BBM 101 – Introduction to Programming I

Fall 2014, Lecture 3

Aykut Erdem, Erkut Erdem, Fuat Akal

Today

- **Introduction to Programming**
  - Basic Concepts
  - Developing Algorithms
  - Creating Flowcharts

- **The C Programming Language**
  - Your first C Program
  - Programming Process
  - Lexical Elements of a C Program
    - Keywords, Identifiers, Constants, Data Types, Operators
  - Standard Input and Output
  - Type Conversion and Casting

Today

- **Introduction to Programming**
  - Basic Concepts
  - Developing Algorithms
  - Creating Flowcharts

- **The C Programming Language**
  - Your first C Program
  - Programming Process
  - Lexical Elements of a C Program
    - Keywords, Identifiers, Constants, Data Types, Operators
  - Standard Input and Output
  - Type Conversion and Casting

What is a Program?

- A computer **program** is a set of instructions for a computer to follow
  - e.g. instructions to find the maximum value in a list of numbers

What is a Program?

- A computer **program** is a set of instructions for a computer to follow
  - e.g. instructions to find the maximum value in a list of numbers

- An **algorithm** is a sequence of precise instructions which leads to a solution
  - e.g. **how** to find the maximum value in a list of numbers

What is a Program?

- A computer **program** is a set of instructions for a computer to follow
  - e.g. instructions to find the maximum value in a list of numbers
- An **algorithm** is a sequence of precise instructions which leads to a solution
  - e.g. **how** to find the maximum value in a list of numbers
- **Program** is an **algorithm** expressed in a language that the computer can understand
  - e.g. The C Programming Language

What is a Program?

- A computer **program** is a set of instructions for a computer to follow
  - e.g. instructions to find the maximum value in a list of numbers
- An **algorithm** is a sequence of precise instructions which leads to a solution
  - e.g. **how** to find the maximum value in a list of numbers
- **Program** is an **algorithm** expressed in a language that the computer can understand
  - e.g. The C Programming Language
- Computer **software** is the collection of programs
  - e.g. Microsoft Office, iTunes, FireFox etc.

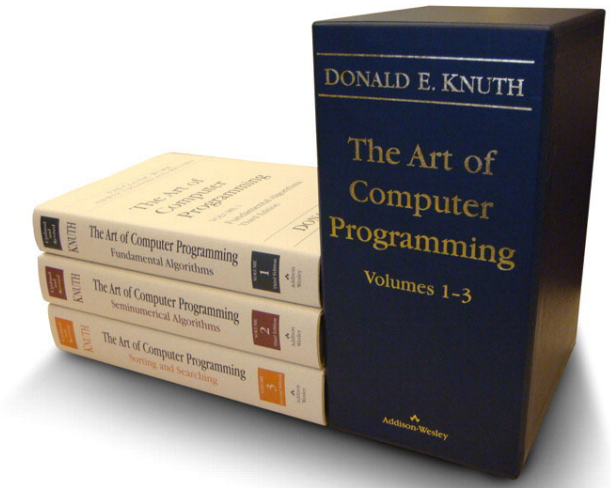
Pseudocode

- Artificial, informal language that helps us develop algorithms
  - Similar to everyday English
  - Not actually executed on computers
  - Helps us “think out” a program before writing it
  - Easy to convert into a corresponding C program
- Example:
  1. Display “Enter two integer number” message
  2. Read the 1st number from keyboard
  3. Read the 2nd number from keyboard
  4. Compute *sum* of entered numbers
  5. Print “Sum = “ + *sum*

Program Design

- Programming is a creative process
  - No complete set of rules for creating a program

- Program Design Process
  - Problem Solving Phase
    - Result is an algorithm that solves the problem
  - Implementation Phase
    - Result is the algorithm translated into a programming language



The Art of Computer Programming, Donald Knuth *(expected to be 7 volumes)*

Problem Solving Phase

- Be certain the task is completely specified
  - What is the input?
  - What information is in the output?
  - How is the output organized?
- Develop the algorithm before implementation
  - Experience shows this saves time in getting your program to run
  - Test the algorithm for correctness

Implementation Phase

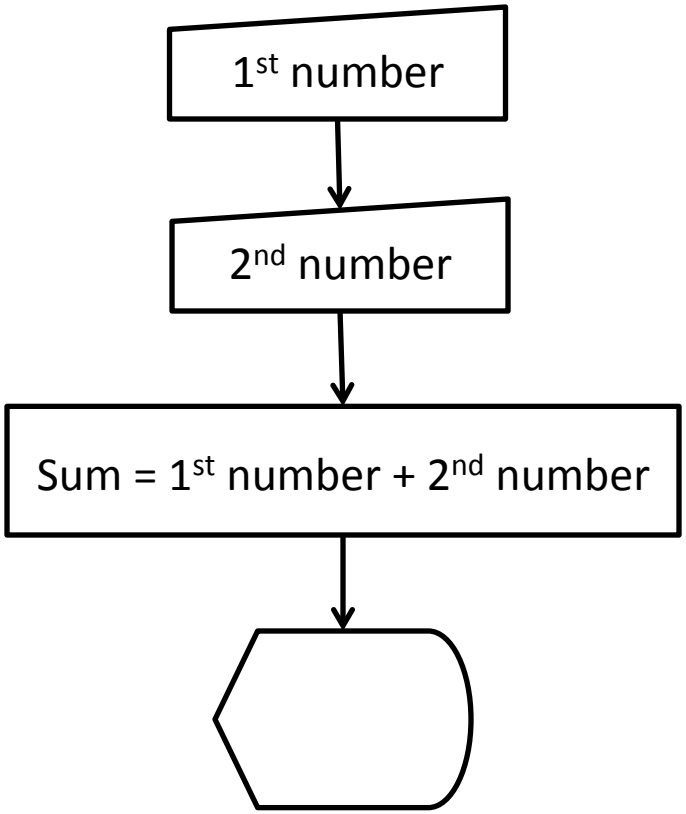
- Translate the algorithm into a programming language
  - Easier as you gain experience with the language
- Compile the source code
  - Locates errors in using the programming language
- Run the program on sample data
  - Verify correctness of results
- Results may require modification of the algorithm and program

