16.070 Introduction to Computers and Programming

February 22 Recitation 3 Spring 2001

Coding examples included at the end of this document

Topics

Operators in C
- Operator Properties

Control Structures
- Conditional Branching
- Pretest Repetition
- Posttest Repetition

Programming Guide
- Arithmetic Assignment
- If Statements
- Switch Statements
- Repetition Statements

Operators in C

Operator Properties
When parentheses are not used to explicitly indicate the grouping of operands and operators, precedence is used to determine the order of operations. The following table lists some of the C operands and their properties.

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Operator</th>
<th>Associativity</th>
<th>Class</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>array subscripting</td>
<td>→</td>
<td>postfix</td>
<td>15</td>
</tr>
<tr>
<td>(...)</td>
<td>function call</td>
<td>postfix</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>-&gt; .</td>
<td>member reference</td>
<td>postfix</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>- +</td>
<td>unary minus, plus</td>
<td>←</td>
<td>unary</td>
<td>14</td>
</tr>
<tr>
<td>++ , --</td>
<td>increment, decrement</td>
<td>postfix</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>++ , --</td>
<td>increment, decrement</td>
<td>prefix</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>logical not</td>
<td></td>
<td>unary</td>
<td>13</td>
</tr>
<tr>
<td>&amp; , *</td>
<td>address of, indirection</td>
<td></td>
<td>unary</td>
<td>13</td>
</tr>
<tr>
<td>(type name)</td>
<td>casts</td>
<td></td>
<td>unary</td>
<td>12</td>
</tr>
<tr>
<td>* / %</td>
<td>multiplicative</td>
<td>→</td>
<td>binary</td>
<td>11</td>
</tr>
<tr>
<td>+ -</td>
<td>additive</td>
<td></td>
<td>binary</td>
<td>10</td>
</tr>
<tr>
<td>&lt; &gt; &lt;= !=</td>
<td>relational</td>
<td></td>
<td>binary</td>
<td>9</td>
</tr>
<tr>
<td>== !=</td>
<td>equality, inequality</td>
<td></td>
<td>binary</td>
<td>8</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>logical and</td>
<td></td>
<td>binary</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical or</td>
<td></td>
</tr>
<tr>
<td>+= -= *=</td>
<td>assignment</td>
<td>←</td>
<td>binary</td>
<td>2</td>
</tr>
<tr>
<td>/= %= &amp;=</td>
<td>sequential evaluation</td>
<td>→</td>
<td>binary</td>
<td>1</td>
</tr>
</tbody>
</table>

Associativity is the order of operations if adjacent operators have the same precedence.
**Program Control Structures**

### Conditional Branching (If-statement and switch-statement)

If-statements can be used in many different ways. They should always be used for simple two-way selections. When considering multi-way selection and the control expression has a >, <, >=, or <= you should use nested if statements or a series of if statements. Be careful when using nested if-statements, the logic becomes hard to follow if there are too many selections. Nested if-statements can also be used if the multi-way selection structure depends on more than one variable.

```c
if (thrust_state == 1)
    if (fuel_level <= 0.25)
        printf("Fuel level is low\n\n");
```

You can also use complex control expressions by using the logical operators to combine the if-statements. You should not normally use more than two different control expressions in one if statement (Good style!).

```c
if (thrust_state == 1 && fuel_level <= 0.25)
    printf("Fuel level is low")
```

When your multi-way selection structure depends on only one variable, and that variable can only be equal to integral values (integers and characters), you should use the switch statement. Make sure to place the break commands in the correct location in order to get the desired outcome.

```c
switch (control_command) {
    case 'l':
        command = left_turn;
        break;
    case 'r':
        command = right_turn;
        break;
    case 'u':
        command = climb;
        break;
    case 'd':
        command = descend;
        break;
    default:
        command = none;
}
```

A break statement stops the program pointer in its tracks and moves it to the outside of the current layer of “nesting”. This means that it effectively “breaks”/(exits from) the most local loop or nest.

You’ll find that most experienced programmers frown upon the use of break commands within loops or branching structures. In this course, we will use breaks only in switch statements. The question remains: Why doesn’t a switch statement already break whenever a “case” is satisfied? Well, C allows room for more elaborate switch statements like:

```c
switch (my_age) {
    case 88:
        printf("Booyah Grandma!\n\n");
    case 21:
        printf("I've seen you at the local Pub!\n\n");
    case 18:
        printf("You're not in High School anymore!\n\n");
    case 16:
        printf("You have a drivers license!\n\n");
    case 4:
        printf("You can already talk!\n\n");
    case 2:
        printf("You can already walk!\n\n");
    default:
        printf("That will be $12 dollars!\n\n");
}
```
Pretest Repetitions

One of the first questions that arises once you decide that you need a loop is, which loop do you use? This question has no definitive answer. Anything that you can do with a for loop you can also do with a while loop. Therefore it is mostly a matter of preference. There is one factor that can make your decision easier. The repetition structure will either be counter or sentinel controlled. A counter controlled repetition structure is one where the number of times that the loop must be executed is already known. It is most commonly and easily coded using the for loop. The following example demonstrates the use of a for loop to compute a factorial:

```c
for (i = 1; i <= number; i++) {
    factorial = factorial * i;
}
```

A sentinel controlled repetition structure is one where the number of times that the loop is executed is dependent on the value of a variable. These are most commonly and easily coded using the while loop. The following example demonstrates the use of a while loop to calculate the amount of fuel remaining when the thrusters are firing:

```c
thrust_state = get_thrust_state();
while (thrust_state) {
    fuel_level = /* however you want to calculate it */;
    thrust_state = get_thrust_state();
}
```

The fact still remains that the for and while loops are for the most part interchangeable. Below are the two examples written again using the other form of loop:

```c
i = 0;
while (i <= number) {
    factorial = factorial * i;
    i++;
}
```

As you can see the factorial loop is more compact and concise in the for loop format and the fuel level loop is easier to understand in the while format.

