ICS103 Programming in C

Lecture 4: Data Types, Operators & Expressions

Outline

- C Arithmetic Expressions
  - Operators
  - Data Type of Expression
  - Mixed-Type Assignment Statement
  - Type Conversion through Cast
  - Expressions with Multiple Operators
  - Writing Mathematical Formulas in C
- Programming Style

Why Arithmetic Expressions

- To solve most programming problems, you will need to write arithmetic expressions that manipulate type `int` and `double` data.
- The next slide shows all arithmetic operators. Each operator manipulates **two operands**, which may be constants, variables, or other arithmetic expressions.
- Example
  - 5 + 2
  - sum + (incr* 2)
  - (B/C) + (A + 0.5)

C Operators

Arithmetic Operator	Meaning	Examples
+ (*int,double*)	Addition	5 + 2 is 7 5.0 + 2.0 is 7.0
- (*int,double*)	Subtraction	5 - 2 is 3 5.0 - 2.0 is 3.0
* (*int,double*)	Multiplication	5 * 2 is 10 5.0 * 2.0 is 10.0
/ (*int,double*)	Division	5 / 2 is 2 5.0 / 2.0 is 2.5
% (*int*)	Remainder	5 % 2 is 1

Operator / & %

- **Division**: When applied to two positive integers, the division operator (/) computes the integral part of the result by dividing its first operand by its second.
  - For example 7.0 / 2.0 is 3.5 but the but 7 / 2 is only 3
  - The reason for this is that C makes the answer be of the same type as the operands.
- **Remainder**: The remainder operator (%) returns the integer remainder of the result of dividing its first operand by its second.
  - Examples: 7 % 2 = 1, 6 % 3 = 0
  - The value of m%n must always be less than the divisor n.
  - / is undefined when the divisor (second operator) is 0.

Data Type of an Expression

- The data type of each variable must be specified in its declaration, but how does C determine the data type of an expression?
    - Example: What is the type of expression `x+y` when both `x` and `y` are of type `int`?
- The data type of an expression depends on the type(s) of its operands.
    - If both are of type `int`, then the expression is of type `int`.
    - If either one or both is of type `double`, then the expression is of type `double`.
- An expressions that has operands of both `int` and `double` is a **mixed-type** expression.

Mixed-Type Assignment Statement

- The expression being evaluated and the variable to which it is assigned have different data types.
    - Example what is the type of the assignment `y = 5/2` when `y` is of type `double`?
- When an assignment statement is executed, the expression is first evaluated; then the result is assigned to the variable to the left side of assignment operator.
- **Warning**: assignment of a type `double` expression to a type `int` variable causes the fractional part of the expression to be lost.
    - What is the type of the assignment `y = 5.0 / 2.0` when `y` is of type `int`?

Type Conversion Through Casts

- C allows the programmer to convert the type of an expression.
- This is done by placing the desired type in parentheses before the expression.
- This operation called a **type cast**.
  - `(double)5 / (double)2` is the `double` value 2.5, and not 2 as seen earlier.
  - `(int)3.0 / (int)2.0` is the `int` value 1
- When casting from `double` to `int`, the decimal portion is just truncated – *not* rounded.

Example

```
/*   Computes a test average  */
#include <stdio.h>

int main(void)
{
        int    total_score, num_students;
        double average;
        printf("Enter sum of students' scores> ");
        scanf("%d", &total_score);
        printf("Enter number of students> ");
        scanf("%d", &num_students);
        average = (double) total_score / (double) num_students;
        printf("Average score is %.2f\n", average);
        return (0);
}
```

Expressions with Multiple Operators

- Operators can be split into two types: **unary** and **binary.**
- **Unary operators** take only one operand
  - - (negates the value it is applied to)
- **Binary operators** take two operands.
  - **+,-,*,/**
- A single expression could have multiple operators
  - -5 + 4 * 3 - 2

Rules for Evaluating Expressions

- **Rule (a): Parentheses rule** - All expressions in parentheses must be evaluated separately.
  - Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.
- **Rule (b): Operator precedence rule –** Multiple operators in the same expression are evaluated in the following order:
  - First: unary –
  - Second: *, /, %
  - Third: binary +,-
- **Rule (c): Associativity rule**
  - Unary operators in the same subexpression and at the same precedence level are evaluated right to left
  - Binary operators in the same subexpression and at the same precedence level are evaluated left to right.

