

UNIVERSITY OF  
**Nebraska**  
Lincoln

CSCE150A

Introduction

Language Elements

General Form

Operators & Expressions

Formatting I/O

Running Programs

Common Pitfalls

# Computer Science & Engineering 150A


## Problem Solving Using Computers

### Lecture 02 - Introduction To C

Stephen Scott

(Adapted from Christopher M. Bourke)

Fall 2009



## Notes

[illegible]

University of  
**Nebraska**  
Lincoln

CSCE150A

Introduction

Language Elements

General Form

Operators & Expressions

Formatting I/O

Running Programs

Common Pitfalls

- C Language Elements
- Variable Declarations and Data Types
- Executable Statements
- General Form of a C Program
- Arithmetic Expressions
- Formatting Numbers in Program Output
- Interactive Mode, Batch Mode, and Data Files
- Common Programming Errors

2 / 77

## Notes

---

---

---

---

---

---

UNIVERSITY OF  
**Nebraska**  
LINCOLN

CSCE150A

Introduction

Language Elements

General Form

Operators & Expressions

Formatting I/O

Running Programs


Common Pitfalls

# Overview of C Programming

This chapter introduces C – a high-level programming language developed in 1972 by Dennis Ritchie at AT&T Bell Laboratories.

This chapter describes the elements of a C program and the types of data that can be processed by C. It also describes C statements for performing computations, for entering data, and for displaying results.

3 / 77



## Notes

---

---

---

---

---

---

## C Language Elements

CSCE150A

Introduction

Language  
Elements

Preprocessor

Comments

Main Function

Reserved Words

Program Style

Executable

Statements

Input/Output

General Form

Operators &amp;

Expressions

Formatting

I/O

Running

Programs

Common

pitfalls

- Preprocessor Directives
- Syntax Displays for Preprocessor Directives
- "int main()" Function
- Reserved Words
- Standard Identifiers
- User-Defined Identifiers
- Uppercase and Lowercase Letters
- Program Style

## Notes

---

---

---

---

---

---

---

## Preprocessor Directives

CSCE150A

Introduction

Language  
Elements

Preprocessor

Comments

Main Function

Reserved Words

Program Style

Executable

Statements

Input/Output

General Form

Operators &amp;

Expressions

Formatting

I/O

Running

Programs

Common

pitfalls

- The *C preprocessor* modifies the text of the C program before it is passed to the compiler.
- Preprocessor directives are C program lines beginning with a **#** that provide instructions to the C preprocessor.
- Preprocessor directives begins with a **#**, e.g. **#include** and **#define**.
- Predefined libraries are useful functions and symbols that are predefined by the C language (standard libraries).

## Notes

---

---

---

---

---

---

---

## #include and #define

CSCE150A

Introduction

Language  
Elements

Preprocessor

Comments

Main Function

Reserved Words

Program Style

Executable

Statements

Input/Output

General Form

Operators &amp;

Expressions

Formatting

I/O

Running

Programs

Common

pitfalls

- **#include**<libraryName> gives the program access to a library
- Example: **#include**<stdio.h> (standard input and output) has definitions for input and output, such as **printf** and **scanf**.
- **#define** NAME value associates a constant *macro*
- Example:

```
1 #define KMS_PER_MILE 1.609
2 #define PI 3.14159
```

## Notes

---

---

---

---

---

---

---

UNIVERSITY OF  
**Nebraska**  
LINCOLN

# Comments

CSCE150A

Introduction

Language

Elements

Preprocessor

Comments

Main Function

Reserved Words

Program Style

Executable

Statements

Input/Output

General Form

Operators &

Expressions

Formatting

I/O

Running

Programs

Common

pitfalls

Comments provide supplementary information making it easier for us to understand the program, but comments are ignored by the C preprocessor and compiler.

- `/* */` - anything between them will be considered a comment, even if they span multiple lines.
- `//` - anything after this and before the end of the line is considered a comment.

## Notes

[illegible]

University of  
Nebraska  
Lincoln

# Function main

CSC150A

Introduction

Language

Elements

Preprocessor

Comments

Main Function

Reserved Words

Program Style

Executable

Statements

Input/Output

General Form

Operators &

Expressions

Formatting

I/O

Running

Programs

Common

pitfalls

- The point at which a C program begins execution is the **main** function:


```
1 int main(void)
```

- Every** C program must have a main function.
- The main function (and every other function) body has two parts:
  - Declarations - tell the compiler what memory cells are needed in the function
  - Executable statements - (derived from the algorithm) are translated into machine language and later executed

Navigation icons: back, forward, search, etc.