Posttest Repetition

In general the posttest repetition structure is not used over the pretest structure for a couple of reasons. Code is easier to read when the conditions are at the beginning of the loop, especially in a long loop. In the posttest structure the loop is executed first and this can cause unexpected results in some cases. It is important to watch the order of statements in a do-while loop. For example if we write the factorial loop in the do-while format as follows:

```c
i = 1;
do {
    factorial = number * i;
    i++;
} while(i <= number);
```
it performs correctly, however in a more complex loop it would be easy to have the counter before the evaluation statement and in that case the array would have been initialized to array_size + 1. This type of error is avoided by using the for loop.

Nonetheless, it is possible to do all of the above designs using the do while loop.

**Programming Guide**

**Arithmetic Assignment Expressions**

- Use parentheses to change the order in which the operators are evaluated and to ensure that the order of operations is what you want.
- Avoid calls to mathematical functions for simple expressions.
- Avoid using increment and Decrement Operators in complex statements.
- Avoid using long expressions, split long expressions into multiple smaller ones.

**If Statements**

- Indent the statements under the *if* and the *else* commands.
- Avoid deeply nested *if* statements, the logic becomes hard to understand.
- Choose easy to understand conditional expressions.
- Avoid logical negation in expressions whenever possible.
- Don’t make logical expressions too complex, no more than to 2 evaluations.
- Place a blank line before and after the statement.

**Switch Statements**

- Remember a nested *if* statement is more general than a *switch*.
- Rule of thumb, use *switch* for 3 – 10 alternative selections.
- Use proper indentation when forming *switch* statements, indent the word *case* and then indent the statements below each *case*.
- Always use the *default* clause in switch statements, works well for error handling.
- Avoid having different clauses perform the same operation.
- Use the *break* statement to ensure proper selection.
- Place a blank line before and after the statement.

**Repetition Statements**

- Indent the loop body of *while* and *for* statements.
- Choose easy to understand conditional expressions.
- Avoid logical negation in expressions whenever possible.
- Place a blank line before and after the statement block.
- Always include; initialization statement, loop control expression and update expression in *for* statement.
Coding Examples

The If-statement

#include <stdio.h>

int main(void) { /* This program illustrates the use of nested if-statements */
    int thrust_state=1;
    double fuel_level=0.8;
    if (thrust_state == 1) { /* Check for thrust on */
        if (fuel_level <= 0.25) /* Check if fuel level low */
            printf("Fuel level is too low for thrust!\n\n");
        else
            printf("Fuel level is good for thrust!\n\n");
    }
    else if (thrust_state ==0) { /* Check for thrust off */
        /* Check if fuel level high */
        if ( (fuel_level <=1)&&(fuel_level >0.25) )
            printf("No thrust, Fuel is good for thrust!\n\n");
        else /* Check for thrust error */
            printf("Fuel level is too low for thrust!\n\n");
    }
    else
        printf("Thrust variable error");
    return 0;
} /* end main */

The Case-statement

(With break)

#include <stdio.h>

int main(void) { /* Switch statements with breaks */
    int my_age=18;
    switch (my_age) { /* The psychic (s)witch */
        case 88 : printf("Booyah Grandma!\n\n");
            break;
        case 21 :
printf("I've seen you at the local Pub!

");
break;
case 18 :
    printf("You're not in High School anymore!

");
    break;
case 16 :
    printf("You have a drivers license!

");
    break;
case 4 :
    printf("You can already talk!

");
    break;
case 2 :
    printf("You can already walk!

");
    break;
default:
    printf("That will be $12 dollars!

");
}

return 0;
} /* end main */

(WITHOUT break)

/****************************
* TJ Feb 2001    *
* Examples galore!  *
* Recitation 3    *
****************************/

/* A user categorizes a caller as older than a specific age. This age is
entered and information is inferred from it by the switch/case-statements.
This information is then displayed
/* All applicable lines of information are displayed on the screen, including
cost of information */
#include <stdio.h>

int main(void) /* Switch statements without breaks */
{
    int my_age=88;
    switch (my_age) { /* The psychic (s)witch */
        case 88 :
        case 21 :
            printf("I've seen you at the local Pub!

");
        case 18 :
            printf("You're not in High School anymore!

");
        case 16 :
            printf("You have a drivers license!

");
            break;
        case 4 :
            printf("You can already talk!

");
        case 2 :
            printf("You can already walk!

");
            break;
        default:
            printf("That will be $12 dollars!

");
    }
    return 0;
} /* end main */
Equivalent while- and for-loops

#include <stdio.h>

int main(void)           /* for and while equivalency */
{
    int thrust_state = 1;
    /* Stay in either of the following loops as long as thrust is on */
    /* Only run one loop at a time. Keep the other in comments */
    /*while (thrust_state) {
        printf("Thrust On\n\n");
        printf("Enter new thrust state:");
        scanf("%d", &thrust_state);
    }*/
    for (thrust_state = 1; thrust_state == 1; scanf("%d", &thrust_state)) {
        printf("Thrust On\n\n");
        printf("Enter new thrust state:");
    }
    printf("Thrust Off\n\n");
    return 0;
} /* end main */