Software (*Bigger Programs*) Development

- Problem Definition
- Analysis
  - Analyze the problem
  - Define input, output and variables
- Design
  - Design the algorithm
- Implementation
  - Coding the algorithm
- Testing
  - Test and verify the correctness of the program
- Maintenance
  - Fix bugs and add new features

Flowcharts

- A **flowchart** is a type of diagram, that represents an algorithm or process



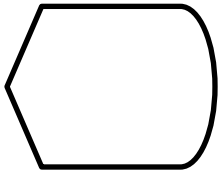
Basic Flowchart Symbols 1/2

Start / Stop

Input

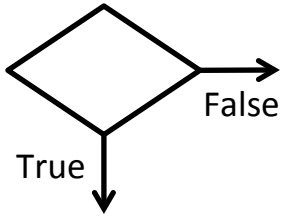
Sequence

Output

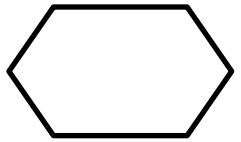


Basic Flowchart Symbols 2/2

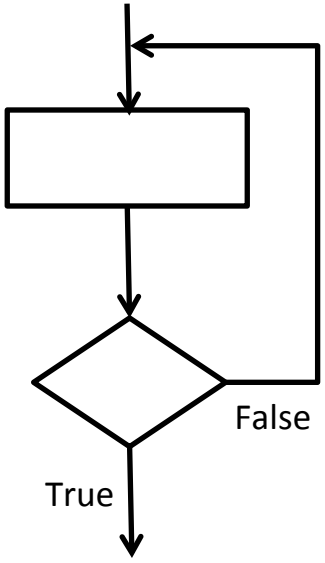
Decision



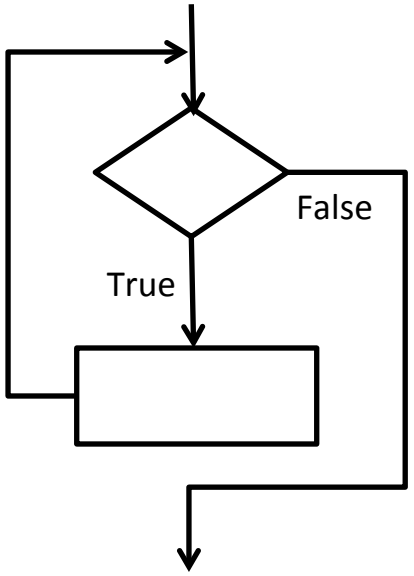
Repeat



Repeat Loop



While Loop



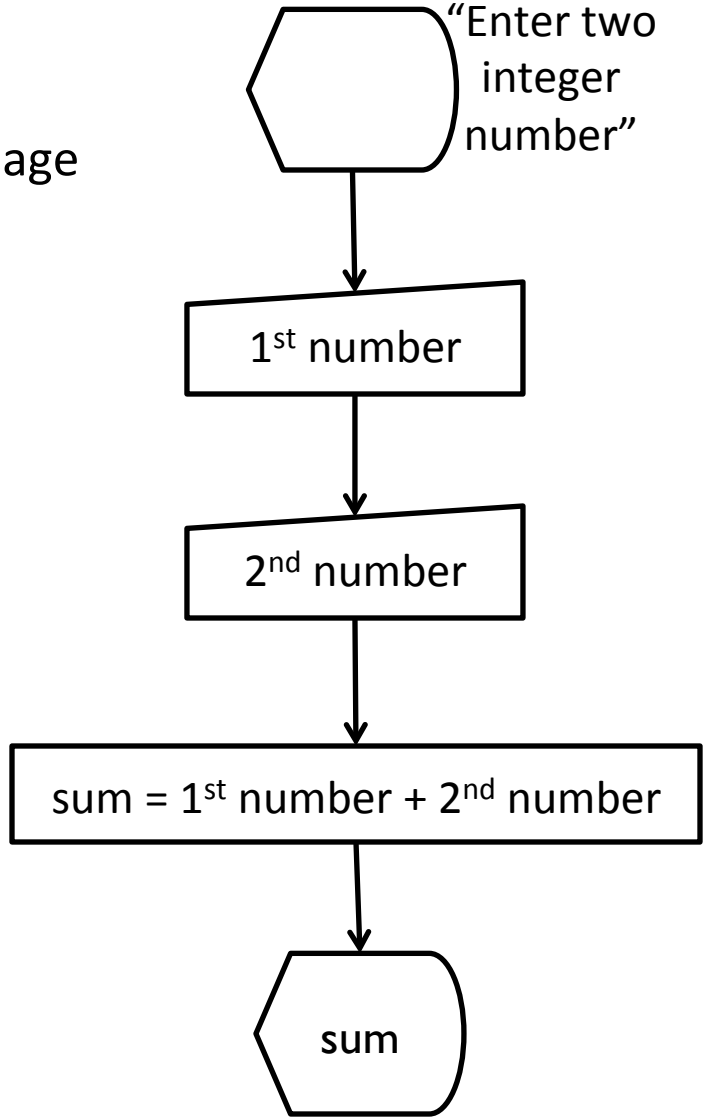
Example 1: Algorithm to find and display the sum of two integers entered via keyboard

Example 1: Algorithm to find and display the sum of two integers entered via keyboard

- Algorithm
  1. Display “Enter two integer number” message
  2. Read the 1st number from keyboard
  3. Read the 2nd number from keyboard
  4. Compute *sum* of entered numbers
  5. Print “Sum = “ + *sum*