## Notes

[illegible]



# Function main

CSCE150A

Introduction

Language

Elements

Preprocessor

Comments

Library Functions

Reserved Words

Program Style

Executable

Statements

Input/Output

General Form

Operators &

Expressions

Formatting

I/O


Running

Programs

Common

Patterns

- All C functions contain *punctuation* and *special symbols*
  - Punctuation - commas separate items in a list, semicolons appear at the end of each statement
  - Special symbols: \*, =, {, }, etc.
  - Curly braces mark the beginning and end of the body of every function, including main



## Notes

[illegible]

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
Preprocessor  
Comments  
Main Function  
Reserved Words  
Program Code  
Executable  
Statements  
Input/Output  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Patterns??

## Reserved Words

- A word that has special meaning in C. E.g.:
  - `int` - Indicates that the main function (or any other function) returns an integer value, or that a memory cell will store an integer value
  - `double` - Indicates that a function returns a real number or that a memory cell will store a real number
- Always lower case
- Can not be used for other purposes
- Appendix E has a full listing of reserved words (ex: `double`, `int`, `if`, `else`, `void`, `return` etc.)

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
Preprocessor  
Comments  
Main Function  
Reserved Words  
Program Code  
Executable  
Statements  
Input/Output  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Patterns??

## Standard Identifiers

- Standard identifiers have a special meaning in C (assigned by standard libraries).
- Standard identifiers can be redefined and used by the programmer for other purposes
  - Not recommended If you redefine a standard identifier; C will no longer be able to use it for its original purpose.
- Examples: input/output functions `printf`, `scanf`

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
Preprocessor  
Comments  
Main Function  
Reserved Words  
Program Code  
Executable  
Statements  
Input/Output  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Patterns??

## User-Defined Identifiers

We choose our own identifiers to name memory cells that will hold data and program results and to name operations (*functions*) that we define (more on this in Chapter 3)

- An identifier must consist only of letters [`a-zA-Z`], digits [`0-9`], and underscores.
- An identifier cannot begin with a digit (and *shouldn't* begin with an underscore).
- A C reserved word cannot be used as an identifier.
- An identifier defined in a C standard library should not be redefined.

Notes

---

---


---

---

---

---

---



# Style Tips

## Rigorous Comments

**CSCE150A**

- Introduction
- Language Elements
  - Preprocessor
  - Comments
  - Main Function
  - Reserved Words
- Program Style**
  - Lexical Statements
  - Input/Output
- General Form
- Operators & Expressions
- Formatting I/O
- Running Programs
- Common pitfalls

- The number of comments in your program doesn't affect its speed or size.
- Always best to include as much documentation as possible in the form of comments.
- Begin each program or function with a full explanation of its inputs, outputs, and how it works.
- Include comments as necessary throughout the program

[illegible]

University of  
Nebraska  
Lincoln

# Style Tips

## Naming Conventions

CSCE150A

Introduction

Language  
Elements

Preprocessor  
Comments  
Main Function  
Reserved Words

Basic Style  
Executable  
Statements  
Input/Output

General Form

Operators &  
Expressions

Formatting  
I/O


Running  
Programs

Common  
Pitfalls

- Give your variables *meaningful* names (identifiers)
- `x`, `y` may be good if you're dealing with coordinates, but bad in general.
- `myVariable`, `aVariable`, `anInteger`, etc are *bad*: they do not describe the purpose of the variable.
- `tempInt`, `PI`, `numberOfStudents` are good because they do.

16 / 17

[illegible]



# Style Tips

## CamelCaseNotation

CSCE150A

Introduction

Language Elements

Preprocessor

Comments

Main Function

Reserved Words

Program Style

Executable Statements

Input/Output

General Form

Operators & Expressions

Formatting


I/O

Running Programs

Common Pitfalls

Appendix

- Old School C convention: separate compound words with underscores
- `number_of_students`, `interest_rate`, `max_value`, etc.
- Underscore (shift-hyphen) is inconvenient
- Solution: camelCaseNotation - connect compound words with upper-case letters.
- Example: `numberOfStudents`, `interestRate`, `maxValue`, etc.
- Much easier to shift-capitalize
- Much more readable
- Ubiquitous outside of programming: MasterCard, PetsMart, etc.



**Notes**

---

---


---

---

---

---

---



# Anatomy of a Program

CSCE150A

Introduction

Language

Elements

Preprocessor

Comments

Main Function

Reserved Words

Program Style

Executable

Statements

Input/Output

General Form

Operators &

Expressions

Formatting

I/O

Running

Programs


Common

Patterns

```

1  /*
2   * Converts distances from miles to kilometers.
3   */
4  #include <stdio.h> /* printf, scanf definitions */
5  #define KMS_PER_MILE 1.609
6
7  int main(void)
8  {
9      double miles, kilometers;
10     printf("How many miles do you have?");
11
12     scanf("%lf",&miles);
13
14     kilometers = miles * 1.609;
15     printf("You have %f kilometers\n", kilometers);
16
17     return 0;
18 }
```

Comments



[illegible]

