Lecture 5.1

Topics

1. Side Effects – Introduction
2. Mixed-Type Expressions and Conversions – Introduction

---

1. Side Effects – Introduction

1.1 Definition

A side effect is an action or event that would result from evaluating an expression.

- Changing value of a variable is a side effect.
- Any changes to the state of the program are also called side effects, such as displaying data on screen.

Consider the following statement:

```
iX = 4;
```

- This statement has a simple expression with two primary expressions and an assignment operator. This simple expression will have value 4 with a side effect of variable iX receiving the value 4.
- The assignment expression/statement will have a value, which is the same as the value on the right of the equal sign.

The following program will display the results of several expressions and statements.

Example 1

```
/**
 *Program Name:  cis26L0511.c
 *Discussion:    Assignment & Side Effects
 */
#include <stdio.h>

int main() {
  int iVar;

  iVar = 3;

  printf("\nValue of iVar : %d", iVar);

  iVar = iVar + 4;

  printf("\nValue of iVar + 4 : %d", iVar + 4);
  printf("\nValue of iVar = iVar + 4 : %d", iVar = iVar + 4);
  printf("\nValue of iVar : %d", iVar);

  printf("\n");
  return 0;
}
```

OUTPUT

```
Value of iVar : 3
Value of iVar + 4 : 11
Value of iVar = iVar + 4 : 11
```

```
Value of iVar : 11
```

In the above example, the values of the primary and sub-expression(s) are displayed. They may reflect the intermediate or final values that would be produced while evaluating the statements.

1.2 Kinds of Side Effects

Besides displaying the results on screen, C has six (expressional) side effects:

- Four pre-effects, and
- Two post-effects.

The four pre-effects are

(i) The unary prefix increment (e.g., `++iX`),

(ii) The unary prefix decrement (e.g., `--iX`),

(iii) The function call, and

(iv) The assignment (e.g., `iX = 4, ix += 4`).

The side effects for these expressions take place BEFORE the expression is evaluated.

The two post-effect side effects are from the postfix operators:

(i) Postfix increment (e.g., `iX++`) and

(ii) Postfix decrement (e.g., `iX--`).

The side effects take place AFTER the expression has been evaluated; that means variable value is not changed until after it has been used in the expression.

Note! if a variable is being modified more than one then the expression value may not be evaluated with any certainty. In this case, rewrite the expression or come up with a different logic to be expressed in another version of the expression.

1.3 Expression Evaluation – Examples

The evaluation of an expression will depend on the kind of elements in the expression. Some elements may introduce no side effects during the course of evaluating the expression; and some may introduce side effects.

The process of evaluating an expression is described in the steps described previously. Let's simplify the process a bit just to show the important steps and the result, while some of the details are left out (to be discussed in class) but can be performed using the (above) given rules.

Example 2 Expression with no side effects

```
Let
     iX = 3    iY = 4    iZ = 5
Evaluate
     iX * 4 + iY / 2 – iZ * iY
     This is an expression that has no side effect.
Step 1
     Replacing all variables by their values
     3 * 4 + 4 / 2 – 5 * 4
Step 2
     Using precedence and evaluating proper operations
```

```
        12 + 2 - 20
    Step 3
        Simplifying and obtaining a final value
        -6
```

Example 3 Expression with side effects

```
    Let
        iX = 3   iY = 4   iZ = 5

    Evaluate
        --iX * ( 3 + iY ) / 2 – iZ++ * iY
        This is an expression with two side effects.

    Pre-Evaluating Substitution Rules
        (1) Copy any prefix increment or decrement expressions,
            and place them before the expression being evaluated.
            Replace each removed expression with its variable.
        (2) Copy any postfix increment or decrement expressions,
            and place them after the expression being evaluated.
            Replace each removed expression with its variable.

    Apply Substitution Rules & Evaluate Proper Operations
        With
            iX = 3   iY = 4   iZ = 5
        Evaluate
            --iX
        Produce
            iX = 2   iY = 4   iZ = 5
            iX * ( 3 + iY ) / 2 – iZ * iY
            iZ++

        Evaluate
            iX = 2   iY = 4   iZ = 5
            2 * ( 3 + 4 ) / 2 – 5 * 4
            iZ++

        Evaluate
            iX = 2   iY = 4   iZ = 5
            -13
            iZ++

        Final Results
            Expression Value : -13
            Variable Values : iX = 2    iY = 4   iZ = 6
```

Note that if a variable is being modified more than once in an expression then the expression may not be evaluated with any certainty. In this case, it is recommended to rewrite the expression or to come up with a different logic that can be expressed in another form (expression).

2. Mixed-Type Expressions and Conversions – Introduction

There are many expressions that have data of different types. In this case, the system will allow two kinds of guidance in converting the data to match up with the types:

(1) Implicit type conversion (automatic conversion), and

(2) Explicit type conversion.

Note that the main goal of data conversion is to preserve data and its precision if possible.

2.1 Implicit Type Conversion

C will automatically convert intermediate values to values with proper types so that the expression can be evaluated. This automatic conversion is also called an implicit conversion.

Except the assignments, all implicit conversions will follow the promotion hierarchy given below.

`char` → `short` → `int`

`int` → `unsigned int` → `long int` → `unsigned long int`

`unsigned long int` → `float` → `double` → `long double`

Using this hierarchy, the conversions will take place during the evaluating of the expressions. For examples,

`char + float`	→	`float`
`int – long`	→	`long`
`int * double`	→	`double`
`float / long double`	→	`long double`
`( short + long ) / float`	→	`long` then `float`

2.2 Explicit Type Conversion

The expression can be explicitly converted to any type by using an explicit conversion of the following form:

`( SomeType ) ( expression )`

For examples,

```
dAverage = (double) iSum / iNum;
dAverage = ((double) iSum) / iNum;
dAverage = iSum / (double) iNum;
```

What about the following statement?

```
dAverage = (double) (iSum / iNum);
```

Note that it is always better to cast the data explicitly. It makes things clear and shows the understanding of logic and programming operations being used.