Example 1: Algorithm to find and display the sum of two integers entered via keyboard

- Algorithm
  1. Display “Enter two integer number” message
  2. Read the 1st number from keyboard
  3. Read the 2nd number from keyboard
  4. Compute *sum* of entered numbers
  5. Print “Sum = “ + *sum*



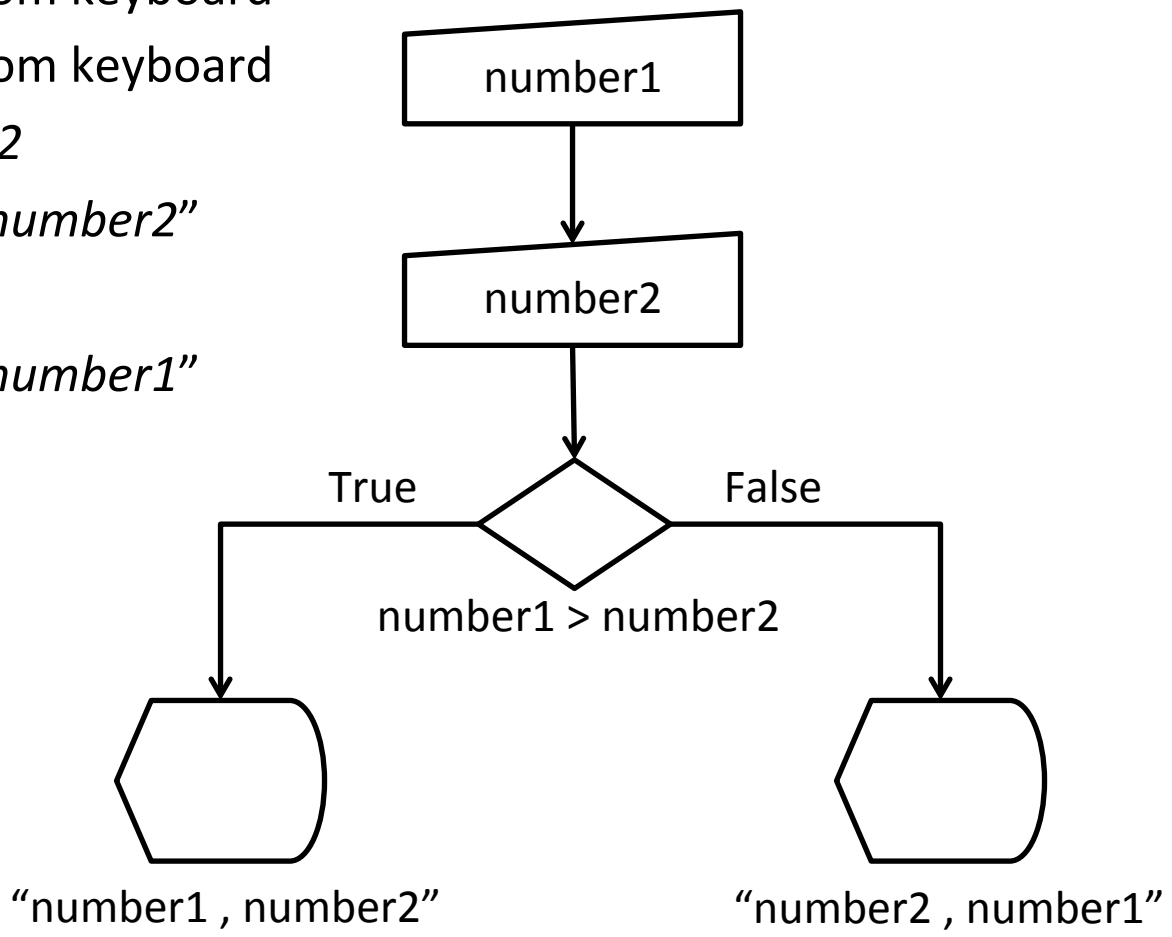
Example 2: Algorithm to display two integers entered via keyboard in descending order

Example 2: Algorithm to display two integers entered via keyboard in descending order

- Algorithm
  1. Read the *number1* from keyboard
  2. Read the *number2* from keyboard
  3. If *number1* > *number2*
     - Print "*number1* > *number2*"
  4. otherwise
     - Print "*number2* > *number1*"

Example 2: Algorithm to display two integers entered via keyboard in descending order

- Algorithm
  1. Read the *number1* from keyboard
  2. Read the *number2* from keyboard
  3. If *number1* > *number2*
     - Print "*number1* > *number2*"
  4. otherwise
     - Print "*number2* > *number1*"



Example 3: Algorithm to display three integers entered via keyboard in ascending order

