Computer Science I

CS 135

1. Introduction to Programming

René Doursat

Department of Computer Science & Engineering
University of Nevada, Reno

Spring 2006

Computer Science I

CS 135

0. Course Presentation
1. Introduction to Programming
2. Functions I: Passing by Value
3. File Input/Output
4. Predefined Functions
5. If and Switch Controls
6. While and For Loops
7. Functions II: Passing by Reference
8. 1-D and 2-D Arrays

Computer Science I

CS 135

1. Introduction to Programming

a. How to Develop a Program

b. Writing Pseudocode

c. First Elements of C++

d. Looking Under the Hood

Computer Science I

CS 135

1. Introduction to Programming

a. How to Develop a Program

- ✓ A program is like a recipe
- ✓ Steps in program development
- ✓ Procedural programming

b. Writing Pseudocode

c. First Elements of C++

d. Looking Under the Hood

1.a How to Develop a Program

A program is like a recipe

Pasta for six

- *boil 1 quart salty water*
- *stir in the pasta*
- *cook on medium until "al dente"*
- *serve*

1.a How to Develop a Program

A program is like a recipe

- What is programming?
  - ✓ programming can be defined as
    - the development of a **solution** to an identified problem and
    - the setting up of a related series of instructions that will produce the desired results
  - ✓ generally, programming is the construction of an **algorithm**

1.a How to Develop a Program

A program is like a recipe

- What is an algorithm?
  - ✓ informally, a general method for solving a problem, such as a recipe
  - ✓ formally, a **set of precise steps** that describe exactly the tasks to be performed and in which order
  - ✓ an algorithm must
    - be precise and unambiguous
    - give the correct solution in all cases
    - eventually end
  - ✓ an algorithm frequently involves repetition of an operation

1.a How to Develop a Program

Steps in program development

- The 7 basic steps in the development of a program

1. define the problem
2. outline the solution
3. develop the outline into an algorithm
4. test the algorithm for correctness
5. code the algorithm into a specific prog. language
6. run the program on the computer
7. document and maintain the program

1.a How to Develop a Program

Steps in program development

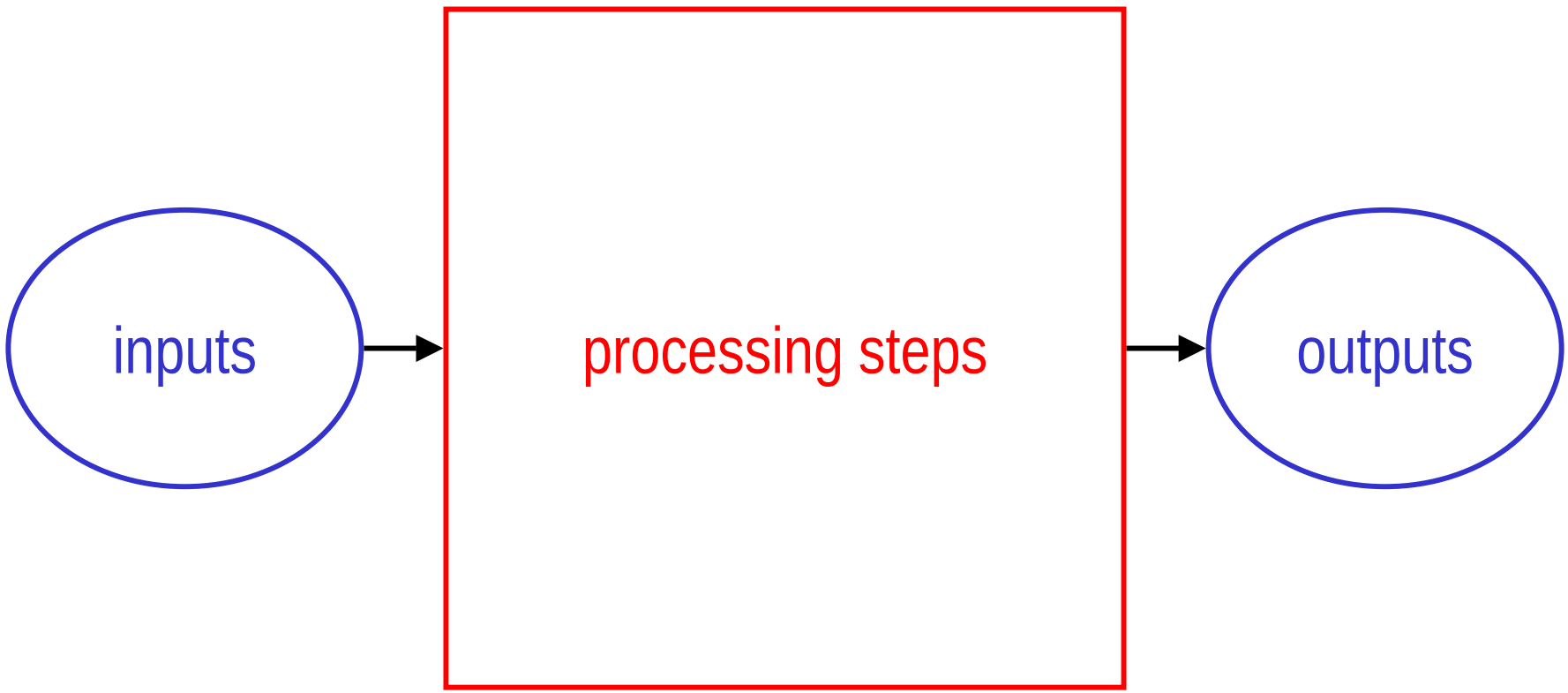
1. Define the problem

- ✓ to help with initial analysis, the problem should be divided into three separate components:
  - the inputs
  - the outputs
  - the processing steps to produce the required outputs from the inputs