Figure 2.8 Evaluation Tree for area = PI * radius * radius;

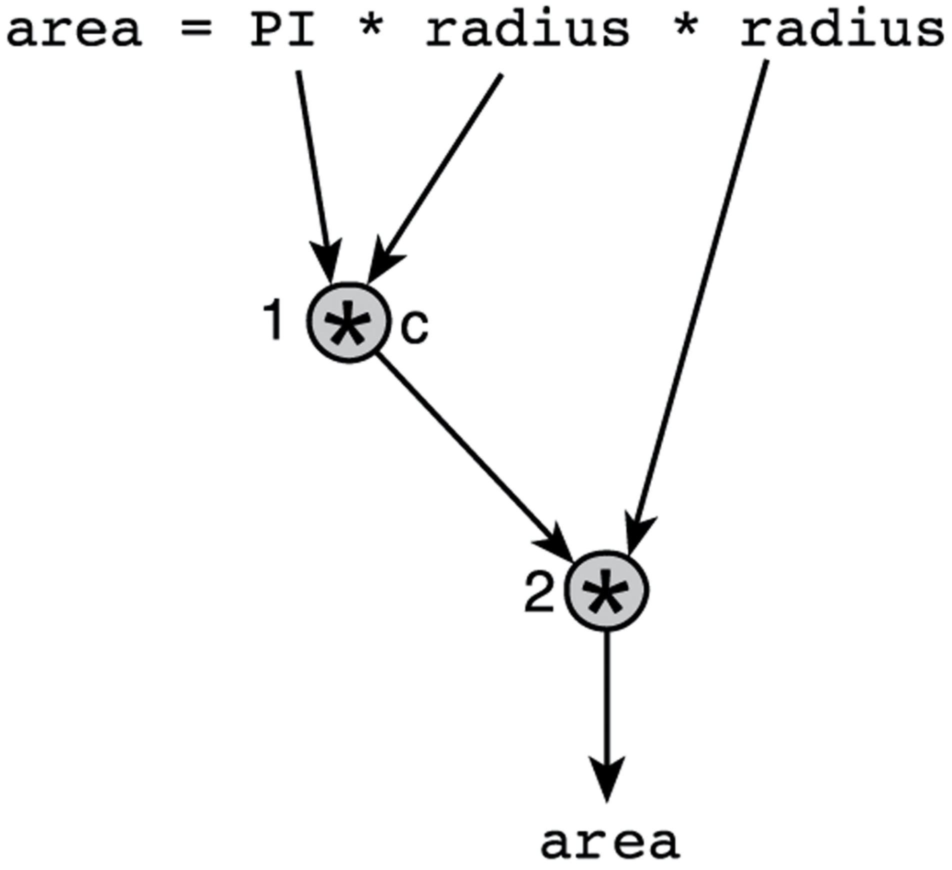
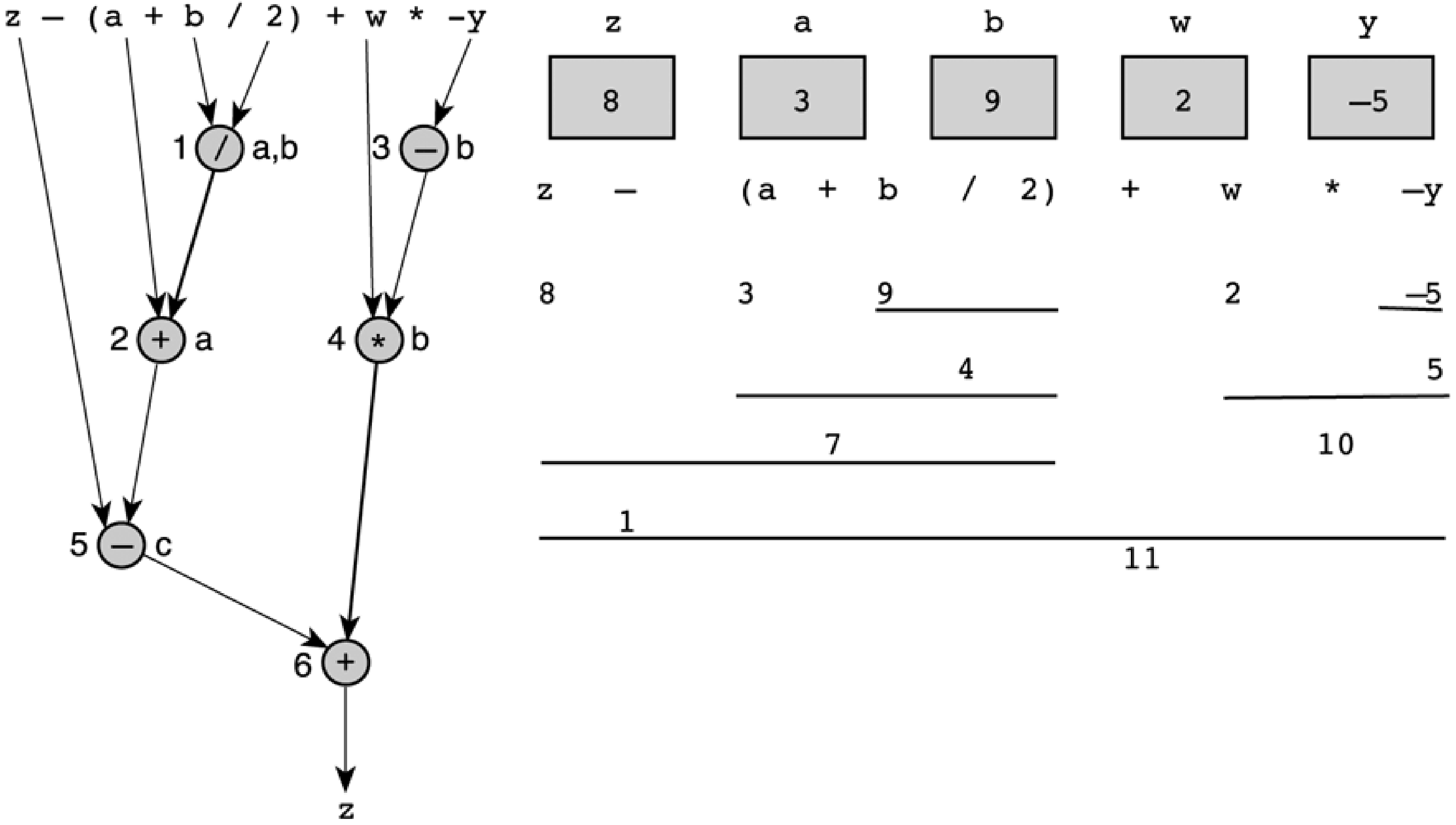