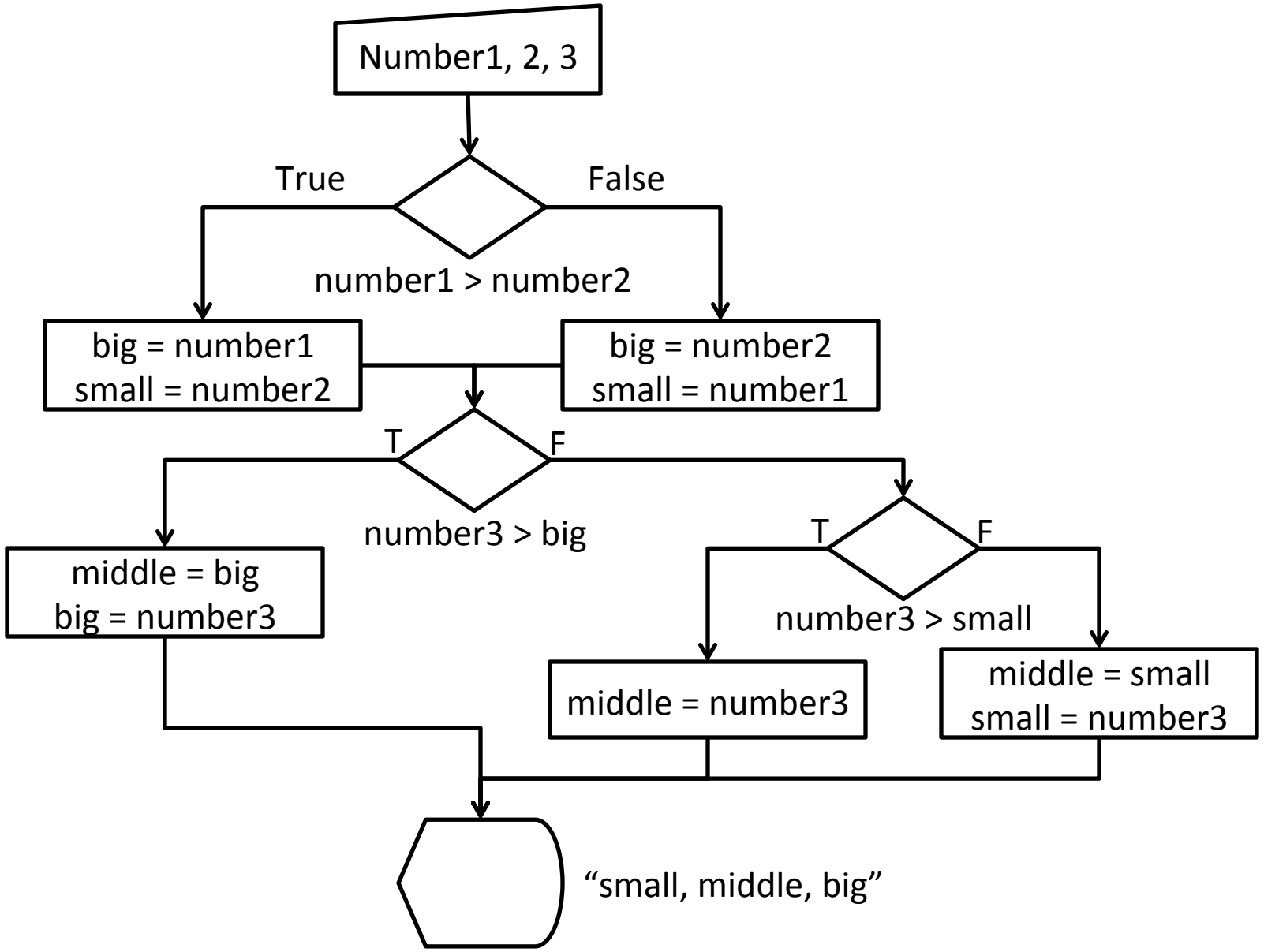
Example 3: Algorithm to display three integers entered via keyboard in ascending order

- Algorithm
  1. Read the *number1, number2 and number3* from keyboard
  2. If *number1* > *number2*
     - Big = number1
     - Small = number2
  3. Otherwise
     - Big = number2
     - Small = number1

Example 3: Algorithm to display three integers entered via keyboard in ascending order

- Algorithm
  1. Read the *number1, number2 and number3* from keyboard
  2. If *number1 > number2*
     - big = number1
     - small = number2
  3. Otherwise
     - big = number2
     - small = number1
  4. If number3 > big
     - middle = big
     - big= number3
  5. Otherwise
     - If number3 > small
       - middle = number3
     - Otherwise
       - middle = small
       - small = number3
  6. Display small, middle, big

Example 3: Algorithm to display three integers entered via keyboard in ascending order



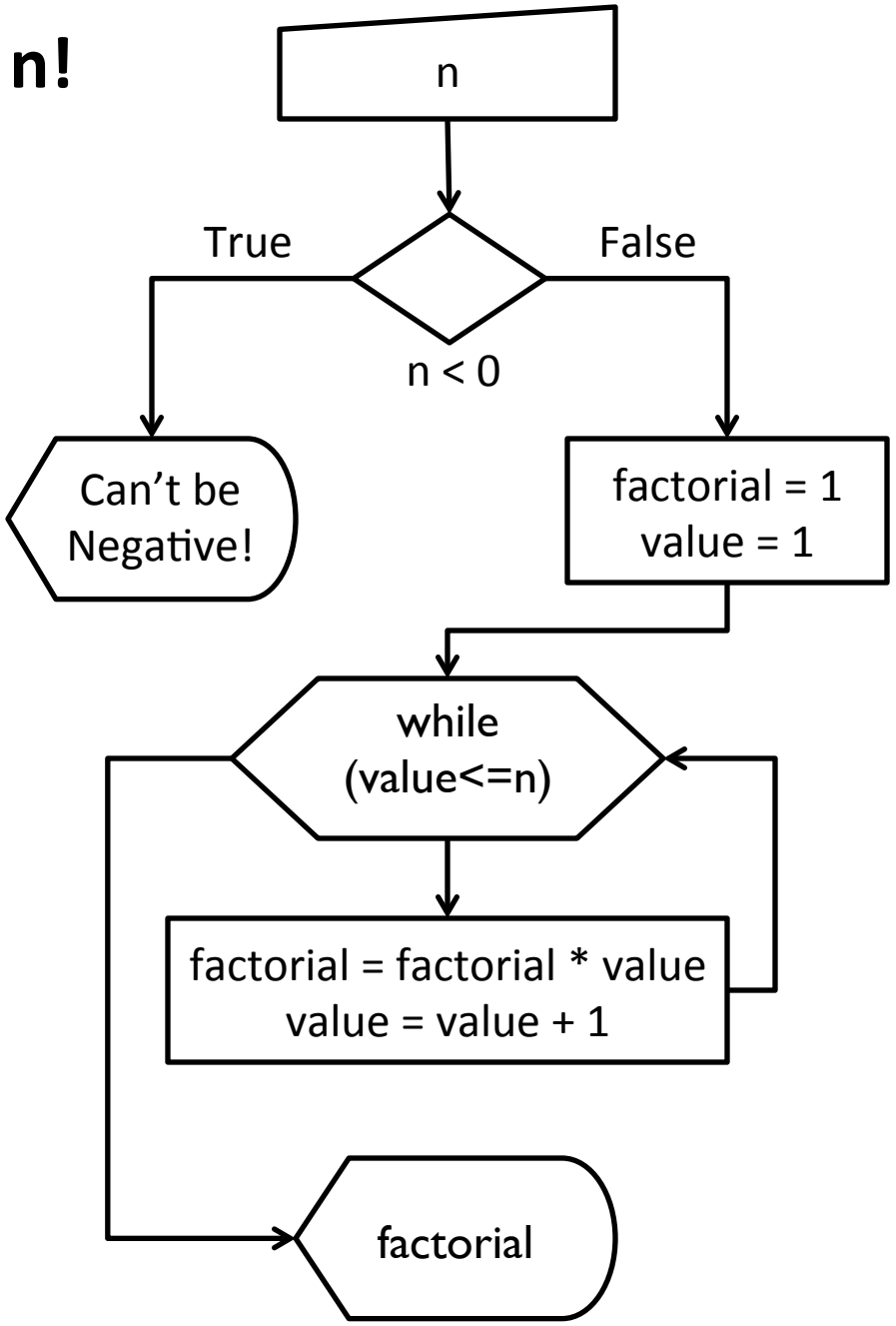
Example 4: Algorithm to find n!

