CS 201
Introduction to C (2)

Debzani Deb

Overview

- C Arithmetic Expressions
- Formatting Numbers in Program Output
- Interactive Mode, Batch Mode, and Data Files
- Common Programming Errors
- Programming Style

Arithmetic Expressions

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

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)(5/2)` is the `double` value 2.5, and not 2 as seen earlier.
  - `(int)(3.0/2.0)` is the `int` value 1
- When casting from `double` to `int`, the decimal portion is just truncated – *not* rounded.

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;

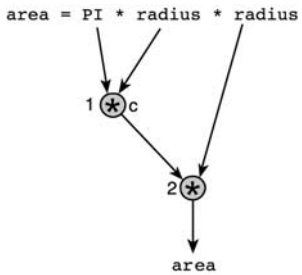
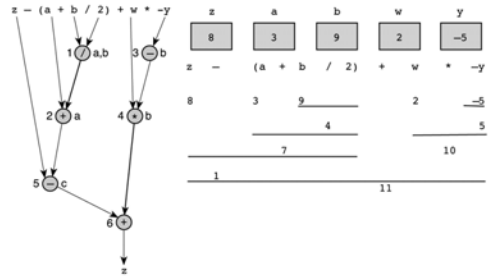


Figure 2.11 Evaluation Tree and Evaluation for z - (a + b / 2) + w * -y



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).

Formatting Numbers in Program Output (for integers)

- You can specify how `printf` will display numeric values
- Use d for integers. %#d
  - % - start of placeholder
  - # - field width (optional) – the number of columns to use to display the output.
  - d - placeholder for integers

```
int n = 123;
printf("%1d\n", n);      123
printf("%3d\n", n);      123
printf("%4d\n", n);       123
```


Formatting Numbers in Program Output (for double)

- Use %n.mf for double
  - % - start of placeholder
  - n - field width (optional)
  - m – Number of decimal places (optional)
  - f - placeholder for real numbers

```
double n = 123.456;
printf("%8.0f\n", n);          123
printf("%8.2f\n", n);       123.46
printf("%8.3f\n", n);      123.456
printf("%8.4f\n", n);     123.4560
Printf("%.2f\n", n);      123.45
```


Interactive Mode, Batch Mode, and Data Files

- Input Redirection
- Output Redirection
- Program-Controlled Input and Output Files

Definitions

- **Interactive mode** - the user interacts with the program by entering (typing in) data while the program is running.
- **Batch mode** - the program scans its data from a data file prepared beforehand instead of interacting with its user.

Input Redirection

- In the next frame we will see the miles-to-kilometers conversion program rewritten as a batch program.
- We assume here that the standard input device is associated with a batch data file instead of with the keyboard.
- In most system, this association can be accomplished relatively easily through input/output redirection using operating system commands.
- Instead of calling the program as:

```
$ conversion
```

  We would call it as:

```
$ conversion < myinput
```

- This **redirects** the text in the file `myinput` and uses it as the program input.
- Here $ represents command prompt.

Miles to Kilometers conversion program in interactive mode

```
/* Converts distances from miles to kilometers */
#include <stdio.h>                  /* printf, scanf definitions */
#define KMS_PER_MILE 1.609          /* conversion constant */
int main(void)
{
    double miles,     //distance in miles
           kms;       //equivalent distance in kilometers

    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```


Miles to Kilometers conversion program with input redirection.

```
/* Converts distances from miles to kilometers */
#include <stdio.h>                  /* printf, scanf definitions */
#define KMS_PER_MILE 1.609          /* conversion constant */
int main(void)
{
        double miles,     //distance in miles
               kms;       //equivalent distance in kilometers

        //Get and echo the distance in miles
        scanf("%lf", &miles);
        printf("The distance in miles is %.2f.\n", miles);

        //Convert the distance to kilometers
        kms = KMS_PER_MILE * miles;

        //Display the distance in kilometers
        printf("That equals %f kilometers.\n", kms);

        return (0);
}
```


Echo Prints vs. Prompts

- In the above program `scanf` gets a value for miles from the first (and only) line of the data file.
- Because the program input comes from a data file, there is no need to precede this statement with a prompting message.
- Instead, we follow the call to `scanf` with the statement

```
printf("The distance in miles is %.2f.\n",miles);
```

- This statement **echo prints** or displays the value just stored in `miles`.
- Without it, we would have no easy way of knowing what value `scanf` obtained for miles.
- Whenever you convert an interactive program to a batch program, make sure you replace each prompt with an echo print after the `scanf`.

Output Redirection

- You can also redirect the output of the program to a file instead of the screen.
- Then you can send the output file to the printer to obtain a hard copy of the program output.
- The command line:

```
$ conversion > myoutput
```

  sends the output of the program `conversion` to the file `myoutput.`
- You can do both input and output redirection by using:

```
$ conversion < myinput > myoutput
```


Program Controlled Input and Output Files

- As an alternative to input/output redirection, C allows a program to read/write from/to files within the program.
- To do this, you need to:
  1. Include `stdio.h`
  2. Declare a variable of type `FILE`
  3. Open the file for reading/writing.
  4. Read/write from/to the file.
  5. Close the file.
- In the example (next slide) you will see each of these steps.

Miles to Kilometers conversion using program controlled input/output

```
#include <stdio.h>
#define KMS_PER_MILE 1.609

int main(void) {
        double kms, miles;
        FILE *inp, *outp;

        inp = fopen("myinput","r");
        outp = fopen("myoutput","w");
        fscanf(inp, "%lf", &miles);
        fprintf(outp, "The distance in miles is %.2f.\n", miles);

        kms = KMS_PER_MILES * miles;

        fprintf(outp, "That equals %.2f kilometers.\n", kms);
        fclose(inp);
        fclose(outp);
        return (0);
}
```


Common Programming Errors

- **Syntax Errors -** this occurs when your code violates one or more grammar rules of C.
- **Run-Time Errors -** these are detected and displayed by the computer during the execution of a program.
- **Undetected Errors -** many execution errors may not prevent a C program from running to completion, but they may simply lead to incorrect results.
- **Logic Errors -** these occur when a program follows a faulty algorithm.
- **Debugging -** Finding bugs/errors in the program.

Syntax Errors

- A syntax error occurs when your code violates one or more grammar rules of C
  - This is detected by the compiler as it attempts to translate your program.
  - If a statement has a syntax error, it cannot be translated and your program will not be executed.
- Common syntax errors:
  - Missing semicolon
  - Undeclared variable
  - Last comment is not closed

Run-Time Errors

- Run-time errors are detected and displayed by the computer during the execution of a program.
- A run-time error occurs when the program directs the computer to perform an illegal operation, such as dividing a number by zero.
- When a run-time errors occurs, the computer will stop executing your program and will display a diagnostic message
  - This message may indicate the line where the error was detected.

Undetected Errors

- Many execution errors may not prevent a C program from running to completion, but they may simply lead to incorrect results.
- It is essential that you predict the results your program should produce and verify that the actual output is correct.
- A very common source of incorrect results in C programs is the input of a mixture of character and numeric data.
  - These errors can be avoided if the programmer always keeps in mind the scanf's different treatment of %c and and %d/%lf placeholders.
- These may also occur if you make a mistake about the evaluation order of an arithmetic expression with multiple operators.

Logic Errors

- Logic errors occur when a program follows a faulty algorithm.
- Because logic errors usually do not cause run-time errors and do not display error messages, they are difficult to detect.
- The only sign of a logic error may be incorrect program output.
- You can detect logic errors by testing the program thoroughly, comparing its output to calculated results.

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.
  - Program 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 sensible 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.)