Figure 2.11 Evaluation Tree and Evaluation for z - (a + b / 2) + w * -y with type int variables only



Writing Mathematical Formulas in C

- You may encounter two problems in writing a mathematical formula in C.
- First, multiplication often can be implied in a formula by writing two letters to be multiplied next to each other. In C, you must state the * operator
  - For example, 2a should be written as 2 * a.
- Second, when dealing with division we often have:

$$\frac{a+b}{c+d}$$

  - This should be coded as (a + b) / (c + d).

Programming Style

- Why we need to follow conventions?
  - A program that ”looks good” is easier to read and understand than one that is sloppy.
  - 80% of the lifetime cost of a piece of software goes to maintenance.
  - Hardly any software is maintained for its whole life by the original author.
  - Programs that follow the typical conventions are more readable and allow engineers to understand the code more quickly and thoroughly.
- Check your text book and **some useful links** page for some directions.

White Spaces

- The complier ignores extra blanks between words and symbols, but you may insert space to improve the readability and style of a program.
- You should always leave a blank space after a comma and before and after operators such as , −, and =.
- You should indent the lines of code in the body of a function.

White Space Examples

Bad:

```
int main(void)
{ int foo,blah; scanf("%d",&foo);
blah=foo+1;
printf("%d", blah);
return 0;}
```

Good:

```
Int main(void)
{
        int foo, blah;
        scanf("%d", &foo);
        blah = foo + 1;
        printf("%d", blah);
        return 0;
}
```

Other Styles Concerns

- Properly comment your code
- Give variables meaningful names
- Prompt the user when you want to input data
- Display things in a way that looks good
    - Insert new lines to make your information more readable.
    - Format numbers in a way that makes sense for the application

Bad Programming practices

- Missing statement of purpose
- Inadequate commenting
- Variables names are not meaningful
- Use of unnamed constant.
- Indentation does not represent program structure
- Algorithm is inefficient or difficult to follow
- Program does not compile
- Program produces incorrect results.
- Insufficient testing (e.g. Test case results are different than expected, program branch never executed, borderline case not tested etc.)