1.a How to Develop a Program

Steps in program development

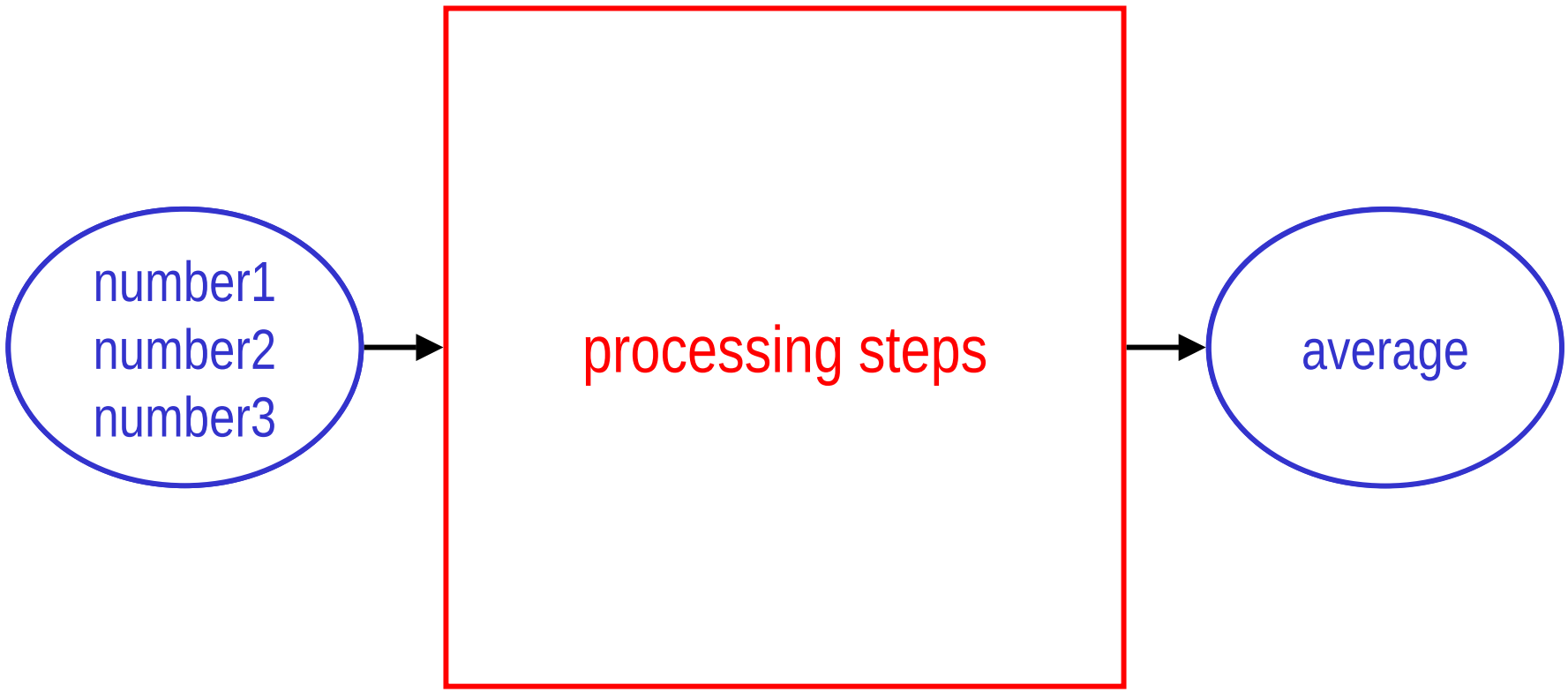
1. Define the problem



1.a How to Develop a Program

Steps in program development

- Example: find the average of three numbers
  - ✓ what are the inputs and outputs?



1.a How to Develop a Program

Steps in program development

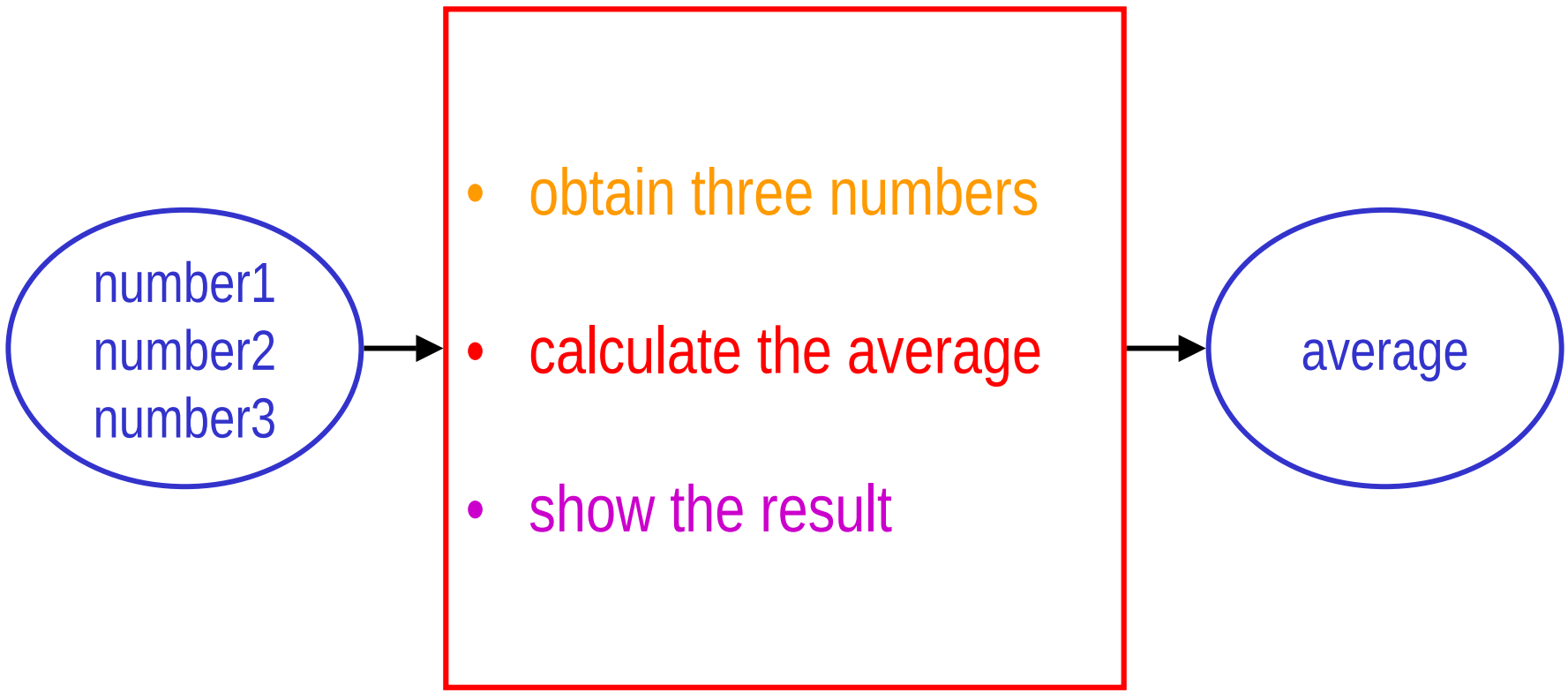
2. Outline the solution

- ✓ decompose the problem in smaller elements and produce a rough draft of the solution:
  - the major processing steps involved
  - the major subtasks (if any)
  - the major control structures
  - the major variables and record structures
  - the mainline logic

1.a How to Develop a Program

Steps in program development

- Example: find the average of three numbers
  - ✓ what are the processing steps?