Nebraska  
Lincoln

CSCE150A

Introduction

Language Elements

Preprocessor Comments

Main Function

Reserved Words

Program Code

Executable Statements

Input/Output

General Form

Operators & Expressions

Formatting I/O

Running Programs

Common Paths

Anatomy of a Program

```
1  /*
2  *  Converts distances from miles to kilometers.
3  */
4  #include <stdio.h> /* printf, scanf definitions */
5  #define KMS_PER_MILE 1.609
6
7  int main(void)
8  {
9      double miles, kilometers;
10     printf("How many miles do you have?");
11     scanf("%lf",&miles);
12
13     kilometers = miles * 1.609;
14     printf("You have %lf kilometers\n", kilometers);
15
16     return 0;
17 }
18
```

Preprocessor Directives

Notes

Nebraska  
Lincoln

CSCE150A

Introduction

Language Elements

Preprocessor Comments

Main Function

Reserved Words

Program Code

Executable Statements

Input/Output

General Form

Operators & Expressions

Formatting I/O

Running Programs

Common Paths

Anatomy of a Program

```
1  /*
2  *  Converts distances from miles to kilometers.
3  */
4  #include <stdio.h> /* printf, scanf definitions */
5  #define KMS_PER_MILE 1.609
6
7  int main(void)
8  {
9      double miles, kilometers;
10     printf("How many miles do you have?");
11     scanf("%lf",&miles);
12
13     kilometers = miles * 1.609;
14     printf("You have %lf kilometers\n", kilometers);
15
16     return 0;
17 }
18
```

Reserved Words

Notes

Nebraska  
Lincoln

CSCE150A

Introduction

Language Elements

Preprocessor Comments

Main Function

Reserved Words

Program Code

Executable Statements

Input/Output

General Form

Operators & Expressions

Formatting I/O

Running Programs

Common Paths

Anatomy of a Program

```
1  /*
2  *  Converts distances from miles to kilometers.
3  */
4  #include <stdio.h> /* printf, scanf definitions */
5  #define KMS_PER_MILE 1.609
6
7  int main(void)
8  {
9      double miles, kilometers;
10     printf("How many miles do you have?");
11     scanf("%lf",&miles);
12
13     kilometers = miles * 1.609;
14     printf("You have %lf kilometers\n", kilometers);
15
16     return 0;
17 }
18
```

Variables

Notes

## Anatomy of a Program

CSCE150A

Introduction  
Language  
Elements  
Preprocessor  
Comments  
Main Function  
Reserved Words  
Program Style  
Executable  
Statements  
Input/Output  
General Form  
Operators & Expressions  
Formatting  
I/O  
Running Programs  
Common Paths

```
1  /*
2  *  Converts distances from miles to kilometers.
3  */
4  #include <stdio.h> /* printf, scanf definitions */
5  #define KMS_PER_MILE 1.609
6
7  int main(void)
8  {
9      double miles, kilometers;
10     printf("How many miles do you have?");
11     scanf("%lf", &miles);
12
13     kilometers = miles * 1.609;
14     printf("You have %f kilometers\n", kilometers);
15
16     return 0;
17 }
18
```

Special Symbols

◀ ▶ 🔍 ↺ ↻ ⌂

Notes

## Anatomy of a Program

CSCE150A

Introduction  
Language  
Elements  
Preprocessor  
Comments  
Main Function  
Reserved Words  
Program Style  
Executable  
Statements  
Input/Output  
General Form  
Operators & Expressions  
Formatting  
I/O  
Running Programs  
Common Paths

```
1  /*
2  *  Converts distances from miles to kilometers.
3  */
4  #include <stdio.h> /* printf, scanf definitions */
5  #define KMS_PER_MILE 1.609
6
7  int main(void)
8  {
9      double miles, kilometers;
10     printf("How many miles do you have?");
11     scanf("%lf", &miles);
12
13     kilometers = miles * 1.609;
14     printf("You have %f kilometers\n", kilometers);
15
16     return 0;
17 }
18
```

Punctuation

◀ ▶ 🔍 ↺ ↻ ⌂

Notes

## Anatomy of a Program

CSCE150A

Introduction  
Language  
Elements  
Preprocessor  
Comments  
Main Function  
Reserved Words  
Program Style  
Executable  
Statements  
Input/Output  
General Form  
Operators & Expressions  
Formatting  
I/O  
Running Programs  
Common Paths

```
/*  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;
}
```

◀ ▶ 🔍 ↺ ↻ ⌂

Notes



University of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
Preprocessor Comments  
Main Function  
Reserved Words  
Program Style  
Executable Statements  
Input/Output  
General Form  
Operators & Expressions  
Formatting  
I/O  
Running Programs  
Common Pitfalls

