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

- **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.

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

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 1\textsuperscript{st} number from keyboard  
  3. Read the 2\textsuperscript{nd} 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

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

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

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 2/2

- Decision
- Repeat
- Repeat Loop
- While Loop

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

$$\text{Sum} = 1^{\text{st}} \text{ number} + 2^{\text{nd}} \text{ number}$$
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 1\textsuperscript{st} number from keyboard
  3. Read the 2\textsuperscript{nd} number from keyboard
  4. Compute \textit{sum} of entered numbers
  5. Print “Sum = ” + \textit{sum}

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

- Algorithm
  1. Read the \textit{number1} from keyboard
  2. Read the \textit{number2} from keyboard
  3. If \textit{number1} > \textit{number2}
     - Print “\textit{number1} > \textit{number2}”
  4. otherwise
     - Print “\textit{number2} > \textit{number1}”
Example 2: Algorithm to display two integers entered via keyboard in descending order

- Algorithm
  1. Read the \textit{number1} from keyboard
  2. Read the \textit{number2} from keyboard
  3. If \textit{number1} > \textit{number2}
     - Print “\textit{number1} > \textit{number2}”
  4. Otherwise
     - Print “\textit{number2} > \textit{number1}”

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

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

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

- Algorithm
  1. Read the \textit{number1}, \textit{number2} and \textit{number3} from keyboard
  2. If \textit{number1} > \textit{number2}
     - \textit{big} = \textit{number1}
     - \textit{small} = \textit{number2}
  3. Otherwise
     - \textit{big} = \textit{number2}
     - \textit{small} = \textit{number1}
  4. If \textit{number3} > \textit{big}
     - \textit{middle} = \textit{number3}
  5. Otherwise
     - \textit{middle} = \textit{small}
     - \textit{small} = \textit{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 3:** Algorithm to display three integers entered via keyboard in ascending order

1. **Number1, 2, 3**
2. **True**
   - **number1 > number2**
     - **big = number1**
     - **small = number2**
     - **middle = big**
3. **False**
   - **number3 > big**
     - **big = number2**
     - **small = number1**
     - **middle = small**
4. **“small, middle, big”**

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**

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**

Can’t be Negative!

while (value <= n)

factorial = factorial * value

value = value + 1

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

Your First C Program

```c
/* Welcome to BBM 101 */
#include <stdio.h>

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

```bash
$ gcc hello.c --ansi --Wall -o hello
$ ./hello
Hello world!
```

Anatomy of a Typical C Program

```c
#include <stdio.h>

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

Your First C Program

```c
/* Welcome to BBM 101 */
#include <stdio.h>

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

Hello world!
Your First C Program

`hello.c`

/* Welcome to BBM 101 */
#include <stdio.h>

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

Hello world!

Your First C Program

`hello.c`

/* Welcome to BBM 101 */
#include <stdio.h>

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

Hello world!

Your First C Program

`hello.c`

/* Welcome to BBM 101 */
#include <stdio.h>

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

Hello world!

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

- No arguments.
- Returns an integer variable.

Program | Arguments | Return value
---------|-----------|---------------
int main(void)
{
    printf("Hello world!\n");
    return 0;
}
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;`

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.

The Programming Process

- Create/Edit Program
- Compile
- Execute

“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 😊”

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

<table>
<thead>
<tr>
<th>auto</th>
<th>double</th>
<th>int</th>
<th>struct</th>
</tr>
</thead>
<tbody>
<tr>
<td>break</td>
<td>else</td>
<td>long</td>
<td>switch</td>
</tr>
<tr>
<td>case</td>
<td>enum</td>
<td>register</td>
<td>typedef</td>
</tr>
<tr>
<td>char</td>
<td>extern</td>
<td>return</td>
<td>union</td>
</tr>
<tr>
<td>const</td>
<td>float</td>
<td>short</td>
<td>unsigned</td>
</tr>
<tr>
<td>continue</td>
<td>for</td>
<td>signed</td>
<td>void</td>
</tr>
<tr>
<td>default</td>
<td>goto</td>
<td>sizeof</td>
<td>volatile</td>
</tr>
<tr>
<td>do</td>
<td>if</td>
<td>static</td>
<td>while</td>
</tr>
</tbody>
</table>

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., `firstName` and `firstname` are two different identifiers.

- Identifiers are used for  
  - Variable names  
  - Function names  
  - Macro names

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

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

This is a keyword
Basic Data Types

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

Data Type Modifiers
- signed / unsigned
- short / long

Slide credit: Bert Huang

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

float

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

Slide credit: Bert Huang
Basic Data Types

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

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;

Operators

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

Common Escape Sequences

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

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

Assignment Operator (=)

- variable = expression ;

- Expressions
  - Operations
    - total = number1 + number2 ;
  - 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

\[ \begin{align*}
  x &= 5; \\
y &= 10;
\end{align*} \]

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.

Increment/Decrement Operators

- **Post-increment/decrement**
  - Use the value then increase/decrease
  - Notation: \( i++ \) or \( i-- \)
  - \( i = 5; \)
    - \( j = (i++) \times 2; \) \( \rightarrow i = 6 \) and \( j = 10 \)

- **Pre-increment/decrement**
  - Increase/Decrease the value then use
  - Notation: \( ++i \) or \( --i \)
  - \( i = 5; \)
    - \( j = (++i) \times 2; \) \( \rightarrow i = 6 \) and \( j = 12 \)

- **Invalid Usage Examples**
  - \( ++(i-3) \)
  - \( ++(++i) \)
  - \( 2 \times i++ -i \) \( \rightarrow \) ambiguous, compiler-dependent

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 \( x = (y += z); \)

printf

- printf(“formatted text”, exp1, exp2,...);
- Use placeholders for variables:
  - \%d int
  - \%f float
  - \%c char

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

Slide credit: Bert Huang
Increment/Decrement Operators

```c
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;
}
```

Standard Input and Output Example

```c
#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;
}
```

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.

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:

```
long double (highest)
double
float
long
int
char/short (lowest)
```
Implicit Casting

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

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

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

Effects of Casting

- Casting a float as an int causes truncation
  ```c
  float a = 3.1;
  int x = (int) a; /* x is now 3 */
  ```

- Be careful with math:
  ```c
  float a, b, c;
  int x = 2, y = 3;
a = x/y; /* what happens here? */
b = (float)x/y;
c = (float)x / (float)y;
  ```

Explicit Casting

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

average = total_score / (float) 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

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