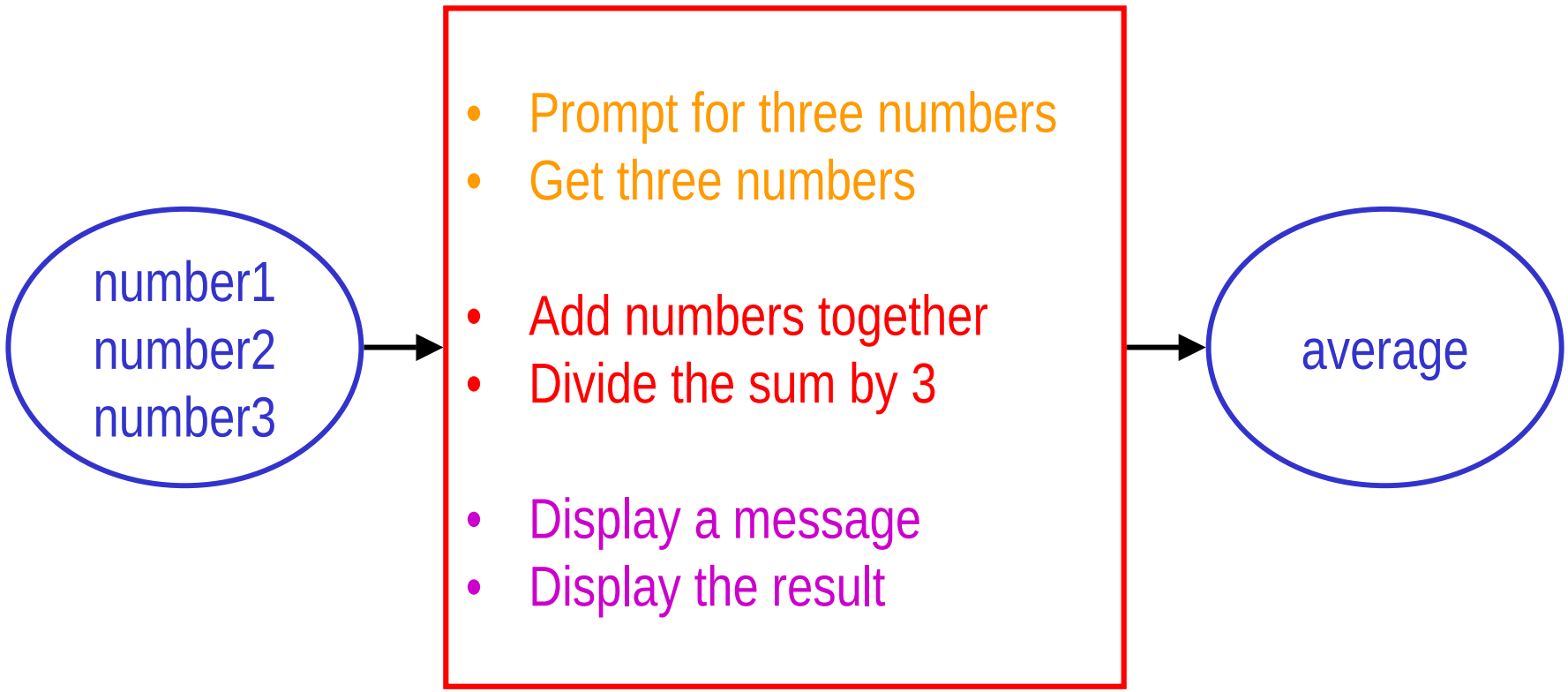


1.a How to Develop a Program

Steps in program development

3. Develop the outline into an algorithm

- ✓ the solution outline of Step 2 is expanded into an algorithm

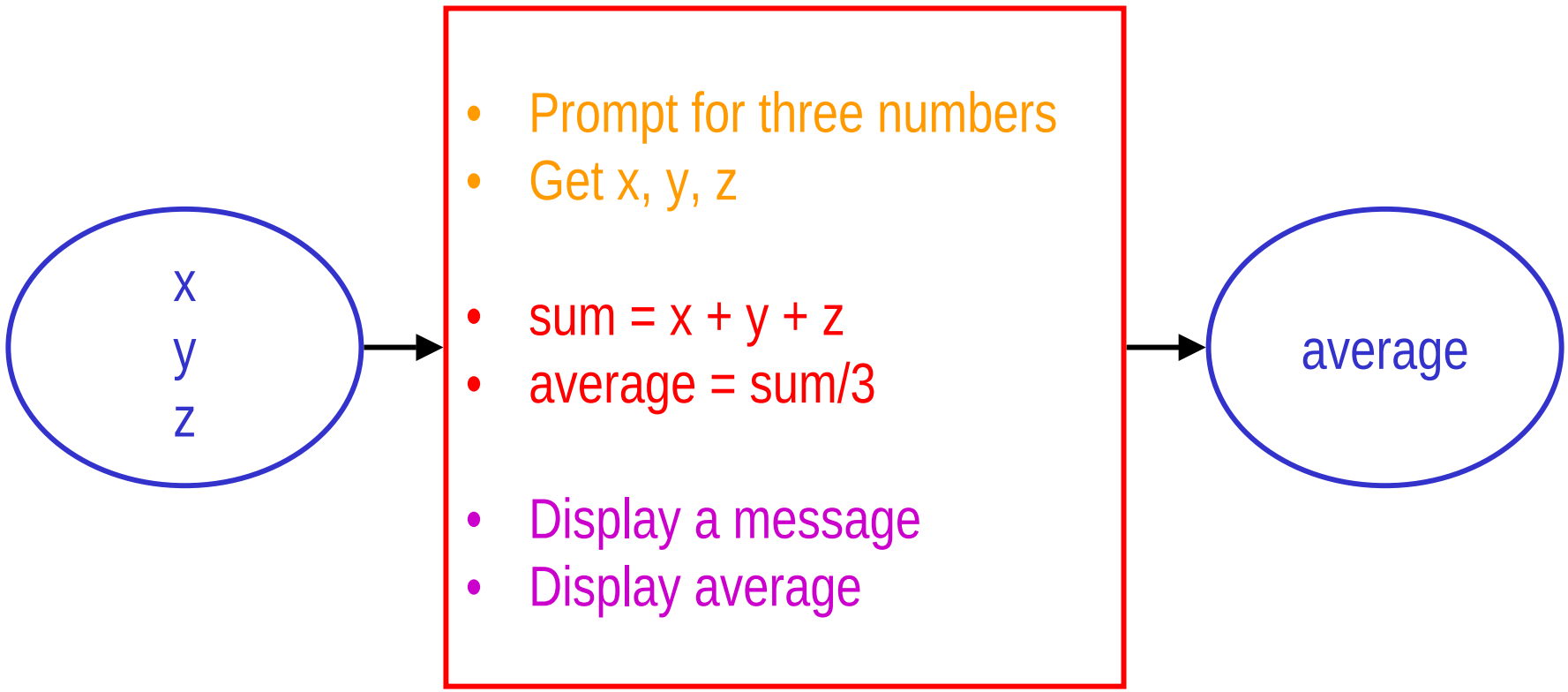


1.a How to Develop a Program

Steps in program development

3. Develop the outline into an algorithm

- ✓ here is an equivalent algorithm in a more formal style



1.a How to Develop a Program

Steps in program development

4. Test the algorithm for correctness

- ✓ testing or “desk-checking” is one of the most important step in the development of a program, yet it is often forgotten
- ✓ the main purpose of testing the algorithm is to identify major logic errors early, so that they may be easily corrected