Example 4: Algorithm to find n!

- Algorithm
  1. Read *n* from keyboard
  2. if *n < 0*
     - Display error message
  3. else
     - factorial = 1
     - value = 1
     - while (value <= n)
       - factorial = factorial * value
       - increment value by 1
  4. Display factorial

Example 4: Algorithm to find n!

- Algorithm
  1. Read *n* from keyboard
  2. if *n < 0*
     - Display error message
  3. else
     - factorial = 1
     - value = 1
     - while (value <= n)
       - factorial = factorial * value
       - increment value by 1
  4. Display factorial



Today

- Introduction to Programming
  - Basic Concepts
  - Developing Algorithms
  - Creating Flowcharts

- **The C Programming Language**
  - Your first C Program
  - Programming Process
  - Lexical Elements of a C Program
    - Keywords, Identifiers, Constants, Data Types, Operators
  - Standard Input and Output
  - Type Conversion and Casting

Anatomy of a Typical C Program

```
#preprocessor directives

declarations
  variables
  functions

int main (void){
   declarations;
   statements;
   return value;
}
```

Your First C Program

hello.c

```
/* Welcome to BBM 101 */

#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

$>> gcc hello.c –ansi –Wall –o hello
$>> ./hello

```
Hello world!
```

Your First C Program

hello.c

```
/* Welcome to BBM 101 */

#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

/* comments */

global declarations
#include external files

main function

```
Hello world!
```

Your First C Program

hello.c

```
/* Welcome to BBM 101 */

#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

Text surrounded by /* and */ is ignored by computer

/* comments */

global declarations
#include external files

main function

```
Hello world!
```

Your First C Program

hello.c

```
/* Welcome to BBM 101 */

#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

/* comments */

global declarations
#include external files

"stdio.h" allows standard input/output operations

main function

```
Hello world!
```

Your First C Program

hello.c

```
/* Welcome to BBM 101 */

#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

/* comments */

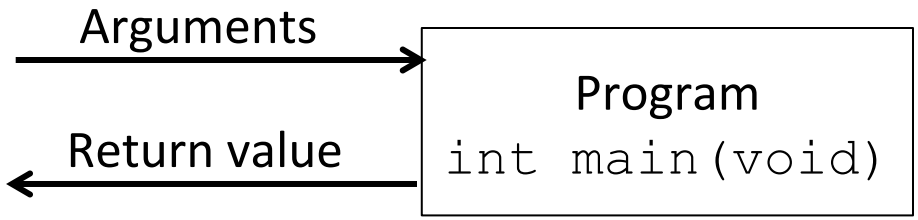
global declarations
#include external files

main function

C programs contain one or more functions, exactly one of which must be **main**

