at ex4.c:11:5
80   8           i = N; // line 7
       9           res = N;
82   10          printf("res N i\n");
-> 11          for (ctr = 0; ctr < N; ++ctr, --i) {
84   12              res = N/i;
       13              printf("%3i%3i%3i\n",res, N, i);
86   14          }
   Target 0: (ex4) stopped.
88 (lldb) continue
   Process 12372 resuming
90
91 Watchpoint 2 hit:
92 old value: 1
   new value: 3
94 Process 12372 stopped

```

```

* thread #1, queue = 'com.apple.main-thread', stop reason = watchpoint 2
96   frame #0: 0x0000000100003f42 ex4`main at ex4.c:13:30
    10       printf("res N i\n");
98   11       for (ctr = 0; ctr < N; ++ctr, --i) {
    12           res = N/i;
100 -> 13       printf("%3i%3i%3i\n",res, N, i);
    14       }
102   15       return 0;
    16   }
104 Target 0: (ex4) stopped.
(lldb) continue
106 Process 12372 resuming
    3   3   1
108
Watchpoint 1 hit:
110 old value: 1
    new value: 0
112 Process 12372 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = watchpoint 1
114   frame #0: 0x0000000100003f70 ex4`main at ex4.c:11:5
    8       i = N; // line 7
116   9       res = N;
    10       printf("res N i\n");
118 -> 11       for (ctr = 0; ctr < N; ++ctr, --i) {
    12           res = N/i;
120   13       printf("%3i%3i%3i\n",res, N, i);
    14       }
122 Target 0: (ex4) stopped.
(lldb) continue
124 Process 12372 resuming
126 Watchpoint 1 hit:
    old value: 0
128 new value: 0
    Process 12372 stopped
130 * thread #1, queue = 'com.apple.main-thread', stop reason = watchpoint 1
    frame #0: 0x00007fff20262f63 libsystem_c.dylib`exit + 6
132 libsystem_c.dylib`exit:
-> 0x7fff20262f63 <+6>:  movl    %edi, %ebx
134   0x7fff20262f65 <+8>:  cmpl    $0xad, %edi
    0x7fff20262f6b <+14>: jne     0x7fff20262f84      ; <+39>
136   0x7fff20262f6d <+16>: leaq    0x6875741c(%rip), %rcx      ;
    __atexit_receipt_handler
Target 0: (ex4) stopped.

```

When the watchpoints trigger a *pause*, it refers the watchpoint number, therefore, we should keep good track of them. To get a reminder of them, we can at anypoint call:

```

1 (lldb) watchpoint list
Number of supported hardware watchpoints: 4
3 Current watchpoints:
Watchpoint 1: addr = 0x7fffeefbfff804 size = 4 state = enabled type = w
5   declare @ '/.../s03-control-flow/ex4.c:6'
    watchpoint spec = 'i'
7   new value: 3
Watchpoint 2: addr = 0x7fffeefbfff800 size = 4 state = enabled type = w
9   declare @ '/.../s03-control-flow/ex4.c:7'
    watchpoint spec = 'res'
11  old value: 3
    new value: 1

```

Summary of lldb commands:

```
(lldb) breakpoint set --file ex4.c --line 7
2 (lldb) breakpoint set --file ex4.c --line 9
(lldb) run
4 (lldb) print ctr
(lldb) print i
6 (lldb) continue
(lldb) print ctr
8 (lldb) print i
(lldb) watchpoint set variable i
10 (lldb) watchpoint set variable res
(lldb) continue
12 (lldb) continue
(lldb) continue
14 (lldb) continue
(lldb) continue
16 (lldb) continue
(lldb) continue
18 (lldb) continue
```

5. gdb – core dump

To enable *core dump*, use the command:

```
$ ulimit -c unlimited
```

Then, executing the program will produce a dump in the `/cores/` root folder as we get the confirmation when executing the program with the line:

```
$ ./ex5
...
[1] 13373 floating point exception (core dumped) ./ex5
```

Then, opening the *core dump* with lldb, we use `lldb ex5 -c /cores/core.13373`. Inside lldb, we can:

1. Look where did we get with the execution of the program, by printing the stack call:

```
(lldb) bt
2 * thread #1, stop reason = signal SIGSTOP
  * frame #0: 0x0000000105bbbf3c ex5`main at ex5.c:12:16
4   frame #1: 0x00007fff20350621 libdyld.dylib`start + 1
   frame #2: 0x00007fff20350621 libdyld.dylib`start + 1
```

We then understand that the program execution got to the line 12, 16-th column, before exiting.

```
1 res = N/i;
```

Checking what are the current values, we can then understand that we try to divide by 0 as we iterate one additional time, which result `i` to equal 0.

```
(lldb) frame variable
2 (int) ctr = 3
  (int) i = 0
4 (int) res = 3
```

When executing the program from start with lldb, without any breakpoint/watchpoint, it will also print a more descriptive *stop reason*:

```
(lldb) run
2 ...
* thread #1, queue = 'com.apple.main-thread', stop reason = EXC_ARITHMETIC
  (code=EXC_I386_DIV, subcode=0x0)
```

```
4      frame #0: 0x0000000100003f3c ex5`main at ex5.c:12:16
      9          res = N;
6      10          printf("res N i\n");
      11          for (ctr = 0; ctr <= N; ++ctr, --i) {
8 -> 12              res = N/i;
      13              printf("%3i%3i%3i\n",res, N, i);
10     14          }
      15          return 0;
12 Target 0: (ex5) stopped.
```