- sum = x + y + z
- average = sum/3

- ✓ try different test values by hand!

x	y	z	sum	avg
1	1	1	3	1
0	0	6	6	2
13	15	17	45	15
...	...	...	...	...

1.a How to Develop a Program

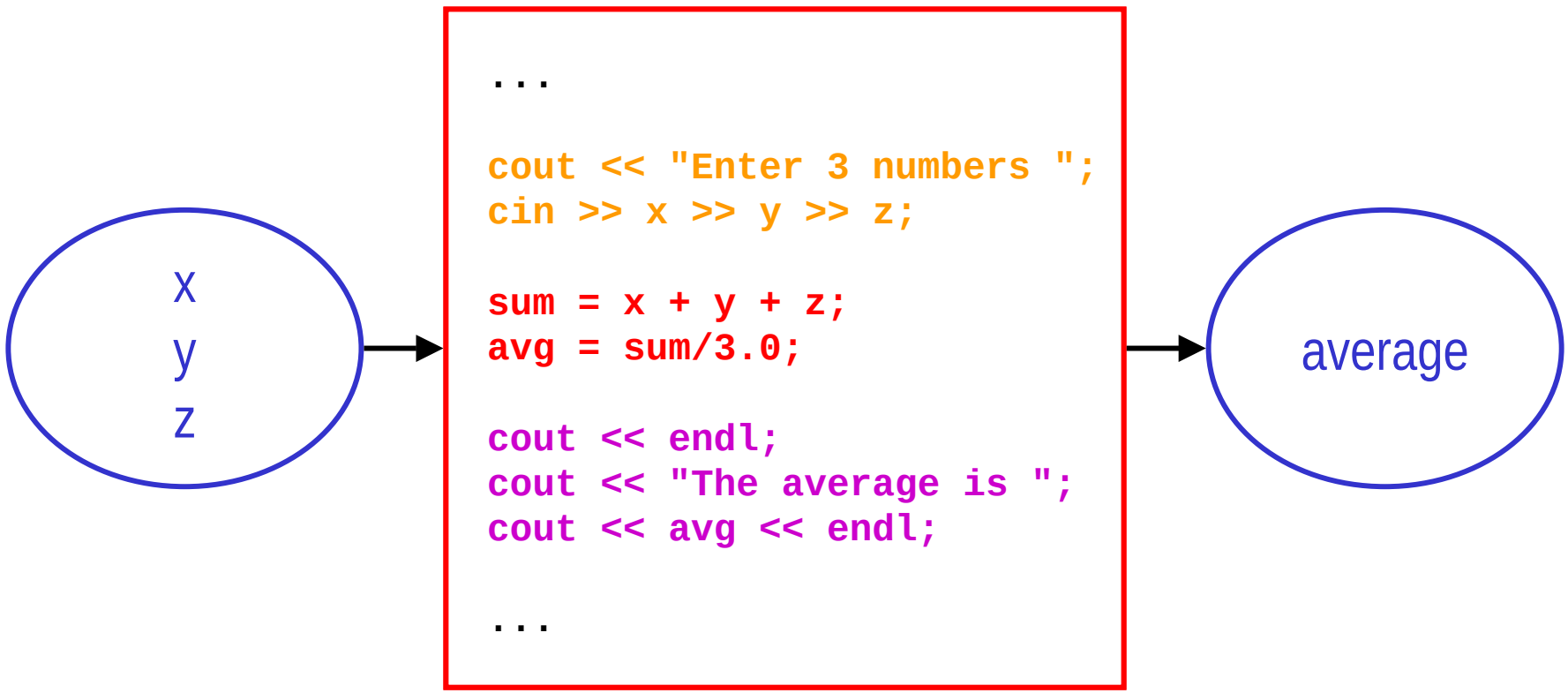
Steps in program development

5. Code the algorithm into a specific programming language
   - ✓ only after all design considerations have been met should you actually start to code the program into your chosen programming language:
     - C++
     - Java
     - FORTRAN
     - Basic
     - COBOL
     - etc.

1.a How to Develop a Program

Steps in program development

- Example: find the average of three numbers
  - ✓ core of the program in C++



1.a How to Develop a Program

Steps in program development

6. Run the program on the computer

- ✓ this step uses a program **compiler** and some test data to "machine-check" the code for errors:
  - syntax errors are detected at compile time
  - logic errors are detected at run time
- ✓ a compiler is like an interpreter: it translates a high-level language (such as C++) into low-level machine language (lots of 0's and 1's)
- ✓ compiling and running the program can be the most exciting and, at the same time, most frustrating part of the development process

1.a How to Develop a Program

Steps in program development

7. Document and maintain the program

- ✓ documentation should not be the last step but an ongoing task throughout the development process
  - external documentation: specifications, implementation, user manual, etc.
  - internal documentation: comments in the code

1.a How to Develop a Program

Steps in program development

- Summary

1. Define the problem
2. Outline the solution
3. Develop the outline into an algorithm
4. Test the algorithm for correctness

