CSE 1310 - Introduction to Computers & Programming

Expressions, Statements, and Basic Control

Alex Dillhoff

University of Texas at Arlington

Expressions and Statements

An **expression** is a sequence of tokens that evaluates to a numerical quantity.

Expressions and Statements

An **expression** is a sequence of tokens that evaluates to a numerical quantity.

- `1 + 1;`
- `y = 3;`
- `1;`
- `46 + 8 / (20 / (1 * 0.5));`
- `x * 2;`

Expressions and Statements

A **statement** is anything that can be evaluated by the compiler.

Expressions and Statements

A **statement** is anything that can be evaluated by the compiler.

- `int x;`
- `x;`
- `y = 3;`
- `int y = x * 2;`
- `;`

lvalues and rvalues

An **lvalue** is an expression with a location in memory.

lvalues and rvalues

An **lvalue** is an expression with a location in memory.

- `x`
- `myVar`
- `v_ptr`
- `avg_height`

lvalues and rvalues

An **rvalue** is any expression not representing some location in memory.

lvalues and rvalues

An **rvalue** is any expression not representing some location in memory.

- `x + 3;`
- `5;`
- `-1 * y + (-x);`
- `1 / 0.5;`

lvalue and rvalue Examples

- lvalues can be used on either side of an assignment.
- rvalues can only be used on the right side of an assignment.

Blocks and Compound Statements

A **block** is a variable number of statements contained between braces.

```
{
    x = 3;
    int y = x * 0.5;
}
```

Blocks and Compound Statements

Groups of statements contained within a block are called **compound statements**.

```
{
    float x = 0.1;
    {
        int y = x * 3;
        printf("%d %d\n", x, y);
    }
}
```

Blocks and Scope

Variables defined in a block are called **local variables**.

```
{
    int x = 1;
    {
        float y = 2.9;

        printf("x = %d\n", x);
    }
    printf("%d\n", y);
}
```

Blocks and Scope

What is the printed value of y in the previous example?

Blocks and Scope

What is the printed value of y in the previous example?

y does not exist.

Local variables only exist within their scope.

A note of formatting

In the previous examples, it was easy to tell which statements belonged to which block. The statements of each block were indented.

The number of spaces to use for indentation can change from group to group, but it is generally considered good formatting to indent statements such that they visually represent their **scope**.

Adding Control – if-else Statements

The syntax of control statements is very similar throughout many languages. Basic control can be added to the program through `if` and `else` statements.

Adding Control – if-else Statements

```
scanf("%d", &a);
if (a == 0) {
    printf("The value is 0.\n");
} else {
    printf("The value is not 1.\n");
}
```

Adding Control - if-else Statements

Example: Checking input commands from a menu.

```
char choice;
scanf("%c", &choice);
if (choice == 'q') {
    printf("Exiting the program.\n");
} else if (choice == 'l') {
    printf("Load option selected.\n")
} else if (choice == 'p') {
    printf("Print option selected.\n");
} else {
    printf("Invalid command.\n");
}
```

if-else Examples

- Checking `scanf`
- Even/odd check
- ID check

Adding Control - switch Statement

Another way to represent the previous example.

```
switch (choice) {
    case 'q':
        printf("Exiting the program.\n");
    case 'l':
        printf("Load option selected.\n");
    case 'p':
        printf("Print option selected.\n");
    default:
        printf("Invalid command.\n");
}
```

Adding Control - switch Statement

The previous example prints everything else below the option that is selected... **WHY?**

Adding Control - switch Statement

The previous example prints everything else below the option that is selected... **WHY?**

The statements are executed sequentially within the **block**.

Breaking out of switch

We can prevent sequential execution with break.

```
switch (choice) {
    case 'q':
        printf("Exiting the program.\n");
        break;
    case 'l':
        printf("Load option selected.\n");
        break;
    case 'p':
        printf("Print option selected.\n");
        break;
    default:
        printf("Invalid command.\n");
        break;
}
```

Breaking out of switch

A `break` statement allows the control to exit the switch statement.

It also allows us to break out of loops (more on that later).

switch Examples

- Pyramid

Relational Operators

- <= is less than or equal to
- >= is greater than or equal to
- == is equal to
- != is not equal to
- < is less than
- > is greater than

Relational Operators – Examples

- Check if the input matches a required value.
- Verify that an input is within an acceptable range.

Equality and Assignment

Common mistake when programming:

```
if (a = 1) {
    printf("Is this true?\n");
}
```

Equality and Assignment

Common mistake when programming:

```
if (a = 1) {
    printf("Is this true?\n");
}
```

This will **always** be true.

Operator Precedence

Operator Precendence Chart

Logical Operators

- `!` logical NOT
- `&&` logical AND
- `||` logical OR

Logical operators are used to test truth values between expressions or to negate an expression.

- `a * b && c - a`
- `a || b`
- `!TRUE`

Short-circuit Evaluation

Expressions are evaluated from left to right, if a condition is met then the resulting expressions are not evaluated.

Example 1

```
expr1 || expr2
```

If `expr1` is true, then `expr2` will not be evaluated.

Short-circuit Evaluation

Expressions are evaluated from left to right, if a condition is met then the resulting expressions are not evaluated.

Example 2

```
expr1 && expr2
```

If `expr1` is false, then `expr2` will not be evaluated.

Logical Operators – Examples

- Verify that the username and password match.
- Verify that an input is within an acceptable range.

The Ternary Operator

C has a ternary (three inputs) operator that allows us to write a conditional expression in a single line.

```
expr1 ? expr2 : expr3
```

The Ternary Operator

```
expr1 ? expr2 : expr3
```

- `expr1` is first evaluated.
- If it is true, `expr2` is evaluated and becomes the result of the expression.
- If it is false, `expr3` is evaluated and becomes the result of the expression.

The Ternary Operator

This is equivalent to the following if-else statement:

```
if (expr1) {
    expr2;
} else {
    expr3;
}
```