SOP - Alain Schaller - 16-896-375

Control flow

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

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

Shorter alternative:

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

`for` loop alternative

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

`do-while` loop alternative

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

All together in test file:

```
#include <stdio.h>

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

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

```

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

void do_while_alternative (int n) {
    if (n <= 0) {
        return;
    }
    // as the value n is copied
    do {
        // do stuff n times
        printf("%d n\n", n);
    }
    while (--n ### 0);
}

int main () {
    int n = 5;

    printf("Original while:\n");
    original_while(n);
    printf("Original shorter while:\n");
    shorter_while(n);
    printf("For alternative:\n");
    for_alternative(n);
    printf("Do while alternative:\n");
    do_while_alternative(n);
    printf("Do while alternative with n=0:\n");
    do_while_alternative(0);
}
```

3. goto & label, switch, break and continue

for loop rewrite with goto

Original:

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

With goto:

```
loopstart:;

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

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

loopend:;
```

switch rewrite with goto

Original:

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

With goto:

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

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

    goto *switch_gotos[i - 1];

CASE1:;
    printf("case 1\n");
    goto SWITCHOUT;
CASE2:;
    printf("case 2\n");

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

SWITCHOUT:;
```

for loop with `break` rewrite with goto

Original:

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

With goto:

```c
    int i = 0;
loopstart:;

    if (i >= n)
        goto loopend;

    printf("Action 1\n");

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

    printf("Action 2\n");

    i++;
    goto loopstart;

loopend:;
```

for loop with `continue` rewrite with goto

Original:

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

With goto:

```c
    int i = 0;
loopstart:;

    if (i >= n)
        goto loopend;

    printf("Action 1\n");

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

    printf("Action 2\n");

    i++;
    goto loopstart;

loopend:;
```

All together in test file

```c
#include <stdio.h>

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


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


void for_with_goto (int n) {
loopstart:;

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

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

loopend:;
}

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

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

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

    if (i >= n)
        goto loopend;

    printf("Action 1\n");

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

    printf("Action 2\n");

    i++;
    goto loopstart;

loopend:;
}

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

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

    if (i >= n)
        goto loopend;

    printf("Action 1\n");

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

    printf("Action 2\n");

    i++;
    goto loopstart;

```

```
loopend:;
}

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

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

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

    goto *switch_gotos[i - 1];

CASE1:;
    printf("case 1\n");
    goto SWITCHOUT;
CASE2:;
    printf("case 2\n");

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

SWITCHOUT:;
}

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

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


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

    call_original_and_alternative_array(&original_for, &for_with_goto, "for loop",
       n_s, n_s_size);

    printf("\n");
    call_original_and_alternative_array(&original_for_break, &for_break_with_goto,
       "for break loop", n_s, n_s_size);

    printf("\n");
    call_original_and_alternative_array(&original_for_continue,
       &for_continue_with_goto, "for continue loop", n_s, n_s_size);


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

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

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
$ 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 evalute 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

```
(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
(int) $0 = 69669
(lldb) print i
(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
(lldb) continue
(lldb) print ctr
```

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

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

```
(lldb) watchpoint set variable i
Watchpoint created: Watchpoint 1: addr = 0x7ffeefbff804 size = 4 state = enabled
   type = w
    declare @ '/.../s03-control-flow/ex4.c:6'
    watchpoint spec = 'i'
    new value: 3
(lldb) watchpoint set variable res
Watchpoint created: Watchpoint 2: addr = 0x7ffeefbff800 size = 4 state = enabled
   type = w
    declare @ '/.../s03-control-flow/ex4.c:7'
    watchpoint spec = 'res'
    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

Watchpoint 2 hit:
old value: -272631776
new value: 3
Process 12372 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = watchpoint 2
    frame #0: 0x0000000100003f11 ex4`main at ex4.c:10:5
   7        int res;
   8        i = N; // line 7
   9        res = N;
-> 10       printf("res N i\n");
   11       for (ctr = 0; ctr < N; ++ctr, --i) {
   12           res = N/i;
   13           printf("%3i%3i%3i\n",res, N, i);
Target 0: (ex4) stopped.
(lldb) continue
Process 12372 resuming
res N i

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

```
Process 12372 resuming
  1  3  3

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

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

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

Watchpoint 2 hit:
old value: 1
new value: 3
Process 12372 stopped
```

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

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

Watchpoint 1 hit:
old value: 0
new value: 0
Process 12372 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = watchpoint 1
    frame #0: 0x00007fff20262f63 libsystem_c.dylib`exit + 6
libsystem_c.dylib`exit:
->  0x7fff20262f63 <+6>:  movl   %edi, %ebx
    0x7fff20262f65 <+8>:  cmpl   $0xad, %edi
    0x7fff20262f6b <+14>: jne    0x7fff20262f84             ; <+39>
    0x7fff20262f6d <+16>: leaq   0x6875741c(%rip), %rcx     ;
        __atexit_receipt_handler
Target 0: (ex4) stopped.
```

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

```
(lldb) watchpoint list
Number of supported hardware watchpoints: 4
Current watchpoints:
Watchpoint 1: addr = 0x7ffeefbff804 size = 4 state = enabled type = w
    declare @ '/.../s03-control-flow/ex4.c:6'
    watchpoint spec = 'i'
    new value: 3
Watchpoint 2: addr = 0x7ffeefbff800 size = 4 state = enabled type = w
    declare @ '/.../s03-control-flow/ex4.c:7'
    watchpoint spec = 'res'
    old value: 3
    new value: 1
```

Summary of `lldb` commands:

```
(lldb) breakpoint set --file ex4.c --line 7
(lldb) breakpoint set --file ex4.c --line 9
(lldb) run
(lldb) print ctr
(lldb) print i
(lldb) continue
(lldb) print ctr
(lldb) print i
(lldb) watchpoint set variable i
(lldb) watchpoint set variable res
(lldb) continue
(lldb) continue
(lldb) continue
(lldb) continue
(lldb) continue
(lldb) continue
(lldb) continue
(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
* thread #1, stop reason = signal SIGSTOP
  * frame #0: 0x0000000105bbbf3c ex5`main at ex5.c:12:16
    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.

```
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
(int) ctr = 3
(int) i = 0
(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
...
* thread #1, queue = 'com.apple.main-thread', stop reason = EXC_ARITHMETIC
   (code=EXC_I386_DIV, subcode=0x0)
```

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