```
Hello world!
```

The `main(void)` of `hello.c`




```
int main(void)
        {
                printf("Hello world!\n");
                return 0;
        }
```

return "0" to OS:
"everything is OK"

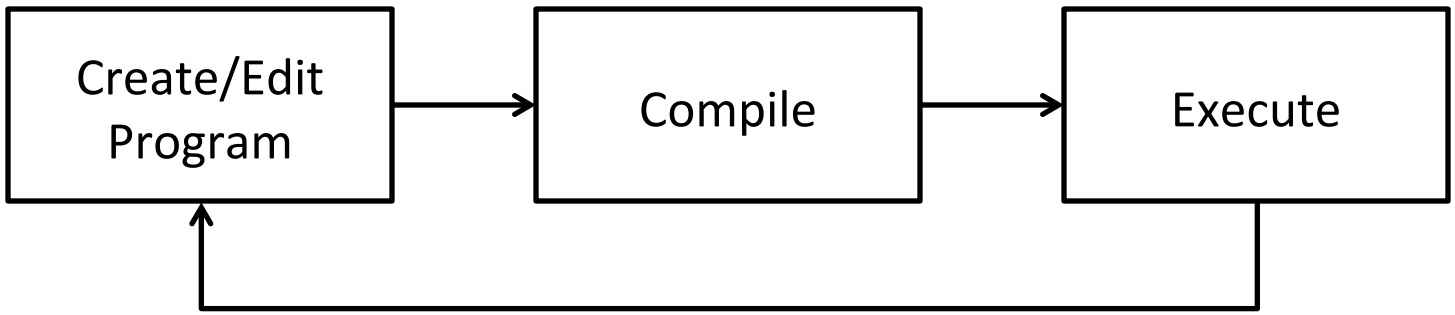
- No **arguments**.
- Returns an integer **variable**.

Slide credit: Bert Huang

C Statements

- One-line commands
- Always end in semicolon **;**
- Examples:
  - call function: `printf("hello"); /* from stdio */`
  - declare **variable**: `int x;`
  - assign variable value: `x = 123+456;`

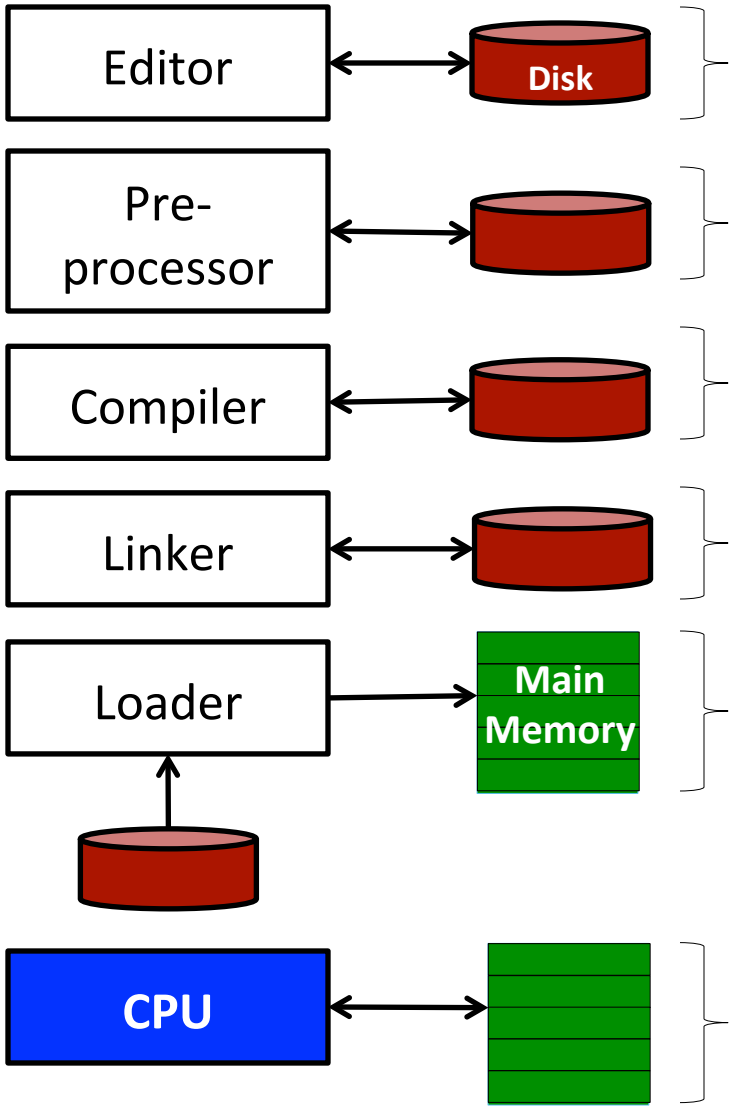
The Programming Process



"The cycle ends once the programmer is satisfied with the program, e.g., performance and correctness-wise."

"Do not forget, your program always has at least twenty bugs even after you have fixed them all ☺"

C Program Development



Create/Edit: Program is created in the editor and stored on disk.

Preprocess: Preprocessor program processes the code.

Compile: Compiler creates object code and stores it on disk.

Link: Linker links the object code with the libraries.

Load: Loader puts program in memory.

Execute: CPU takes each instruction and executes it, possibly storing new data values as the program executes.

Lexical Elements

- **Token:** The smallest element of a program that is meaningful to the compiler

- Kinds of tokens in C:
  - Keywords
  - Identifiers
  - Constants/Literals
  - Operators
  - Punctuators

Keywords

- 32 words are defined as keywords in C
- They have predefined uses and cannot be used for any other purpose in a C program

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Identifiers

- A sequence of letters, digits, and the underscore character '_' satisfying
  - identifier = c { c | d }*
  - with c = {'A',…, 'Z', 'a',…, 'z', '_' }, d = {0,…,9}, and asterisk "*" means "0 or more"
- Case-sensitive
  - e.g., *first**N**ame* and *first**n**ame* are two different identifiers.
- Identifiers are used for
  - Variable names
  - Function names
  - Macro names

Identifier Examples

- Valid identifiers
  - X
  - a1
  - _xyz_33
  - integer1
  - Double

- Invalid identifiers
  - xyz.1
  - gx^2
  - 114West
  - int ← This is a keyword
  - pi*r*r

Variables

- A variable is a location in main memory where a value is stored (just like Algebra)
- Variables must be declared before they are used
- Variable declarations must appear before executable statements
  - A syntax error is raised at compile-time if above two are violated
- Every variable has a name, a type, size and a value

Basic Data Types

- Integer (**int**)
- Character (**char**)
- Floating Point (**float**)
- Double Precision Floating Point (**double**)

- Data Type Modifiers
  - **signed** / **unsigned**
  - **short** / **long**

`int`

- 4 bytes (on Unix)
- Base-2 representation.
- need one bit for + or -
- Range: -2^{31} to 2^{31}
- Variants: `short` (2 bytes), `long` (8 bytes), `unsigned` (only non-negative)

Slide credit: Bert Huang

`char`

- 1 byte
- ASCII representation in base-2
- Range: 0-255 (lots of unused)

Slide credit: Bert Huang

`float`

- Stands for “floating decimal point”
- 4 bytes
- Similar to scientific notation: $4.288 * 10^3$
- Very different interpretation of bits than `int` and `char`.
- Range: -10^{38} to 10^{38}

Slide credit: Bert Huang

Basic Data Types

Type	Size in Bytes	Range
signed char	1	-127 to +127
unsigned char	1	0 to 255
short int	2	-32,767 to +32,767
unsigned short int	2	0 to 65535
int	4	-32,767 to +32,767
unsigned int	4	0 to 65,535
long int	8	-2,147,483,647 to +2,147,483,647
unsigned long int	8	0 to 4,294,967,295
float	4	$\sim 10^{-37}$ to $\sim 10^{38}$
double	8	$\sim 10^{-307}$ to $\sim 10^{308}$
long double	16	$\sim 10^{-4931}$ to $\sim 10^{4932}$

Declaring a Variable

- A declaration consists of a data type name followed by a list of (one or more) variables of that type

