CSC180: Lecture 5

Wael Aboulsaadat

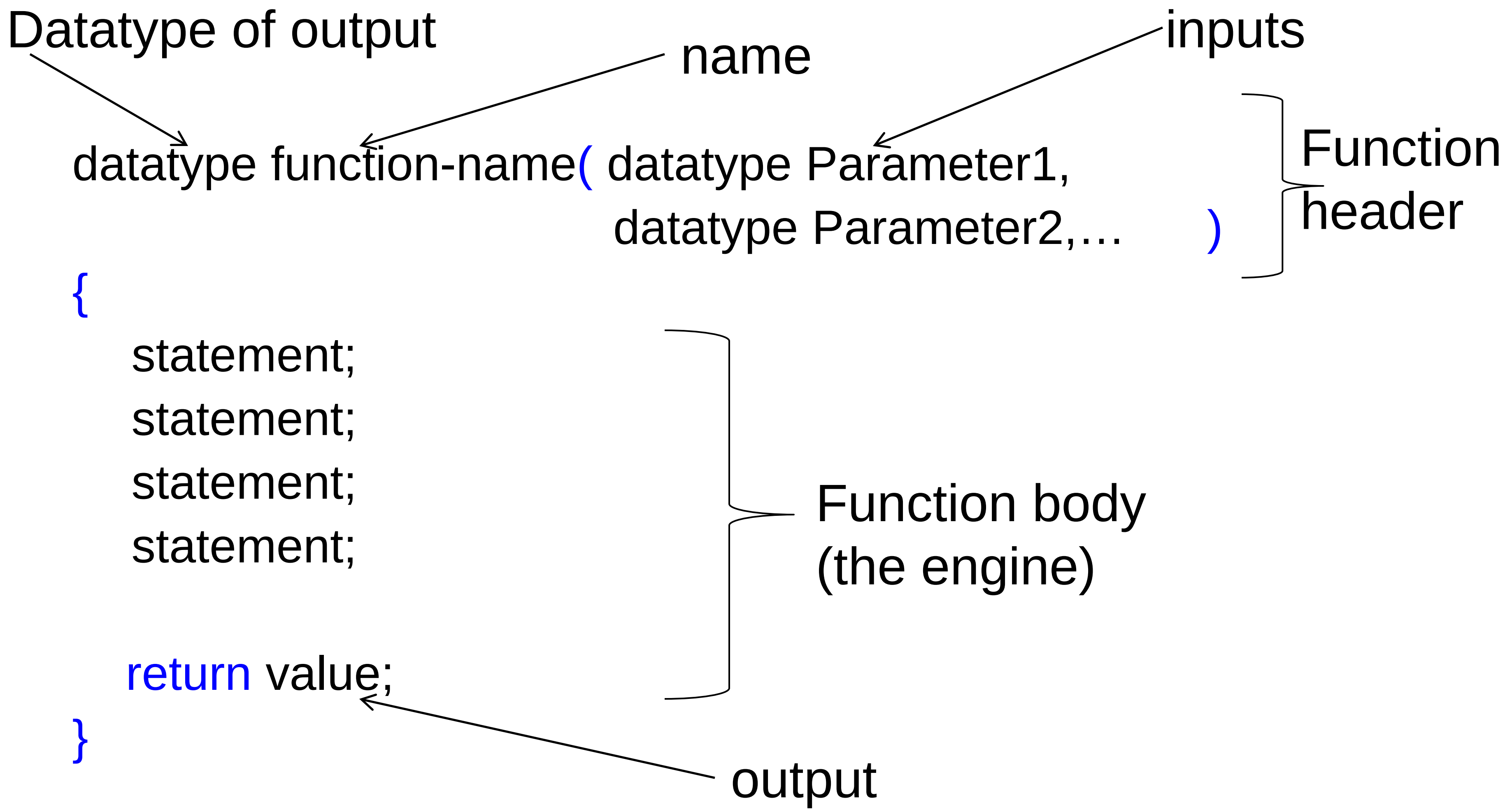
wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

Simple Functions

Function Syntax



```
datatype function-name( datatype Parameter1,
                        datatype Parameter2,…    )
{
    statement;
    statement;
    statement;
    statement;

    return value;
}
```

main is a special function

Output datatype

input (empty)

```
int main(        )
{
        first-statement-to-executed-in-program;

        return 0;
}
```

name is “reserved” by C

output should be 0 if no error during program execution

Typical C program

```
double func1( )
{
         ……
        return …;
 }

int func2( )
{
      ……
      return …;
 }

 double func3( )
 {
     ……
     return …;
 }

 int main( )
 {
       func3( );
       return….;
 }
```

Arithmetic Expressions

Arithmetic

- Arithmetic is performed with operators
  - + for addition
  - - for subtraction
  - * for multiplication
  - / for division
- Example: storing a product in the variable total_weight

```
total_weight  =  one_weight * number_of_bars;
```

Arithmetic

- Arithmetic is performed with operators
  - `+` for addition
  - `-` for subtraction
  - `*` for multiplication
  - `/` for division