## Variable Declarations and Data Types

- Variable Declaration
- Data Types

Notes

---

---

---

---

---

---

---

University of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
Preprocessor Comments  
Main Function  
Reserved Words  
Program Style  
Executable Statements  
Input/Output  
General Form  
Operators & Expressions  
Formatting  
I/O  
Running Programs  
Common Pitfalls

## Variables Declarations

- Variables - a name associated with memory cells (`miles`, `kilometers`) that store a program's input data. The value of this memory cell can be changed.
- Variable declarations - statements that communicate to the compiler that names of variables in the program and the kind of information stored in each variable.
  - Example: `double miles, kms;`
  - Each declaration begins with a *unique* identifier to indicate the type of data
  - Every variable used *must* be declared before it can be used

Notes

---

---

---

---

---

---

---

University of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
Preprocessor Comments  
Main Function  
Reserved Words  
Program Style  
Executable Statements  
Input/Output  
General Form  
Operators & Expressions  
Formatting  
I/O  
Running Programs  
Common Pitfalls

## Data Types

- Data Types: a set of values and a set of operations that can be used on those values.
- In other words, it is a classification of a particular type of information.
  - Integers `int`
  - Doubles `double`
  - Characters `char`
- The idea is to give semantic meaning to 0s and 1s.

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
Preprocessor  
Comments  
Main Function  
Reserved Words  
Program Style  
Executable  
Statements  
Input/Output  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls??

Data Types  
Integers

- Integers are whole numbers, negative and positive
- Declaration: `int`
- The ANSI C standard requires integers be *at least* 16 bits: in the range  $-32767$  to  $32767$
- One bit for the *sign* and 15 for the number
- Modern standard that `int` types are 32 bits. Range:  $-2^{31} = -2,147,483,648$  to  $2,147,483,648 = 2^{31}$
- Newer systems are 64-bit. What range does this give?

## Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
Preprocessor  
Comments  
Main Function  
Reserved Words  
Program Style  
Executable  
Statements  
Input/Output  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls??

Data Types  
Doubles

- Doubles are decimal numbers, negative and positive
- Example: 0.5, 3.14159265, 5, 8.33
- Declaration: `double`
- On most systems, doubles are 8 bytes = 64 bits
- Precision is *finite*: cannot *fully* represent irrationals  $\pi$ ,  $\frac{1}{3}$ , etc.
- An approximation only, but 15–16 digits of precision

## Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
Preprocessor  
Comments  
Main Function  
Reserved Words  
Program Style  
Executable  
Statements  
Input/Output  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls??

Data Types  
Characters

- `char`: an individual character values with single quotes around it.
- Example: a letter, a digit, or a special symbol
- Example: `'a'`, `'B'`, `'*'`, `'!'`
- You can treat each character as a number: see Appendix A
- The ASCII standard assigns number (0 thru 255) to each character: A is 65, many are non-printable control characters

## Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
Preprocessor Comments  
Main Function  
Reserved Words  
Program Style  
Executable Statements  
Input/Output  
General Form  
Operators & Expressions  
Formatting  
I/O  
Running Programs  
Common pitfalls??

## Executable Statements

- Assignment Statements
- Input/Output Operations and Functions
- `printf` Function
- `scanf` Function
- `return` Statement

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
Preprocessor Comments  
Main Function  
Reserved Words  
Program Style  
Executable Statements  
Input/Output  
General Form  
Operators & Expressions  
Formatting  
I/O  
Running Programs  
Common pitfalls??

## Assignment Statements

- Assignment statements - stores a value or a computational result in a variable
- Used to perform most arithmetic operations in a program.
- Form: `variable = expression;`
  - `kms = KMS_PER_MILE * miles;`

The assignment statement above assigns a value to the variable `kms`. The value assigned is the result of the multiplication of the constant macro `KMS_PER_MILE` (1.609) by the variable `miles`

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
Preprocessor Comments  
Main Function  
Reserved Words  
Program Style  
Executable Statements  
Input/Output  
General Form  
Operators & Expressions  
Formatting  
I/O  
Running Programs  
Common pitfalls??

## Memory of Program

The diagram illustrates the memory layout of a program. It shows three rows of memory cells. The first row contains a variable named 'miles' of type 'double' with a question mark, and another variable named 'miles' containing the value '10.00'. The second row contains a variable named 'kms' of type 'double' with a question mark, and another variable named 'kms' containing the value '16.09'. The third row contains a constant named 'KMS\_PER\_MILE' with the value '1.609', and another constant named 'KMS\_PER\_MILE' containing the value '1.609'. Arrows point from the labels 'miles' and 'kms' to their respective memory cells. The label 'type double' is placed to the left of the first two rows. The label 'constant' is placed to the left of the third row.

Figure: Memory

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
Preprocessor  
Comments  
Main Function  
Reserved Words  
Program Style  
Executable  
Statements  
Input/Output  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Patterns??