```
char c;
int myCounter;
float rate;
double trouble;
```

- A variable may be initialized in its declaration

```
char c = 'a';
int a = 220, b = 448;
float x = 1.23e-6; /*0.00000123*/
double y = 27e3; /*27,000*/
```

- Variables that are not initialized may have garbage values
- Placing a new value replaces the previous value
- Reading variables from memory does not change them

Constants

- Integer Constants
  - 0, 37, 2001
- Floating-point Constants
  - 0.8, 199.33, 1.0
- Character Constants
  - ‘a’, ‘5’, ‘+’
- String Constants
  - “a”, “Monday”
- How to define?
  - `#define PI 3.14;`
  - `const double PI = 3.14;`

- Common Escape Sequences
  - \a audible alarm
  - \b backspace
  - \n newline
  - \r carriage return
  - \t horizontal tab
  - \f form-feed
  - \\ backslash
  - \” double quote

Operators

- Arithmetic operators
  - *, /, %, +, -
- Assignment operator
  - =
- Logical operators
  - We will cover this next week in the selective-structures lecture

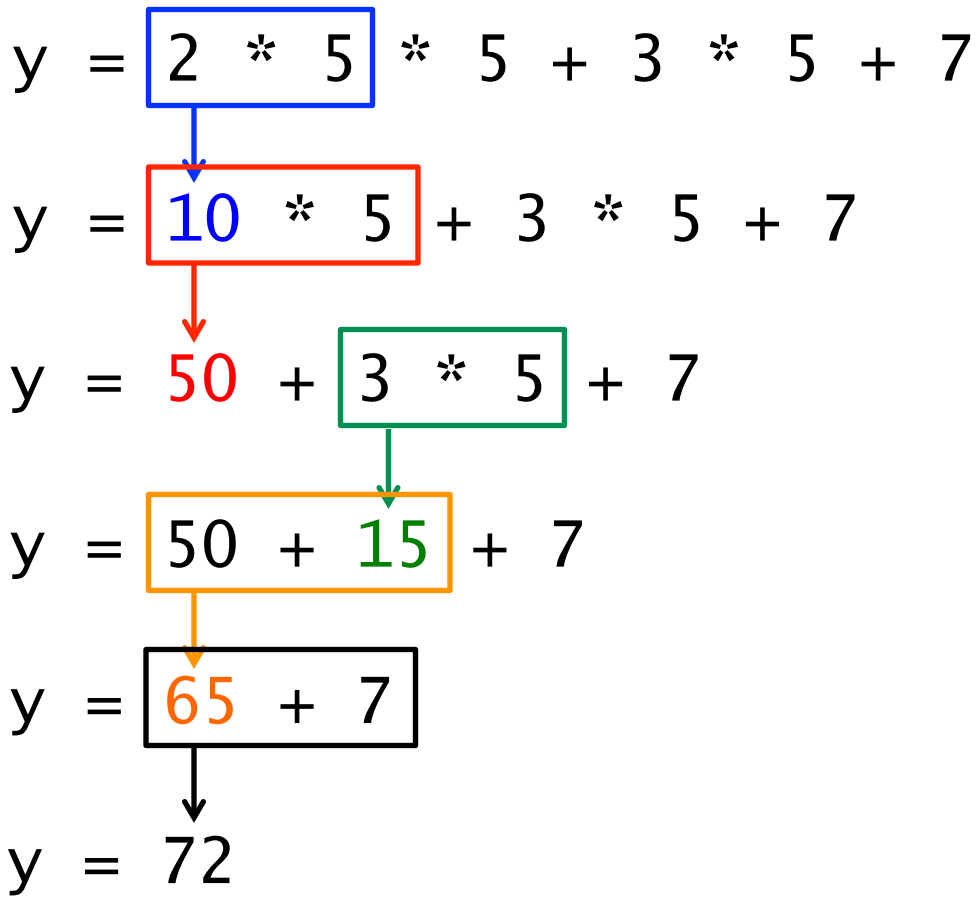
Arithmetic Operators

- For arithmetic calculations
  - Addition (+), subtraction (-), multiplication (*) and integer division (/)
  - Integer division truncates remainder
    - 7 / 5 evaluates to 1
  - Modulus operator(%) returns the remainder
    - 7 % 5 evaluates to 2
- Arithmetic operators associate left to right
- Operator precedence
  - Example: Find the average of three variables a, b and c
    - Do not use: a + b + c / 3
    - Use: (a + b + c) / 3
    - See next slide for why

Operator Precedence

Operator(s)	Operation(s)	Precedence
( )	Parentheses	Evaluated first. Innermost parentheses is evaluated in case of nested parentheses. Same level parentheses are evaluated from left to right.
*, /, %	Multiplication Division Modulus	Evaluated second. If there are several, they are evaluated from left to right.
+, -	Addition Subtraction	Evaluated last. If there are several, they are evaluated from left to right.

Operator Precedence



Left-most multiplication

Left-most multiplication

Multiplication precedes addition

Left-most addition

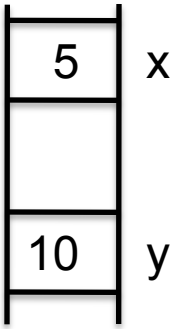
Last operation

Assignment Operator (=)

- *variable* = *expression* ;
- Expressions
  - Operations
    - `total = number1 + number 2 ;`
  - Variables
    - `temp = number1 ;`
  - Constants
    - ```
      #define PI 3.14
      circumference = 2 * PI * radius;
      ```
  - Function Calls
    - `maxValue = findMax(number1, number2) ;`
- Precedence of the assignment operator is lower than the arithmetic operators'
      ```

l-value vs. r-value