- Example: storing a product in the variable total_weight

```
total_weight  =  one_weight * number_of_bars;
```

Results of Operators

- Arithmetic operators can be used with any numeric type
- An operand is a number or variable used by the operator
- Result of an operator depends on the types of operands
  - If both operands are int, the result is int
  - If one or both operands are double, the result is double

Division of Doubles

- Division with at least one operator of type double produces the expected results.

```
double divisor, dividend, quotient;
divisor = 3;
dividend = 5;
quotient = dividend / divisor;
```

  - quotient = 1.6666…
  - Result is the same if either dividend or divisor is of type int

Division of Integers

- Be careful with the division operator!
  - int / int produces an integer result
    (true for variables or numeric constants)

```
int dividend, divisor, quotient;
dividend = 5;
divisor = 3;
quotient = dividend / divisor;
```

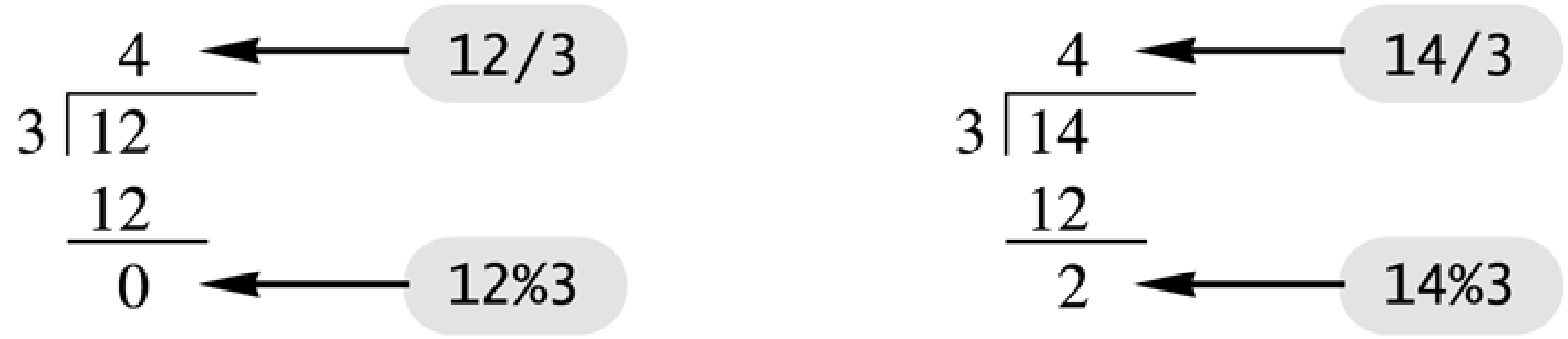
  - The value of quotient is 1, not 1.666…
  - Integer division does not round the result, the fractional part is discarded!

Integer Remainders

- % operator gives the remainder from integer division

```
int dividend, divisor, remainder;
dividend = 5;
divisor = 3;
remainder = dividend % divisor;
```

The value of remainder is 2


4
3 | 12
12
0
12/3
12%3
4
3 | 14
12
2
14/3
14%3

Arithmetic Expressions

- Use spacing to make expressions readable
  - Which is easier to read?

x+y*z   or   x + y * z

- Precedence rules for operators are the same as used in your algebra classes

- Use parentheses to alter the order of operations
  x + y * z   (y is multiplied by z first)
  (x + y) * z   (x and y are added first)

Arithmetic Expressions

Mathematical Formula	C Expression
$b^2 - 4ac$	`b*b - 4*a*c`
$x(y + z)$	`x*(y + z)`
$\frac{1}{x^2 + x + 3}$	`1/(x*x + x + 3)`
$\frac{a + b}{c - d}$	`(a + b)/(c - d)`

Operator Shorthand

- Some expressions occur so often that C contains to shorthand operators for them
- All arithmetic operators can be used this way
  - += count = count + 2; becomes
    count += 2;
  - *= bonus = bonus * 2; becomes
    bonus *= 2;
  - /= time = time / rush_factor; becomes
    time /= rush_factor;
  - %= remainder = remainder % (cnt1+ cnt2); becomes
    remainder %= (cnt1 + cnt2);

Boolean Expressions

Boolean Expressions

- Boolean expressions are expressions that are either true or false
- comparison operators such as '>' (greater than) are used to compare variables and/or numbers
  - (hours > 40)   Including the parentheses, is the boolean expression from the wages example
  - A few of the comparison operators that use two symbols (No spaces allowed between the symbols!)
    - >=   greater than or equal to
    - !=   not equal or inequality
    - = =   equal or equivalent

Comparison Operators

Math Symbol	English	C Notation	C Sample	Math Equivalent
$=$	equal to	`==`	`x + 7 == 2*y`	$x + 7 = 2y$
$\neq$	not equal to	`!=`	`ans != 'n'`	ans ≠ 'n'
$<$	less than	`<`	`count < m + 3`	count < m + 3
$\leq$	less than or equal to	`<=`	`time <= limit`	time ≤ limit
$>$	greater than	`>`	`time > limit`	time > limit
$\geq$	greater than or equal to	`>=`	`age >= 21`	age ≥ 21