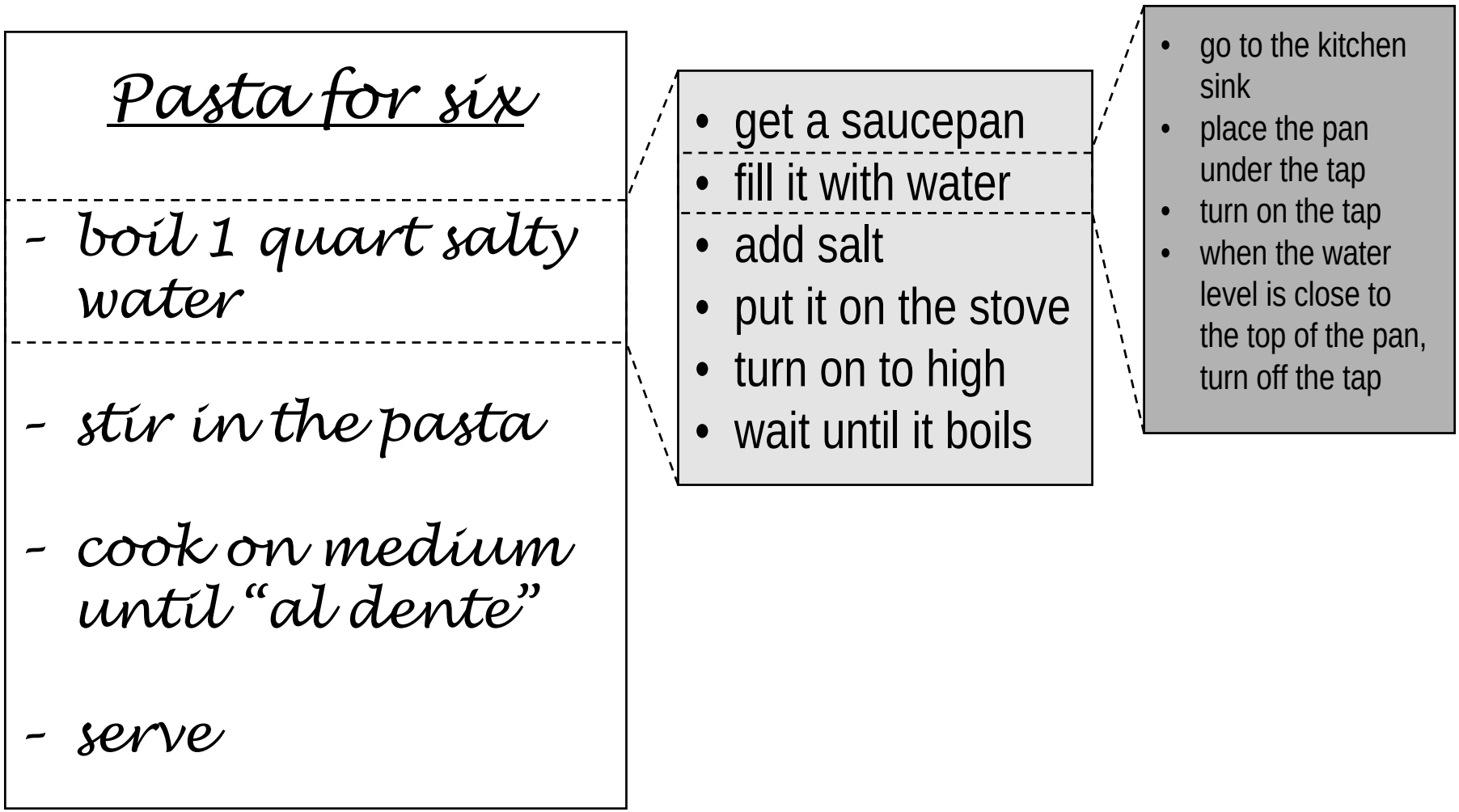
} design

5. Code the algorithm into a specific lang.
6. Run the program on the computer
7. Document and maintain the program

} implementation

1.a How to Develop a Program

Procedural programming



1.a How to Develop a Program

Procedural programming

- Top-down development
  - ✓ in the top-down development of a program design, a general solution to the problem is outlined first
  - ✓ this is then broken down gradually into more detailed steps until finally the most detailed levels have been completed
  - ✓ hierarchy of procedures, subtasks, and elementary steps
- Modular design
  - ✓ procedural programming also incorporates the concept of modular design, which involves grouping tasks together because they all perform the same function
  - ✓ modular design is connected directly to top-down development

Computer Science I

CS 135

1. Introduction to Programming

a. How to Develop a Program

- ✓ A program is like a recipe
- ✓ Steps in program development
- ✓ Procedural programming

b. Writing Pseudocode

c. First Elements of C++

d. Looking Under the Hood

Computer Science I

CS 135

1. Introduction to Programming

a. How to Develop a Program

b. Writing Pseudocode

- ✓ What is pseudocode?
- ✓ Six basic computer operations
- ✓ The structure theorem

c. First Elements of C++

d. Looking Under the Hood

1.b Writing Pseudocode

What is pseudocode?

Adding up a list of prices

Turn on calculator
Clear calculator
Repeat the following instructions
    Key in dollar amount
    Key in decimal point (.)
    Key in cents amount
    Press addition (+) key
Until all prices have been entered
Write down total price
Turn off calculator

1.b Writing Pseudocode

What is pseudocode?

- Pseudocode is a way to write an algorithm (recipe)
  - ✓ flowcharts are another popular way of representing algorithms
  - ✓ pseudocode is easier to read and write and allows the programmer to concentrate on the logic of the problem
  - ✓ pseudocode is really "structured English"
    - statements are written in simple English
    - each instruction is written on a separate line
    - each set of instructions is written from top to bottom, with only one entry and one exit
    - groups of statements may be formed into modules, and that group given a name

1.b Writing Pseudocode

Basic computer operations

- There are six basic computer operations

1. a computer can receive information
2. a computer can put out information
3. a computer can perform arithmetic
4. a computer can assign a value to a variable or memory location
5. a computer can compare two variables and select one of two alternate actions
6. a computer can repeat a group of actions

1.b Writing Pseudocode

Basic computer operations

1. A computer can receive information

- ✓ Get is used when the algorithm must receive input from the keyboard:
  - Get filename
  - Get class number

- ✓ Read is used when the algorithm must receive input from a file:
  - Read course description (from file)
  - Read student names (from file)

1.b Writing Pseudocode

Basic computer operations

2. A computer can put out information

- ✓ Print is used when the output must be sent to the printer:
  - Print 'Program Completed'
- ✓ Write is used when the output must be written to a file:
  - Write student names
- ✓ Display and Prompt are used when the output must be displayed on the screen:
  - Display 'Hello world!'
  - Prompt for class number (generally followed by Get)

1.b Writing Pseudocode

Basic computer operations

3. A computer can perform arithmetic

- ✓ Either actual mathematical symbols or words can be used:
  - Multiply Length by Width to Compute Area
  - Area = Length * Width
- ✓ Words and equivalent symbols used in pseudocode:
  - Add or +
  - Subtract or –
  - Multiply or *
  - Divide or /
  - Modulus or %
  - Parentheses or ( )
- ✓ Compute and Calculate also possible:
  - Compute degrees Celsius
  - C = (F – 32) / 1.8

1.b Writing Pseudocode

Basic computer operations

4. A computer can assign a value to a variable or memory location
   - ✓ Initialize or Set are used to give data an initial value:
     - Initialize total_price to 0
   - ✓ = or ← are used to assign a value as a result of processing:
     - total_price ← cost_price + sales_tax
   - ✓ Save or Store are used to keep a variable for later use:
     - Save customer_name in last_customer_name