## Assignments Continued

- In C, the symbol = is the assignment operator.
- Read as “becomes”, “gets”, or “takes the value of” rather than “equals”
- In C, == tests equality.
- Examples:

```
1 int a, b, c;  
2 b = 10;  
3 a = 15;  
4 c = a + b;
```

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
Preprocessor  
Comments  
Main Function  
Reserved Words  
Program Style  
Executable  
Statements  
Input/Output  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Patterns??

## Assignments

### Misconception

- In C you can write:  
    `sum = sum + item;` or  
    `a = a + 1;`
- These are *not* algebraic expressions
- This does not imply that  $0 = 1$
- Meaning: `a` is to be given the value that `a` had before plus one
- Common programming practice
- Instructs the computer to add the current value of `sum` to the value of `item` then store the result into the variable `sum`.

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
Preprocessor  
Comments  
Main Function  
Reserved Words  
Program Style  
Executable  
Statements  
Input/Output  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Patterns??

## Input/Output Operations and Functions

- Input operation - data transfer from the outside world into memory.
- Output operation - An instruction that displays program results to the program user or sends results to a file or device
- input/output functions - special program units that do all input/output operations. Common I/O functions found in the *Standard Input/Output Library*: `stdio.h`
- Function call - in C, a function call is used to call or activate a function.
- Analogous to ordering food from a restaurant. You (the calling routine) do not know all of the ingredients and procedures for the food, but the called routine (the restaurant) provides all of this for you.

Notes

---

---

---

---

---

---

---

## Output Function: printf Function

CSCE150A

Introduction  
Language Elements  
Preprocessor Comments  
Main Function  
Reserved Words  
Program Style  
Executable Statements  
Input/Output  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common pitfalls??

Included in the `stdio.h` library.

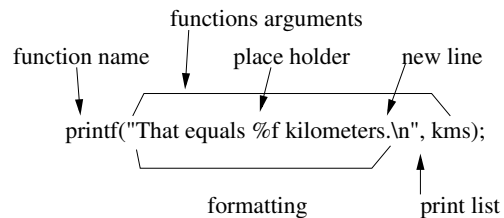


Figure: Parts of the printf function

Navigation icons: back, forward, search, etc.

## Notes

---

---

---

---

---

---

---

## Place Holder

CSCE150A

Introduction  
Language Elements  
Preprocessor Comments  
Main Function  
Reserved Words  
Program Style  
Executable Statements  
Input/Output  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common pitfalls??

A placeholder always begins with the symbol `%`. Note the newline *escape sequence* `\n`. Format strings can have multiple placeholders.

Placeholder	Variable Type	Function Use
<code>%c</code>	char	<code>printf</code> , <code>scanf</code>
<code>%d</code>	int	<code>printf</code> , <code>scanf</code>
<code>%f</code>	double	<code>printf</code> only
<code>%lf</code>	double	<code>scanf</code> only

Navigation icons: back, forward, search, etc.

## Notes

---

---

---

---

---

---

---

## Displaying Prompts

CSCE150A

Introduction  
Language Elements  
Preprocessor Comments  
Main Function  
Reserved Words  
Program Style  
Executable Statements  
Input/Output  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common pitfalls??

When input data is needed in an interactive program, you should use the `printf` function to display a **prompting message**, or **prompt**, that tells the program user what data to enter.

```
printf("Do you have any questions? ");
```

or

```
printf("Enter the number of items> ");
```

Navigation icons: back, forward, search, etc.

## Notes

---

---

---

---

---

---

---

## Input Function: scanf

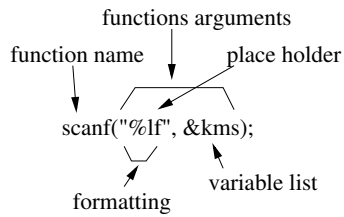


Figure: Parts of the scanf function

- In general, do not put extra characters in the format string
- Be sure to use the *ampersand*, & with `scanf`!!!

## Notes

## Return Statement

```
return 0;
```

This statement returns a 0 to the operating system to signify that the program ended in a correct position. It does not mean the program did what it was supposed to do. It only means there were no runtime errors. There still may have been logical errors.

## Notes

## Program Example

```
1 #include <stdio.h>
2
3 int main(void) {
4     char first, last;
5
6     printf("Enter your first and last initials");
7     scanf("%c %c", &first, &last);
8
9     printf("Hello %c. %c. How are you?\n", first, last);
10
11     return 0;
12 }
```

## Notes

[illegible][illegible][illegible]

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

41 / 77

Comments in Programs

Use comments to do **Program Documentation**, to help others read and understand the program.

