CSCI 200: Foundational Programming Concepts & Design

Data Types

Declaring a Variable

- Recall, computers are dumb
  - Variables need two things

1. **Data type**
   - Why? Computer needs to know how much memory it needs ahead of time
     - Why is memory needed? To store a value
2. **Identifier**
   - Why? We need to know how to access the value in memory

Data Types

- int `-3  0  1`
  - integers aka whole number
- float / double `-3.92f  0.44f / 2.718  3.141`
  - floating point numbers aka decimal numbers
- char 'a' 't' '6'
  - a single character: any letter or number
- bool true 1 false 0
  - true or false

Variables & Memory

- Identifier points to memory address where value is stored

- When you reference a variable, computer looks up in memory the value at the corresponding address

Static Declarations

- Need to declare data type up front so computer can allocate enough memory
- Data types take different amount of memory

Data Type	Size	Range	
`bool`	8 bits / 1 byte*	0 to 1	0 to 1
`char`	8 bits / 1 byte	-2^7 to $+2^7-1$	-128 to +127
`int`	32 bits / 4 bytes	-2^{31} to $+2^{31}-1$	-2,147,483,648 to +2,147,483,647
`float`	32 bits / 4 bytes	±1.18e-38 to ±3.4e38	~7 digits precision
`double`	64 bits / 8 bytes	±2.23e-308 to ±1.80e308	~16 digits precision

Integer Size Modifiers

- `short int`
- `long int`
- `long long int`
  - Uses less or more memory

Data Type	Size	Range	
`short int`	16 bits / 2 bytes	-2^{15} to $+2^{15}-1$	-32,678 to +32,677
`int`	32 bits / 4 bytes	-2^{31} to $+2^{31}-1$	-2,147,483,648 to +2,147,483,647
`long int`	32 bits / 4 bytes	-2^{31} to $+2^{31}-1$	-2,147,483,648 to +2,147,483,647
`long long int`	64 bits / 8 bytes	-2^{63} to $+2^{63}-1$	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807

Assigning A Value To A Variable

- Use = to assign a value to a variable
  - Assignment (=) is not equality (==)
- General form

```
identifier = expression;
```

- Examples

```
double sum = 0.0;

int length;
length = 12;
```

```
int x = 5;
x = x + 1;

int x, y = 4;
x = y;
```

More Assignment Operators

- Compound Operators

Operator	Example	Equivalent Statement
+=	x += y;	x = x + y;
-=	x -= y;	x = x - y;
*=	x *= y;	x = x * y;
/=	x /= y;	x = x / y;
%=	x %= y;	x = x % y;

Negation

- Can either multiply by -1

```
int x, y(3);

x = -1 * y;
```

- Or simply negate variable

```
int x, y(3);

x = -y;
```

Precedence Table

Category	Precedence	Operator	Associativity
Parenthesis	1	( )	Innermost First
Unary Operators	2	+a -a	Right to Left
Binary Operators	3	a*b a/b a%b	Left to Right
	4	a+b a-b	
Assignment Operators	5	a=b a+=b a-=b a*=b a/=b a%=b	Right to Left

More Powerful Math

- Include the standard math library

```
#include <cmath>
```

- Gives you access to math functions
  - sqrt(x)
  - pow(x, y)
  - log(x)
  - sin(x)
  - And many more!