1.b Writing Pseudocode

Basic computer operations

- Example of pseudocode

Find the mean age of the class

1. Prompt user for number_students
2. Get number_students
3. Prompt user for student_ages
4. Get student_ages

5. Add student_ages into total_age
6. Divide total_age by number_students
7. Set average to the result

8. Display average

1.b Writing Pseudocode

Basic computer operations

- Alternative pseudocode

Find the mean age of the class

1. Prompt user for number_students
2. Get number_students
3. Prompt user for student_ages
4. Get student_ages

---

5. total_age = Sum of student_ages
6. average = total_age / number_students

---

7. Display average

Computer Science I

CS 135

1. Introduction to Programming

a. How to Develop a Program

b. Writing Pseudocode

c. First Elements of C++

- ✓ The basics of a C++ program
- ✓ Data types
- ✓ Arithmetic operators
- ✓ Expressions
- ✓ Variables
- ✓ Type casting
- ✓ ASCII characters
- ✓ Input and output

d. Looking Under the Hood

1.c First Elements of C++

The basics of a C++ program

```
#include <iostream>
using namespace std;

int main()
{
      /* read data */
      cout << "Enter 3 numbers ";
      cin >> x >> y >> z;

      sum = x + y + z;    // calculate
      avg = sum/3.0;      // average

      cout << endl;
      cout << "The average is ";
      cout << avg << endl;

      return 0;
}
```

1.c First Elements of C++

The basics of a C++ program

➢ Structure of a C++ program

- ✓ a C++ program is a collection of one or more subprograms, called **functions**
- ✓ a subprogram or a function is a collection of statements that accomplishes something when executed
- ✓ every C++ program has at least one function called **`main`**
- ✓ the smallest individual unit of a program is called a “token”

➢ Tokens of a C++ program

- ✓ special symbols
- ✓ word symbols
- ✓ identifiers

1.c First Elements of C++

The basics of a C++ program

Special symbols

- ✓ mathematical symbols: `+` `-` `*` `/` ...
- ✓ punctuation marks: `.` `;` `?` `,` ...
- ✓ two-character symbols: `<=` `!=` `==` `>=` ...
- ✓ etc.

Word symbols

- ✓ `int`, `float`, `char`, `void`, `return`, etc.
- ✓ also called reserved words or **keywords**, as they belong to the language
  - cannot be redefined and reused
  - are always lowercase

1.c First Elements of C++

The basics of a C++ program

Identifiers

- **`main, x, y, sum, student_name,`** etc.
- identifiers are user-created names of things that appear in programs (variables, constants, functions, etc.):
  - consist of letters, digits, and the underscore character _
  - must begin with a letter or underscore
  - are case sensitive
- should be meaningful: **`student_name`** better than **`sdtnm`**
- there exists predefined identifiers, such as **`cout`** and **`cin`**
  - unlike reserved words, predefined identifiers may be redefined, but not a good idea as it can be confusing

1.c First Elements of C++

The basics of a C++ program

- Examples of legal and illegal identifiers

Identifier	Legal	Illegal	
`first name`		✗	(space break)
`first_name`	✓		
`firstName`	✓		
`1st_name`		✗	(first char is digit)
`first-name`		✗	(hyphen is symbol)
`_name1`	✓		

1.c First Elements of C++

The basics of a C++ program

```
#include <iostream>
using namespace std;

int main()
{
        /* read data */
        cout << "Enter 3 numbers ";
        cin >> x >> y >> z;

        sum = x + y + z;     // calculate
        avg = sum/3.0;       // average

        cout << endl;
        cout << "The average is ";
        cout << avg << endl;

        return 0;
}
```

char symbols
keywords
identifiers
literals
strings
comments

1.c First Elements of C++

Data types

- Different data types for different programs
  - ✓ for example, some programs work with numbers (scientific calculation), some other programs manipulate names (alphabetizing lists) and some use both (grading)
  - ✓ numbers and words are distinct **data types**
- Categories of data types in C++
  - ✓ simple data type
    - integral data type = integer numbers (without a decimal)
    - floating-point data type = decimal numbers
    - enumeration data type = programmer-created type
  - ✓ structured data type & pointers

1.c First Elements of C++

Data types

- Integral data types
  - ✓ `int` represents integer numbers, for example:
    - `0`, `37`, `-45`, `12500` (no comma inside number)
    - minimum: `-2147483648`
    - maximum: `2147483647`
  - ✓ `char` represents any single character, for example:
    - `'a'`, `'2'`, `'B'`, `'$'`, `' '` (in single quotes)
    - also represents small integers between `-128` and `127`
  - ✓ `bool` represents logical (Boolean) values and can be only
    - `true` or `false` (these are keywords)

1.c First Elements of C++

Data types

- Floating-point data types
  - ✓ `float` represents real numbers, for example:
    - `75.924`, `-1.482`, `0.0018`, `180.00`
    - `7.5924e1`, `-1.482e0`, `1.8e-3`, `1.8e2` (in scientific notation)
    - maximum of 6 or 7 significant digits
    - bounds: `-3.4e38` and `3.4e38`
  - ✓ `double` also represent real numbers but with double precision (more significant digits and larger interval)
    - maximum of 15 significant digits
    - bounds: `-1.7e308` and `1.7e308`

1.c First Elements of C++

Arithmetic operators

- The C++ arithmetic operators are
  - ✓ addition `+`
  - ✓ subtraction `-`
  - ✓ multiplication `*`
  - ✓ division `/`
  - ✓ remainder `%` (mod operator: 11 % 3 = 2)
  - ✓ `+`, `-`, `*` and `/` can be used with both integral and floating-point data types
- Two types of operators
  - ✓ unary operators have only one operand: `-` ☐ or `+` ☐
  - ✓ binary operators have two operands: ☐ `*` ☐ or ☐ `+` ☐

1.c First Elements of C++

Arithmetic operators

- Order of precedence
  - ✓ unary operators are evaluated first
  - ✓ then all operations inside parentheses `(  )` are evaluated next
  - ✓ then `*`, `/` and `%` are at the same level of precedence and are evaluated next
  - ✓ finally `+` and `-` are at the same level of precedence and are evaluated last
  - ✓ when operators are on the same level, evaluation is performed from left to right
  - ✓ example: `3 + 7 * 6` evaluates to `45`
  - ✓ example: `(3 + 7) * 6` evaluates to `60`