- The start of the program should consist of a comment that includes the programmer's name, date of current version, and brief description of what the program does.
- Include comments for each variable and each major step in the program.
- For any function, make comments to briefly describe the input to the function, the output of the function, and the use of the function.
- *Comments cannot be nested!*

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

42 / 77

Comments in Programs

Style:

```
1  /*  
2  * Multiple line comments are good  
3  * for describing functions.  
4  */  
5  
6  /* This /* is NOT */ ok. */  
7  
8  /* // ok. */
```

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

43 / 77

Arithmetic Expressions

- Operators / and % (Read *mod* or *remainder*)
- Data Type of Expression
- Mixed-Type Assignment Statement
- Type Conversion through Cast
- Expressions with Multiple Operators
- Writing Mathematical Formulas in C

Notes

---

---

---

---

---

---

---



## Notes

Nebraska

Lincoln

CSCE150A

Introduction

Language Elements

General Form

Operators & Expressions

Formatting I/O

Running Programs

Common Pitfalls

Remainder Operator

The remainder operator (%) returns the integer remainder of the result of dividing its first operand by its second.

- Similar to integer division, except instead of outputting integral portion, outputs remainder.
- The operand % can give different answers when the second operand is negative.
- As with division, % is undefined when the second operand is 0.

47 / 77

Notes

---

---

---

---

---

---

---

Nebraska

Lincoln

CSCE150A

Introduction

Language Elements

General Form

Operators & Expressions

Formatting I/O

Running Programs

Common Pitfalls

Remainder Operator

Exercise

Problem

What are the results of the following operations?

- 51 % 2
- 100 % 4
- 101 % 31

48 / 77

Notes

---

---

---

---

---

---

---

Nebraska

Lincoln

CSCE150A

Introduction

Language Elements

General Form

Operators & Expressions

Formatting I/O

Running Programs

Common Pitfalls

Remainder Operator

Exercise

Problem

What are the results of the following operations?

- 51 % 2
- 100 % 4
- 101 % 31

- 51 % 2 → 1

48 / 77

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

Remainder Operator  
Exercise

Problem

What are the results of the following operations?

❶  $51 \% 2$

❷  $100 \% 4$

❸  $101 \% 31$

❶  $51 \% 2 \rightarrow 1$

❷  $100 \% 4 \rightarrow 0$

48 / 77

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

Remainder Operator  
Exercise

Problem

What are the results of the following operations?

❶  $51 \% 2$

❷  $100 \% 4$

❸  $101 \% 31$

❶  $51 \% 2 \rightarrow 1$

❷  $100 \% 4 \rightarrow 0$

❸  $101 \% 31 \rightarrow 8$

48 / 77

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

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?

- The data type of an expression depends on the type(s) of its operand(s). If both are of type `int`, then result is of type `int`. If either one or both is of type `double`, then result is of type `double`.
- An expression that has operands of both `int` and `double` is a mixed-type expression, and will be typed as `double`.

For a mixed-type assignment, be aware that the expression is evaluated first, and then the *result* is converted to the correct type.  
E.g. if `y` is `double`, then `y=5/2` gets 2.0, not 2.5

49 / 77

Notes

---

---

---

---

---

---

---

University of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common Pitfalls

50 / 77

Type Conversion through Type Casting

C is flexible enough to allow the programmer to convert the type of an expression by placing the desired type in parentheses before the expression, an operation called a *type cast*.

Example: `(double)5 / 2` results in 2.5, not 2 as seen previously.

Notes

---

---

---

---

---

---

---

University of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common Pitfalls

51 / 77

Expressions with Multiple Operators

- In expressions, we often have multiple operators
  - Equations may not evaluate as we wish them to:
    - Is  $x/y * z$  the same as  $(x/y) * z$  or  $x/(y * z)$ ?
- Unary operators** take only one operand: -5, +3, -3.1415, etc.
- Binary operators** take two operands:  
3 + 4, 7 / 5, 2 \* 6, 4 % 3, etc.

Notes

---

---

---

---

---

---

---

University of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common Pitfalls

52 / 77

Rules for Evaluating Expressions

- 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.
- Operator precedence rule:** Operators in the same expression are evaluated in a fixed order (see Table 1 in your book)
  - Unary Operators    +, -
  - Binary Operators    \*, /, %
  - Binary Arithmetic    +, -
- 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.