AND

- Boolean expressions can be combined into more complex expressions with
  - && -- The AND operator
    - True if both expressions are true
- Syntax: (Comparison_1) && (Comparison_2)
- Example: if ((2 < x) && (x < 7))
  - True only if x is between 2 and 7
  - Inside parentheses are optional but enhance meaning

AND semantics

- Let’s look at the relationship between the semantic and logical operator known as the AND operator
- Consider:

  If the *car is fueled* <u>*AND*</u> the *engine works*,
  then the *engine will start*

- AND means that both conditions must be *true* in order for the conclusion to be *true*

AND Operator Truth Table

A	B	Output
F	F	F
F	T	F
T	F	F
T	T	T

OR

- | | -- The OR operator (no space!)
    - True if either or both expressions are true
- Syntax: (Comparison_1) | | (Comparison_2)
- Example: if ((x = = 1) | | (x = = y))
    - True if x contains 1
    - True if x contains the same value as y
    - True if both comparisons are true

OR Semantics

- Another basic operator is the OR

- Consider:

  If I have *cash* *OR* *a credit card*,

  then *I can pay the bill*

- OR works such that the output is *true*, if either of the two inputs is *true*

OR Operator Truth Table

A	B	Output
F	F	F
F	T	T
T	F	T
T	T	T

NOT

- ! -- negates any boolean expression
  - !(x < y)
    - True if x is NOT less than y
  - !(x = = y)
    - True if x is NOT equal to y
- ! Operator can make expressions difficult to understand…use only when appropriate

Inequalities

- Be careful translating inequalities to C
- if x < y < z   translates as

if ((x < y) && (y < z))

NOT

~~if (x < y < z)~~

Pitfall: Using = or ==

- ' = ' is the assignment operator
  - Used to assign values to variables
  - Example: `x = 3;`
- '= = ' is the equality operator
  - Used to compare values
  - Example: `if ( x == 3)`
- The compiler will accept this error:

  `if (x = 3)`

  but stores 3 in x instead of comparing x and 3
  - Since the result is 3 (non-zero), the expression is true

Pitfall: short circuit evaluation

- if (bool-expression1 && bool-expression2)

  if bool-expression1 is F, why eval bool-expression2 ?

- if (bool-expression1 || bool-expression2)

  if bool-expression1 is T, why eval bool-expression2 ?

Pitfall: working with AND, OR and NOT

- !(1 || 0) ANSWER: 0
- !(1 || 1 && 0) ANSWER: 0 (AND is evaluated before OR)
- !((1 || 0) && 0) ANSWER: 1 (Parenthesis are useful)

Simple Flow of Control

Simple Flow of Control

- Flow of control
  - The order in which statements are executed

- Branch
  - Lets program choose between two alternatives

Branch Example

- To calculate hourly wages there are two choices
  - Regular time (up to 40 hours)
    - gross_pay = rate * hours;
  - Overtime (over 40 hours)
    - gross_pay = rate * 40 + 1.5 * rate * (hours - 40);
  - The program must choose which of these to use

Designing the Branch

- Decide if (hours >40) is true
    - If it is true, then use
      gross_pay = rate * 40 + 1.5 * rate * (hours - 40);

    - If it is not true, then use
      gross_pay = rate * hours;

Implementing the Branch

- if-else statement is used in C to perform a branch

```
if (hours > 40)
    gross_pay  =  rate * 40 + 1.5 * rate * (hours - 40);
else
    gross_pay = rate * hours;
```

Syntax for an `if-else` Statement

A Single Statement for Each Alternative:

```
if (Boolean_Expression)
    Yes_Statement
else
    No_Statement
```

A Sequence of Statements for Each Alternative:

```
if (Boolean_Expression)
{
    Yes_Statement_1
    Yes_Statement_2
      . . .
    Yes_Statement_Last
}
else
{
    No_Statement_1
    No_Statement_2
      . . .
    No_Statement_Last
}
```

if-else Flow Control (1)

- if (boolean expression)
      true statement
  else
      false statement
- When the boolean expression is true
  - Only the true statement is executed
- When the boolean expression is false
  - Only the false statement is executed

if-else Flow Control (2)

- if (boolean expression)
```
      {
                    true statements
       }
else
     {
```

-
  ```
                    false statements
      }
  ```
- When the boolean expression is true
  - Only the true statements enclosed in { } are executed
- When the boolean expression is false
  - Only the false statements enclosed in { } are executed

Compound Statements

- A compound statement is more than one statement enclosed in { }
- Branches of if-else statements often need to execute more that one statement
- Example:

```
if (boolean expression)
  {
        true statements….
  }
 else
  {
        false statements….
   }
```

Branches Conclusion

- Can you
  - Write an if-else statement that outputs the word High if the value of the variable score is greater than 100 and Low if the value of score is at most 100? The variables are of type int.

  - Write an if-else statement that outputs the word Warning provided that either the value of the variable temperature is greater than or equal to 100, or the of the variable pressure is greater than or equal to 200, or both. Otherwise, the if_else sttement outputs the word OK. The variables are of type int.