1.c First Elements of C++

Expressions

- Integer expressions
  - ✓ all operands are integer
  - ✓ the result is an integer
  - ✓ example: `10 / 4` evaluates to `2`
  - ✓ example: `72 - 7 * (-60 % 8)` evaluates to `100`
- Floating-point expressions
  - ✓ all operands are floating-point
  - ✓ the result is floating-point
  - ✓ example: `10.0 / 4.0` evaluates to `2.25`
  - ✓ example: `72.36 - 0.09 * 4.0` evaluates to `72.0`

1.c First Elements of C++

Expressions

- Mixed expressions
  - ✓ operands are of different data types, integer and floating-point
  - ✓ example: `5.4 * 2 - 13.6 + 21 / 6`
- Evaluation rules for binary operators
  - ✓ if both operands are integer, then result is integer
  - ✓ if both operands are floating-point, then result is floating-point
  - ✓ if one operand is integer and the other floating-point, then the integer is changed to floating-point and the result is floating-point
  - ✓ example: `5.4 * 2` `- 13.6 +` `21 / 6` yields `0.2`

`10.8` `3`

1.c First Elements of C++

Variables

- Variables allow for formulas and generic processing
  - ✓ variables are like “unknowns” in math, for example:
    - `surface = length * width;`
    - `total = price + (price * 0.08);`
    - `cout << "The name is " << name;`
- Variable assignment
  - ✓ form of assignment statement: **`variable`** `=` *expression*`;`
  - ✓ *expression* is evaluated and its value is assigned to the variable on the left side (**no** expression on the left side!)
  - ✓ `=` is called the assignment operator
  - ✓ for example: `i = i + 1` increases variable `i` by 1

1.c First Elements of C++
Variables

- Compound operators allow more concise assignments
  - ✓ most compound operators are two-character symbols: an arithmetic operator followed by the equal sign (no space)
    - compound addition `+=` and subtraction `-=`
    - compound multiplication `*=` and division `/=`
    - compound remainder `%=`
  - ✓ they are shortcuts used in <u>some</u> assignments where the same variable is on both sides of the equal sign, for example:
    - `i += 1` is the same as `i = i + 1`
    - `x *= y - 3` is the same as `x = x * (y - 3)`
    - . . . but is *not* the same as `x = x * y - 3` (this last assignment doesn't have an easy compound equivalent)

1.c First Elements of C++

Variables

- Variables must be declared before they can be used
  - ✓ form of the declaration statement: *data-type variable_name*;
    - `int age;`
    - `float surface;`
    - `char name_initial;`
  - ✓ once declared, variables can be assigned values using an assignment statement
  - ✓ variable declarations can be placed anywhere in a function, but generally at the beginning
  - ✓ variables of the same data type can be grouped together in a single declaration

1.c First Elements of C++

Variables

- Variables can be initialized at declaration or later

all declarations at the beginning

```
...

int num1, num2;  // only declared
int num3 = 100;  // declared
                 // and initialized

float average, percentage;  // declared

num1 = 7;          // initialized
num2 = 20;         // initialized

average = (num1 + num2)/2;  // assigned
percentage = average/100;   // assigned

...
```

- ✓ result: **average** is **13.0**, **percentage** is **0.13**

1.c First Elements of C++

Variables

```
...

int num1, num2;  // only declared
int num3 = 100;  // declared
                 // and initialized

float average, percentage;  // declared

num1 = 7;          // initialized
num2 = 20;         // initialized

average = (num1 + num2)/2.0;// assigned
percentage = average/100;   // assigned

...
```

✓ result: **average** is **13.5**, **percentage** is **0.135**

1.c First Elements of C++

Variables

```
...

int num1, num2;  // only declared
int num3 = 100;  // declared
                 // and initialized

int average, percentage;  // declared

num1 = 7;         // initialized
num2 = 20;        // initialized

average = (num1 + num2)/2;  // assigned
percentage = average/100;   // assigned

...
```

✓ result: **average** is **13** , **percentage** is **0**

1.c First Elements of C++

Type casting

- You can also explicitly convert one type into another
  - ✓ this is called “type casting” or “type conversion” and can be written in two ways:
    - `age = int(24.7);` yields a value of 24
    - `age = (int)24.7;`
  - ✓ ex: from floating-point to integer → drop the decimal part
    - `tax_payment = int(dollar_cents);`
  - ✓ ex: from integer to floating-point → preserve precision
    - `mean_age = float(total_age)/num;`
  - ✓ ex: from character to integer → alphabetize
    - `rank = int('a');` yields a value of 97

