DS-Systeme
Kapitel 8

Kontroll-strukturen
Teil 3

Inhaltsverzeichnis

Kontrollstrukturen – switch

Eine **switch**-Implementierung ist sehr effizient und basiert auf einer Sprungtabelle (**jump table**).

Es empfiehlt sich, **switch** erst dann zu verwenden, wenn in Verzweigungen so viele Fälle auftreten, dass die Verwendung von **if** zu unübersichtlich würde.

switch

C++-Code:

```
void switch_eg(long x, long n,
long * dest)
{
    long val = x;
    switch(n)
    {
    case 100:
        val *= 13;
        break;
    case 102:
        val += 10;
    // fall through
    case 103:
        val += 11;
        break;
    case 104:
    case 106:
        val *= val;
        break;
    default:
        val = 0;
    }
    *dest = val;
}
```

```
int main()
{
    long l;
    switch_eg(10, 50, &l);
    std::cout << "case =  50, result = "
              << l << '\n';
    switch_eg(10, 100, &l);
    std::cout << "case = 100, result = "
              << l << '\n';
    switch_eg(10, 102, &l);
    std::cout << "case = 102, result = "
              << l << '\n';
    switch_eg(10, 103, &l);
    std::cout << "case = 103, result = "
              << l << '\n';
    switch_eg(10, 104, &l);
    std::cout << "case = 104, result = "
              << l << '\n';
    switch_eg(10, 106, &l);
    std::cout << "case = 106, result = "
              << l << '\n';
}
```

```
Ausgabe:   case  50, result = 0
           case 100, result = 130
           case 102, result = 31
           case 103, result = 21
           case 104, result = 100
           case 106, result = 100
```

switch

C++-Code:

```
void switch_eg(long x, long n,
long * dest)
{
    long val = x;
    switch(n)
    {
    case 100:
        val *= 13;
        break;
    case 102:
        val += 10;
    // fall through
    case 103:
        val += 11;
        break;
    case 104:
    case 106:
        val *= val;
        break;
    default:
        val = 0;
    }
    *dest = val;
}
```

Assembler-Code, Teil 1:

```
 .globl switch_eg
  .type switch_eg, @function
# void switch_eg(long x, long n,
long *dest)
# x in rdi, n in rsi, dest in rdx
switch_eg:
  subq $100, %rsi
  cmpq $6, %rsi
  ja   .L8
  jmp *.L4(,%rsi,8)
.L3:
  leaq (%rdi,%rdi,2), %rax
  leaq (%rdi,%rax,4), %rdi
  jmp .L2
.L5:
  addq $10, %rdi
.L6:
  addq $11, %rdi
  jmp .L2
.L7:
  imulq %rdi, %rdi
  jmp .L2
.L8:
  movq $0, %rdi
.L2:
  movq %rdi, (%rdx)
  ret
```

switch

Assembler-Code, Teil 1, kommentiert:

```
 .globl switch_eg
  .type switch_eg, @function
# void switch_eg(long x, long n, long *dest)
# x in rdi, n in rsi, dest in rdx
switch_eg:
  subq $100, %rsi             #   Compute index = n-100
  cmpq $6, %rsi               #   Compare index:6
  ja  .L8                     #   If >, loc_def
  jmp *.L4(,%rsi,8)           #   Goto *jt[index]
.L3:                          # loc_A:
  leaq (%rdi,%rdi,2), %rax #   3*x
  leaq (%rdi,%rax,4), %rdi #   val = 13*x
  jmp .L2                     #   Goto done
.L5:                          # loc_B:
  addq $10, %rdi              #   x = x + 10
.L6:                          # loc_C:
  addq $11, %rdi              #   val = x + 11
  jmp .L2                     #   Goto done
.L7:                          # loc_D:
  imulq %rdi, %rdi            #   val = x * x
  jmp .L2                     #   Goto done
.L8:                          # loc_def:
  movq $0, %rdi               #   val = 0
.L2:                          # done:
  movq %rdi, (%rdx)           #   *dest = val
  ret                         #   Return
```

Assembler-Code, Teil 1:

```
 .globl switch_eg
  .type switch_eg, @function
# void switch_eg(long x, long n,
long *dest)
# x in rdi, n in rsi, dest in rdx
switch_eg:
  subq $100, %rsi
  cmpq $6, %rsi
  ja  .L8
  jmp *.L4(,%rsi,8)
.L3:
  leaq (%rdi,%rdi,2), %rax
  leaq (%rdi,%rax,4), %rdi
  jmp .L2
.L5:
  addq $10, %rdi
.L6:
  addq $11, %rdi
  jmp .L2
.L7:
  imulq %rdi, %rdi
  jmp .L2
.L8:
  movq $0, %rdi
.L2:
  movq %rdi, (%rdx)
  ret
```

In Teil 2 ist die Sprungtabelle inplementiert.

Assembler-Code, Teil 2:

```
  .section .rodata
   # Align address to multiple of 8
  .align 8
.L4:
  .quad .L3  # Case 100: loc_A
  .quad .L8  # Case 101: loc_def
  .quad .L5  # Case 102: loc_B
  .quad .L6  # Case 103: loc_C
  .quad .L7  # Case 104: loc_D
  .quad .L8  # Case 105: loc_def
  .quad .L7  # Case 106: loc_D
```

Aufgabe: Übersetzen Sie den Assembler-Code in C-Code

Hinweis: **%rcx** entspricht der C-Variablen **val**

```
.globl switcher
.type switcher, @function
#void switcher(long x, long n, long *dest)
#x in %rdi, n in %rsi, dest in %rdx

switcher:
        cmp $4, %rdi
        ja .case_def
        jmp *.s_table(, %rdi, 8)

.case_0:
        leaq 112(%rsi), %rcx
        jmp .s_end

.case_2_4:
        leaq (%rsi, %rdi), %rcx
        jmp .s_end

.case_def:
        movq %rdi, %rcx

.s_end:
        movq %rcx, (%rdx)
        ret
```

```
.section .bss
.lcomm dest, 8

.globl _start
.type _start, @function
_start:
    pushq %rbp
    movq %rsp, %rbp
    movq $0, %rdi
    movq $1, %rsi
    movq $dest, %rdx
    call switcher

    movq $60, %rax
    xor %rdi, %rdi
    popq %rbp
    syscall

.section .rodata

# jump table
.align 8
.s_table:
    .quad .case_0
    .quad .case_def
    .quad .case_2_4
    .quad .case_def
    .quad .case_2_4
```