Notes

---

---

---

---

---

---

---

University of  
Nebraska  
Lincoln

Exercise

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

Problem


*What are the results of the following expressions:*

- 1  $x = -15 * 4 / 2 * 3;$
- 2  $y = 5 + 2 * 4;$
- 3  $z = 4 - 5 * 2 - 4 + -10;$

53 / 77

## Notes

[illegible]



# Exercise

CSCE150A

Introduction

Language  
Elements

General Form

Operators &  
Expressions

Formatting  
I/O

Running  
Programs

Common  
Pitfalls

## Problem

*What are the results of the following expressions:*


$$① \quad x = -15 * 4 / 2 * 3;$$

$$② \quad y = 5 + 2 * 4;$$

$$③ \quad z = 4 - 5 * 2 - 4 + -10;$$

$$④ \quad x = -90 \text{ (why not } -10?)$$

53 / 77



## Notes

[illegible]

UNIVERSITY OF  
Nebraska  
LINCOLN

Exercise

CSCE150A

Introduction

Language

Elements

General Form

Operators &  
Expressions

Formatting  
I/O

Running  
Programs

Common  
Pitfalls

Problem

*What are the results of the following expressions:*

❶  $x = -15 * 4 / 2 * 3;$

❷  $y = 5 + 2 * 4;$

❸  $z = 4 - 5 * 2 - 4 + -10;$

❹  $x = -90$  (why not  $-10$ ?)

❺  $y = 13$

53 / 77

## Notes

---

---

---

---

---

---

Nebraska

Lincoln

CSCE150A

Introduction

Language Elements

General Form

Operators & Expressions

Formatting I/O

Running Programs

Common Pitfalls

Exercise

Problem

What are the results of the following expressions:

- $x = -15 * 4 / 2 * 3;$
- $y = 5 + 2 * 4;$
- $z = 4 - 5 * 2 - 4 + -10;$

- $x = -90$  (why not  $-10$ ?)
- $y = 13$
- $z = -20$

53 / 77

Notes

---

---

---

---

---

---

---

Nebraska

Lincoln

CSCE150A

Introduction

Language Elements

General Form

Operators & Expressions

Formatting I/O

Running Programs

Common Pitfalls

Make Expressions Unambiguous

$$x = (-15 * 4) / (2 * 3);$$

Even if unnecessary, add parentheses to improve readability

54 / 77

Notes

---

---

---

---

---

---

---

Nebraska

Lincoln

CSCE150A

Introduction

Language Elements

General Form

Operators & Expressions

Formatting I/O

Running Programs

Common Pitfalls

Writing Mathematical Formulas in C

Pitfalls

Common misconception: Mathematical Formulas in algebra and in C are *not* the same.

- Algebra: multiplication is implied:  $xy$
- C: Operations must be explicit:  $x * y$
- Algebra: division is written  $\frac{a+b}{c+d}$
- C: Cannot use such conveniences, must write  $(a + b)/(c + d)$

55 / 77

Notes

---

---

---

---

---

---

---

[illegible]

---

---

---

---

---

---

---

---

---

---

---

---

University of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common Pitfalls

59 / 77

## Formatting Values of Type double

- Recall the placeholder for doubles: `%f`
- Must specify both the total number of columns as well as the *precision* (number of decimal digits)
- Format: `%n.mf`
  - n* is the field width (minimum number of columns to be printed *including* the decimal)
  - m* is the number of digits after the decimal to be printed (may end up being padded with zeros)
- May or may not define both *n*, *m*.

Notes

---

---

---

---

---

---

---

University of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common Pitfalls

60 / 77

## Formatting Values of Type double

### Example

```
1 double pi = 3.141592;  
2 printf("%.2f\n",pi);  
3 printf("%.6.2f\n",pi);  
4 printf("%.15.10f\n",pi);
```

Result:

```
1 3.14  
2 3.14  
3 3.1415920000
```

Notes

---

---

---

---

---

---

---

University of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common Pitfalls

61 / 77

## Programming Tip

### Man pages

- On a UNIX system like cse, the command `man` (short for manual) can be used to read documentation on standard library functions.
- Example: `:prompt> man printf` gives a detailed description on `printf`
- Contains additional information: how to print leading zeros instead of blanks, etc.
- Can also look at web-based resources, and Kernighan and Ritchie's reference book

Notes

---

---

---

---

---

---

---



[illegible][illegible][illegible]

University of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common Pitfalls

65 / 77

Recall the mile-to-kilometer program.

```
1 #include <stdio.h>
2 int main(void)
3 {
4     double miles, kilometers;
5     printf("How many miles do you have?");
6
7     scanf("%lf",&miles);
8
9     kilometers = miles * 1.609;
10    printf("You have %f kilometers\n",kilometers);
11
12    return 0;
13 }
```

Notes

---

---

---

---

---

---

---

University of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common Pitfalls

66 / 77

Echo Prints vs. Prompts

- `scanf` gets a value for *miles* from the first (and only) line of the data file
- If we will only run the program in batch mode, there is no need for the prompting message
- We do need to output the answer, though:  
`printf("The distance in miles is %.2f.\n",miles);`
- However, we can also redirect the output to a file:  
`prompt:> conversion < mydata > result.txt`
- It's enough to echo only the number: `printf("%.2f.\n",miles);`

Notes

---

---

---

---

---

---

---

University of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common Pitfalls

67 / 77

Program-Controlled Input and Output Files

As an alternative to input/output redirection, C allows a program to explicitly name a file from which the program will take input and a file to which the program will send output. The steps needed to do this are:

- 1 Include `stdio.h`
- 2 Declare a variable of type `FILE *`.
- 3 Open the file for reading, writing or both.
- 4 Read/write to/from the file.
- 5 Close the file.