1.c First Elements of C++

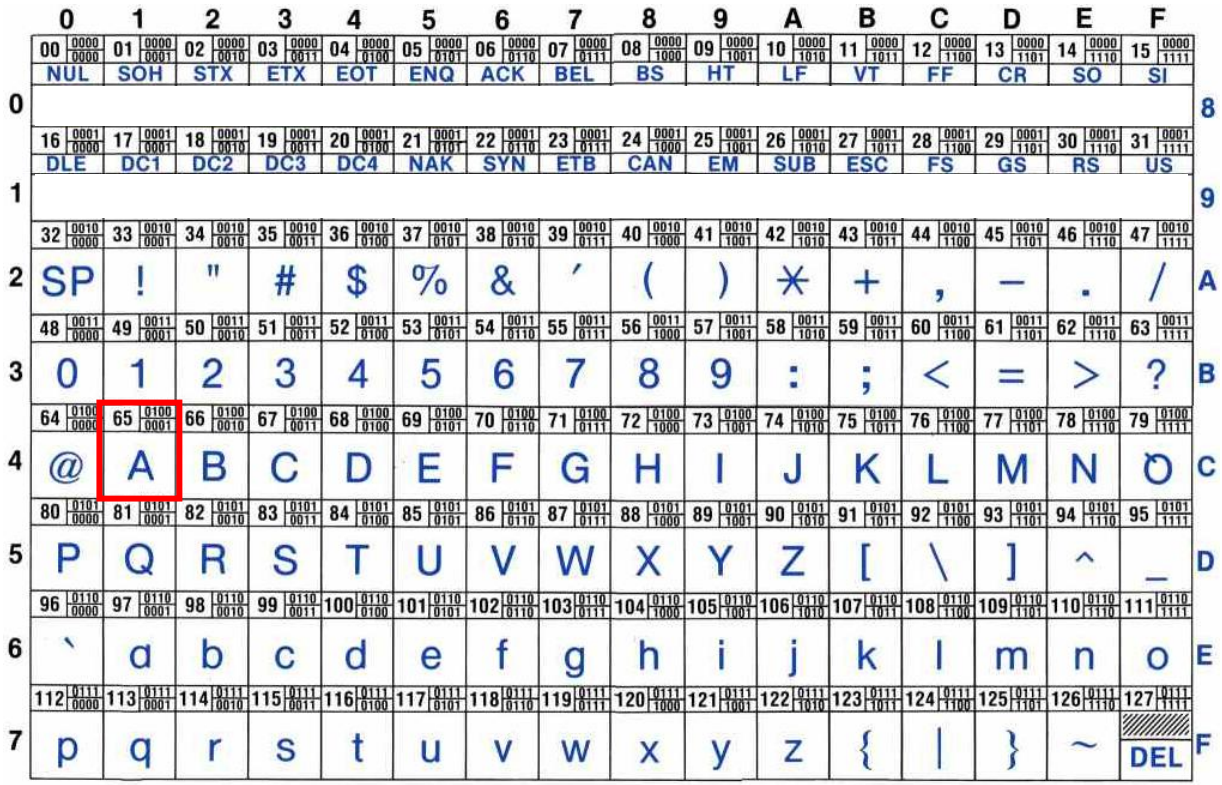
ASCII characters

- American Standard Code for Information Interchange
  - ✓ the ASCII standard is character set and a character encoding based on the Roman alphabet, as used in modern English
  - ✓ contains printable characters and control characters
  - ✓ **printable** characters are
    - alphabetical characters: lowercase and uppercase letters
    - numerical characters: digits from **0** to **9**
    - all symbol characters: **!** , @, **#**, **$**, **%**, **^**, **&**, *****, **(** , **)**, etc.
  - ✓ **control** (nonprintable characters) include
    - line feed, carriage return, end of file, escape, delete, etc.

1.c First Elements of C++

ASCII characters

- American Standard Code for Information Interchange
  - ✓ the code of a character is an integer between 0 and 127
  - ✓ example: `int('A')` gives `65`, and `char(65)` gives `'A'`



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
	00 0000 0000	01 0000 0001	02 0000 0010	03 0000 0011	04 0000 0100	05 0000 0101	06 0000 0110	07 0000 0111	08 0000 1000	09 0000 1001	10 0000 1010	11 0000 1011	12 0000 1100	13 0000 1101	14 0000 1110	15 0000 1111	
	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI	
0																	8
	16 0001 0000	17 0001 0001	18 0001 0010	19 0001 0011	20 0001 0100	21 0001 0101	22 0001 0110	23 0001 0111	24 0001 1000	25 0001 1001	26 0001 1010	27 0001 1011	28 0001 1100	29 0001 1101	30 0001 1110	31 0001 1111	
	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US	
1																	9
	32 0010 0000	33 0010 0001	34 0010 0010	35 0010 0011	36 0010 0100	37 0010 0101	38 0010 0110	39 0010 0111	40 0010 1000	41 0010 1001	42 0010 1010	43 0010 1011	44 0010 1100	45 0010 1101	46 0010 1110	47 0010 1111	
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	–	.	/	A
	48 0011 0000	49 0011 0001	50 0011 0010	51 0011 0011	52 0011 0100	53 0011 0101	54 0011 0110	55 0011 0111	56 0011 1000	57 0011 1001	58 0011 1010	59 0011 1011	60 0011 1100	61 0011 1101	62 0011 1110	63 0011 1111	
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	B
	64 0100 0000	65 0100 0001	66 0100 0010	67 0100 0011	68 0100 0100	69 0100 0101	70 0100 0110	71 0100 0111	72 0100 1000	73 0100 1001	74 0100 1010	75 0100 1011	76 0100 1100	77 0100 1101	78 0100 1110	79 0100 1111	
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	C
	80 0101 0000	81 0101 0001	82 0101 0010	83 0101 0011	84 0101 0100	85 0101 0101	86 0101 0110	87 0101 0111	88 0101 1000	89 0101 1001	90 0101 1010	91 0101 1011	92 0101 1100	93 0101 1101	94 0101 1110	95 0101 1111	
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	D
	96 0110 0000	97 0110 0001	98 0110 0010	99 0110 0011	100 0110 0100	101 0110 0101	102 0110 0110	103 0110 0111	104 0110 1000	105 0110 1001	106 0110 1010	107 0110 1011	108 0110 1100	109 0110 1101	110 0110 1110	111 0110 1111	
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	E
	112 0111 0000	113 0111 0001	114 0111 0010	115 0111 0011	116 0111 0100	117 0111 0101	118 0111 0110	119 0111 0111	120 0111 1000	121 0111 1001	122 0111 1010	123 0111 1011	124 0111 1100	125 0111 1101	126 0111 1110	127 0111 1111	
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL	F

1.c First Elements of C++

Input and output

- Reading variables from the keyboard
  - ✓ **`cin`** takes input from the keyboard and stores it in a designated variable
  - ✓ **`cin`** is used together with **`>>`** (the "extraction" operator) and the variable name to gather the input value:
    - **`cin >> num1;`**
  - ✓ using more than one variable in a **`cin`** statement allows more than one value to be read at a time:
    - **`cin >> num1 >> num2;`**
    - this reads two integers from the keyboard and inputs them into variables **`num1`** and **`num2`** respectively

1.c First Elements of C++

Input and output

- Displaying messages and values to the screen
  - ✓ **`cout`** takes strings and values (expressions, variables or literals) and outputs them to the screen
  - ✓ **`cout`** is used together with **`<<`** (the "insertion" operator) and the string or value:
    - **`cout << average;`**
  - ✓ outputs can be combined into one **`cout`** statement:
    - **`cout << "The result is " << num;`**
  - ✓ **`endl`** causes a new line and flushes the output buffer:
    - **`cout << (2 + 2) << endl;`**

1.c First Elements of C++

Input and output

prompt message & read values

output result

```
#include <iostream>
using namespace std;

int main()
{
        int x, y, z;
        float avg;

        /* read data, calc and print */
        cout << "Please enter 3 numbers: ";
        cin >> x >> y >> z;

        int sum = x + y + z; // calculate
        avg = sum/3.0;       // average

        cout << endl;
        cout << "The average is: ";
        cout << avg << endl;

        return 0;
}
```

Computer Science I

CS 135

1. Introduction to Programming

a. How to Develop a Program

b. Writing Pseudocode

c. First Elements of C++

- ✓ The basics of a C++ program
- ✓ Data types
- ✓ Arithmetic operators
- ✓ Expressions
- ✓ Variables
- ✓ Type casting
- ✓ ASCII characters
- ✓ Input and output

d. Looking Under the Hood

Computer Science I

CS 135

1. Introduction to Programming

a. How to Develop a Program

b. Writing Pseudocode

c. First Elements of C++

d. Looking Under the Hood

Computer Science I

CS 135

0. Course Presentation
1. Introduction to Programming
2. **Functions I: Passing by Value**
3. **File Input/Output**
4. **Predefined Functions**
5. **If and Switch Controls**
6. **While and For Loops**
7. **Functions II: Passing by Reference**
8. **1-D and 2-D Arrays**