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> = <expression>

Short form of assignments (important)

13 <name> <operator> = <expression>
14 x += i; /* equal to x = x + i */
15 x *= y + i;

Priorities among operators

19 <expression1> op = <expression2>
20 /* expression2 is evaluated first */

Overview Operators in C

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>(dot) access to elements of structures and unions.</td>
</tr>
<tr>
<td>-</td>
<td>arithmetic negation</td>
</tr>
<tr>
<td>*</td>
<td>address of variable</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>%</td>
<td>remainder</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>(</td>
<td>precedence</td>
</tr>
<tr>
<td>)</td>
<td>precedence</td>
</tr>
<tr>
<td>[</td>
<td>index of array</td>
</tr>
<tr>
<td>]</td>
<td>index of array</td>
</tr>
<tr>
<td>-&gt;</td>
<td>dereferencing operator</td>
</tr>
<tr>
<td>,</td>
<td>comma operator</td>
</tr>
</tbody>
</table>

Special Operators

Comma operator separates expressions

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

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

1 struct Person {
2     char name[80], vorname[80];
3     int jahrgang;
4 }
5 struct Person pers, *pers_ptr;
6 pers_ptr = &pers;
7 pers.vorname[0] = 'P';
8 pers_zeiger->vorname[1] = 'a';
9 /* operator "content", dereferencing operator/ 

() determine order of evaluation.

[] access an array element.

Conditional assignment in expression

10 if (a > b)
11 \( z = a; \)
12 else
13 \( z = b; \)
14 equivalent to
15 \( <expr1> \ ? <expr2> : <expr3> \)
Control Structures in C

Statements in C:

- a <statement> is one instruction or a block of multiple instructions {............}
- a <statement> is terminated by “;” an 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

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
main() {
    char t;
    while((t = getchar()) != '!') {
        if (t >= 'A' && t <= 'Z')
            printf("%c\n", (char)(t+'a'-'A'));
        else
            printf("%c\n", (char)t);
    }
}
```

- While is used with checks for more complex termination conditions.
- Equivalence of loop and while

```c
for (<expr1>; <expr2>; <expr3>) <statement>;
```

```c
<expr1>
while (<expr2>) {
    <statement>;
    <expr3>;
}
```

- 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:

```c
do {
    <statement>
} 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
if (!(buf = AllocMem (BufSize);)) {
    printf("kein Speicher vorhanden");
    exit(NO_MEM);
}
```

- 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