Notes

---

---

---

---

---

---

---

UNIVERSITY of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common Pitfalls

68 / 77

Program Example

File Input/Output

```
1 #include <stdio.h>
2 #define KMS_PER_MILE 1.609
3
4 int main(void) {
5     double kms, miles;
6     FILE *inp, *outp;
7
8     inp = fopen("distance.dat","r");
9     outp = fopen("distance.out","w");
10    fscanf(inp, "%lf", &miles);
11    fprintf(outp, "The distance in miles is %.2f.\n", miles);
12
13    kms = KMS_PER_MILES * miles;
14    fprintf(outp, "That equals %.2f kilometers.\n", miles);
15    fclose(inp);
16    fclose(outp);
17    return 0;
18 }
```

Notes

---

---

---

---

---

---

---

UNIVERSITY of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common Pitfalls

69 / 77

Common Programming Errors

- Syntax Errors
- Run-Time Errors
- Undetected Errors
- Logic Errors

Notes

---

---

---

---

---

---

---

UNIVERSITY of Nebraska  
Lincoln

CSCE150A

Introduction  
Language Elements  
General Form  
Operators & Expressions  
Formatting I/O  
Running Programs  
Common Pitfalls

70 / 77

Errors

- Bugs - Errors in a programs code.
- Debugging - Finding and removing errors in the program.
- When the compiler detects an error, it will output an error message.
  - May be difficult to interpret
  - May be misleading
- Three types of errors
  - Syntax error
  - Run-time error
  - Undetected error
  - Logic error

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

71 / 77

## Syntax Errors

A syntax error occurs when your code violates one or more grammar rules of C and is detected by the compiler at it attempts to translate your program. If a statement has a syntax error, it cannot be translated and your program will not be compiled.

Common syntax errors:

- Missing semicolon
- Undeclared variable
- Last comment is not closed
- A grouping character not closed ('(', '{', '[')

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

72 / 77

## Run-Time Errors

- Detected and displayed by the computer during the *execution* of a program
- Occurs when the program directs the computer to perform an illegal operation. Example: dividing a number by zero or opening a file that does not exist
- When a run-time error occurs, the computer will stop executing your program
- May display a useful (or not) error message
- Segmentation fault, core dump, bus error, etc.

Notes

---

---

---

---

---

---

---

Nebraska  
Lincoln

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

73 / 77

## Undetected Errors

- Code was correct and logical, executed fine, *but* led to incorrect results.
- Essential that you test your program on known correct input/outputs.
- Common formatting errors with `scanf/printf`: keep in mind the correct placeholders and syntax.

Notes

---

---

---

---

---

---

---

## Logic Errors

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

74 / 77

- Logic errors occur when a program follows a faulty algorithm.
- Usually do not cause run-time errors and do not display error messages—difficult to detect
- Must rigorously test your program with various inputs/outputs
- Pre-planning your algorithm with pseudocode or flow charts will also help you avoid logic errors

## Notes

---

---

---

---

---

---

## Questions

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

75 / 77

Questions?

## Notes

---

---

---

---

---

---

## Exercise

CSCE150A

Introduction  
Language  
Elements  
General Form  
Operators &  
Expressions  
Formatting  
I/O  
Running  
Programs  
Common  
Pitfalls

76 / 77

Debug the following program:

```
1  /*
2  * Calculate and display the difference of two input values.
3  */
4  #include <stdio.h>
5
6  int main(void)
7  {
8      int a, b; /* inputs
9      integer sum; /* sum of inputs */
10     printf("Input the first number: %d", &A);
11     printf("Input the second number: ");
12     scanf("%d", b);
13     a + b = sum;
14     printf("%d + %d = %d\n", a; b, sum);
15     return 0;
```

## Notes

---

---

---

---

---

---

## Exercise: Answer

CSCE150A

Introduction

Language  
Elements

General Form

Operators &  
Expressions

Formatting  
I/O

Running  
Programs

Common  
Pitfalls

77 / 77

```

1  /*
2  * Calculate and display the sum of two input values
3  */
4  #include <stdio.h>
5
6  int main(void)
7  {
8      int a, b; /* inputs */
9      int sum; /* sum of inputs */
10     printf("Input the first number: ");
11     scanf("%d", &a);
12     printf("Input the second number: ");
13     scanf("%d", &b);
14     sum = a + b;
15     printf("%d + %d = %d\n", a, b, sum);
16     return 0;
17 }
```

◀ ▶ ↺ ↻ 🔍 🔄

## Notes

---



---



---



---



---



---