```
x = 5;
y = 10;
```



l(eft)-value
usage of y

r(ight)-value
usage of x

```
y = x ;
```

There is a memory location named *y*. This location will receive a value.

There is a memory location named *x*, where there is a value sitting, go and get me that value.

"X + 1 = 3 ;" is an invalid l-value expression.

Further Assignment Operations

- Compound Assignment
  - "`x += y ;`" equals to "`x = x + y ;`"

- Nested Assignments
  - "`x = y = z = 0 ;`" equals to "`x = (y = (z = 0)) ;`"
  - "`x -= y = z ;`" equals to "`x -= (y = z) ;`"
  - "`x = y += z ;`" equals to "`x = (y += z) ;`"

Increment/Decrement Operators

- Post-increment/-decrement
  - Use the value then increase/decrease
  - Notation: “i++” or “i--”
  - ```
    i = 5;
    j = (i++) * 2;   → i = 6 and j = 10
    ```
- Pre-increment/-decrement
  - Increase/Decrease the value then use
  - Notation: “++i” or “--i”
  - ```
    i = 5;
    j = (++i) * 2;   → i = 6 and j = 12
    ```
- Invalid Usage Examples
  - `++(i-3)`
  - `++(++i)`
  - `2 * i++ -i` → ambiguous, compiler-dependent

printf

- `printf("formatted text",exp1,exp2,...);`

- Use placeholders for variables:

```
%d int
%f float
%c char
```

- Examples:
  - `printf("Hello world!\n");`
  - `printf("%d plus %d is %d\n", x, y, x+y);`

Slide credit: Bert Huang

Increment/Decrement Operators

```
int main(void)
{
   int a = 0 , b = 0, c = 0;

   a = ++b + ++c:
   printf("\n%d %d %d", a,b,c);

   a = b++ + c++:
   printf("\n%d %d %d", a,b,c);

   a = ++b + c++:
   printf("\n%d %d %d", a,b,c);

   a = b-- + --c:
   printf("\n%d %d %d", a,b,c);

   return 0;
}
```

```
2 1 1
```

```
2 2 2
```

```
5 3 3
```

```
5 2 2
```

scanf

- `scanf("formatted text",&var1, &var2,...);`

  `%c` a single character is expected in the input

  `%d` an integer is expected in the input

  `%f` a floating point is expected in the input

- Each argument must be a pointer to the variable where the results of input are to be stored.

Standard Input and Output Example

```
#include<stdio.h>
int main(void)
{
  float principal, rate, interest;
  int years;
  printf("principal, rate, and years? ");

  scanf("%f %f %d", &principal, &rate, &years);

  rate /= 100;
  interest = principal * rate * years;
  printf("interest = %f\n", interest);

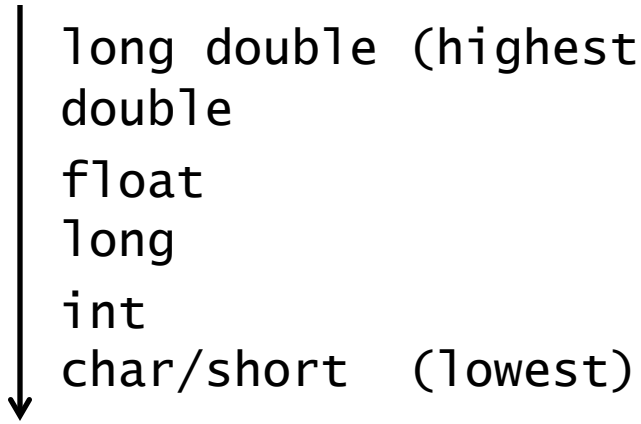
  return 0;
}
```

Expects two float and one integer numbers as input (from keyboard)

Prints a float number (on to the screen)

Type Conversion and Casting

- If operands are of mixed data types,
  the compiler will convert one operand to agree with the other
  using the following hierarchy structure:



Implicit Casting

- Done automatically by the compiler whenever data from different types is intermixed

- ```
  int i;
  double x = 17.7;
  i = x;
  ```

  `i = 17`

- ```
  float x;
  int i = 17;
  x = i;
  ```

  `x = 17.0`

Explicit Casting

```
int total_score = 333, num_students = 4;
float average;

average=total_score/num_students;
printf("Average score (no casting) is %.2f\n", average);

average=(float)total_score/(float)num_students;
printf("Average score (with casting) is %.2f\n", average);
```

```
Average score (no casting) is 83.00
Average score (with casting) is 83.25
```

Effects of Casting

- Casting a float as an int causes truncation

```
float a = 3.1;
int x = (int) a;          /* x is now 3 */
```

- Be careful with math:

```
float a, b, c;
int x = 2, y = 3;
a = x/y;                  /* what happens here? */
b = (float)x/y;
c = (float)x / (float)y;
```

Slide credit: Bert Huang

Summary

- **Introduction to Programming**
  - Basic Concepts
  - Developing Algorithms
  - Creating Flowcharts

- **The C Programming Language**
  - Your first C Program
  - Programming Process
  - Lexical Elements of a C Program
    - Keywords, Identifiers, Constants, Data Types, Operators
  - Standard Input and Output
  - Type Conversion and Casting

Next week

- **Conditional Branching**
  - **Logical** Expressions
  - **if** and **If-else** statements
  - **switch** statement
  - **goto** statement