Low level programming (37-023)

Programming in Assembly
Basics of Operating Systems
Machine models

Lecturer:
Dipl. Ing. Irina Chihaia

WebSite:
www.cs.inf.ethz.ch/37-023

Textbooks:
R. Paul: SPARC Architecture, Assembly Language Programming and C
B. Kernighan, D. Richie: The C Programming Language

Topic of Today:
• Introduction to C (con’t)
  • Operators in C
  • Control Structures in C
Operators

Assignments

11 \(<name> = \langleexpression\>

Short form of assignments (important)

13 \(<name> \langleoperator\>= \langleexpression\>
14
15 \(x += i; \quad /* equal to x = x + i */
16
17 \(x *= y + i;
18

Priorities among operators

19 \(<expression1> \op = <expression2>
20
21 \quad /* expression2 is evaluated first */
Overview Operators in C

primary and postfix expressions

16 names literals
16 $a[k]$  simple tokens
16 $f(\ldots)$  subscripting
16 .  function call
16 $\rightarrow$  direct selection
15 $++ --$  indirect selection

postfix increment / decrement

unary operators

14 $++ --$  prefix increment / decrement
14 sizeof  size
14 (type name)  casts (type conversion)
14 -  bitwise not
14 !  logical not
14 -  arithmetic negation
14 &  address of
14 *  indication

binary and ternary operators

13L * / \\  multiplicative
12L + -  additive
11L << >>  left and right shift
10L < > <= >=  relational
9L == !=  equality / inequality
8L &  bitwise and
7L ^  bitwise xor
6L |  bitwise or
5L &&  logical and
4L ||  logical or
3R ? :  conditional (ternary)
2R = += -= *% /= %=  assignment
2R <<= >>= &= ^= |=  sequential evaluation

The numbers on the left indicate relative precedence; larger numbers indicate higher precedence. "L" indicates left-associative operators and "R" indicates right-associative operators.
Special Operators

Comma operator separates expressions

```c
x = (y=3, y+1);  /* y=3; x=y+1; */
```

(dot) and -> allow access to the elements of structures and unions.

```c
struct Person {
    char name[80], vorname[80];
    int jahrgang;
}
struct Person pers, *pers_ptr;

pers_ptr = &pers;
pers.vorname[0] = 'P';
pers_zeiger->vorname[1] = 'a';
/* operator “content”, dereferencing operator/```

• () determine order of evaluation.

• [] access an array element.

• Conditional assignment in expression

```c
if (a > b)
    z = a;
else
    z = b;
```

 equivalent to

```c
z = (a > b) ? a : b
```

```
8.11.00 - 4 37-023 Systemprogrammierung © Stricker
```
Control Structures in C

Statements in C:

• a <statement> is one instruction or a block of multiple instructions {............}

• a <statement> is terminated by “;” and not separated by “;” like in e.g. Oberon.

Purpose of Control Structures

• Control of the program/instruction flow

• Four rules of style for control structs:
  • Use function call hierarchically. Do step-wise refinement of complex calculations through nested function calls.
  • Keep functions short and stick to a single functionality (unless you are desperate to optimize for speed).
  • Keep functions simple. Pass data only by parameters between functions. Avoid global variable!
  • This is not BASIC - do not use goto’s!
if-Statement

16 if (expression) <statement>;
17 else <statement>;

• Caveat: Nested if’s... Unlike you use braces the else belongs to the last if. (the C syntax does have a dangling else problem).

18 if (x)
19   if (y) printf("....");
20   else printf("....");
21
22 if (x) {
23   if (y) printf("....");
24 }
25 else printf("....");
switch-Statement

• Syntax of switch statement

```java
1 switch (<expression>) {
2     case <const.1> : <statement 1>;
3             break;
4     case <const.2> : <statement 2>;
5             break;
6     case <const.3> : <statement 3>;
7             break;
8     default :        <statement x>;
9             break;
10 }
```

• Replaces sequences of if ... else ... if

• Result of `<expression>` must be of type `int` or `char`.

• `break` statement causes jump to the end of the switch statement.

• `default` is optional and is invoked when the value of the switch expression is not matched by any of the cases.
for-Loop

• is more flexible than in other languages and is used as follows:

1    for (<expr1>;<expr2>;<expr3>) <statement>;
2        /* e.g. for(;;);for(i=1;i<5;i++) */
3 ...
4    for (<initialization>;
5        <end condition>;
6        <increment>)
7        <statement>;

• used in particular when assignment, conditions are simple expressions and the variable is simply in/decremented.

• could also do more complex loops, equivalent to while loop...

8    for ( 
9        printf(" press ´q´ and RETURN to quit"),
10        c = getchar(); c != EOF, 
11        printf(" press ´q´ and RETURN to quit"));
12        c = getchar()
13    )

• discouraged - use while loop instead!
while-Loop

```c
1 void main() {
2     char t;
3
4     while((t = getchar()) != '!') {
5         if (t >= 'A' && t <= 'Z')
6             printf("%c
", (char)(t+'a'-'A'));
7         else
8             printf("%c
",(char)t);
9     }
10 }
```

- While is used with checks for more complex termination conditions.

- Equivalence of loop and while

```c
11 for (<expr1>;<expr2>;<expr3>) <statement>;
12 <expr1>
13 while (<expr2>) {
14     <statement>;<expr3>;
15 }
```

- Used for 0..n iterations.
do-while-Loop

• pushing the termination condition to the end produced control structure that is different from while and for loop.

• Syntax:

```
1  do {
2       <statement>
3  } while (<expression>);
```

• Used for 1..n iterations.
exit()

- is actually a function that is imported from C standard library (libc.a).

- terminates the program execution and returns control to operating system, passing an integer error code. e.g. 0 -> no error, 1 -> not found, 99 -> crash.

- is used to resolve error condition e.g. failure to open file, to allocate memory.

```c
1   if (!(buf = AllocMem (BufSize);)) {
2       printf(“kein Speicher vorhanden”);
3       exit(NO_MEM);
4   }
```

- more civilized programming languages (Java, C++) do have exception handling.

goto()

C does have a goto, use is problematic. At the end of the semester we will know why... knowledge of stack frames