Chapter 2

Variables, Expressions, and Assignments

Warmup!

Write a program that asks a user to enter how many classes they are taking this semester. Then output the total amount of hours they are enrolled in.

Assume each class is 3 hours

Variables

- **Variable** - a location in computer's memory where a value can be stored for use by a program. A variable declaration is made up of:
  - Variable ***name (identifier)***
  - Variable ***type***
- The Variable name is an ***alias*** to a variables memory address (or location in memory)

```
int y{10};
```

example: warmup.cpp

Variables - Data Types

- ***Data Type*** - an attribute of a variable or data that tells the compiler the size of the data and the type of the data.
- ***Primitive Data Types -*** built in data types that a can be used directly by the user in C++ with no additional libraries. The simplest form of a data type.

Data Type	Description
int	Represents an integer. 4 bytes
char	Represents a single character. 1 byte
bool	Stores *true* or *false*
float	Single precision floating point values. 4 bytes
double	Double Floating Point. 8 bytes
void	Represents a valueless entity. Return type void

Modifiers and ***sizeof()***

- ***Modifiers -*** used with primitive data types to modify size of data. short and long can be used as data types by themselves.

```
signed
unsigned
short
long
```

- **`sizeof()`** - returns the size of a datatype. Accepts the datatype keyword or variable.

example: sizeof.cpp
datatype_sizeof.cpp

Character Data Type

- A ***char*** stores a single ASCII character.
- A character in C++ is represented by an ASCII code or integer.
  - `A` - 65 (ascii)
  - `a` - 97 (ascii)
- ***A != a***
- Special / Escaped Characters
  - `\n`, `\t`, `\0`, etc..
- Full ASCII Table can be found here: https://en.cppreference.com/w/cpp/language/ascii

```
char grade{‘A’};
grade = 'B';
```

The ***auto*** data type

- With ***auto,*** the compiler will determine the type based on the initialized value.

```
auto x; // Error: variable isn't initialized

auto y = 5; // Compiler sets type to int

auto z = 'a'; // Compiler sets type to char
```

Be careful with ***auto…*** sometimes the compiler will determine types differently from how you might expect:

```
auto s = "SMU"; // Compiler sets type to const
                        char* NOT string object
```

example: auto.cpp

Identifiers (Variable Names)

- ***Identifiers*** are a name created by a programmer for an item such as a variable or function. The C++ standard for identifiers is as follows:
  - Consist of sequence of letters, underscores and digits
  - Start with letter or underscore
  - Variables are camelcase or snake_case (**pick one and stay consistent!**)
  - Use **full names** instead of abbreviations (Ex: sumVal - sum Value? Summary validation? …)

```
numberOfCats
percentIncomeTax
startTime
~~1timeCharge~~
~~$money~~
```

Table 2.3.1: C++ reserved words / keywords.

alignas (since C++11)	decltype (since C++11)	namespace	struct
alignof (since C++11)	default	new	switch
and	delete	noexcept (since C++11)	template
and_eq	do	not	this
asm	double	not_eq	thread_local (since C++11)
auto	dynamic_cast	nullptr (since C++11)	throw
bitand	else	operator	true
bitor	enum	or	try
bool	explicit	or_eq	typedef
break	export	private	typeid
case	extern	protected	typename
catch	false	public	union
char	float	register	unsigned
char16_t (since C++11)	for	reinterpret_cast	using
char32_t (since C++11)	friend	return	virtual
class	goto	short	void
compl	if	signed	volatile
const	inline	sizeof	wchar_t
constexpr (since C++11)	int	static	while
const_cast	long	static_assert (since C++11)	xor
continue	mutable	static_cas	xor_eq

Source: http://en.cppreference.com/w/cpp/keyword.

Variable Declaration / Initialization

```
int number1;
int number2 = 10;
int number3{20};
```

- Variable Declaration
- Variable Declaration and Initialization
- Variable List Initialization ***(Preferred Method)***

Tip: Even though it is not necessary to initialize a variable when declaring it, doing so will help avoid potential errors

Variables vs Literals

- ***Variables*** have a memory address, ***literals*** do not.
  - `2` – An integer literal
  - `'a'` – a character literal
  - `"Elroy"` – a string literal

- A ***literal*** typically is used as a value that is stored inside of a variable.

Constant Variables

- A ***constant*** variable is a variable who's value does not change throughout the program
- A compiler will throw an error if an attempt is made to update a constant variable.
- Identity a global ***const*** by using ALL CAPS

```
const double MINUTES_PER_HOUR = 60;

MINUTES_PER_HOUR = 70; // Error
```

Operators and Expressions

- ***Expression*** - combination of items, variables, literals, operators and parentheses that evaluates to a value.

```
2 + (x * (y - 10))
```

- Expressions follow the same precedence rule as standard mathematics
- Best Practice to include a space between each literal / variable and the operator (excluding parenthesis)

examples: polynomial.cpp

Division and Modulo Operator

Integer Division where both the numerator and denominator are integers yields an integer quotient.

- Ex: `17 / 5` results in `3`
- Any fractional remainder is truncated from the integer.

Modulo Operator can be used to get the remainder of a division operation. It can also be used for other purposes, like determining the parity of a value.

example: modulo.cpp

The Assignment Operator

The ***assignment*** operator (i.e **=**) is used to assign a value to a variable.

- DO NOT think of the assignment operator as an equality!
- Typically an assignment operator is used with an expression that evaluates to a value
- The left hand side of the assignment and the right hand assignment have to be of the same type.

The Math Library

- The ***Math Library*** provided by `cmath` allows a programmer to utilize over 20 mathematical functions that a are not supported with the built in operators in C++.

- Utilizing a Library is common practice in any programming language. Why re-invent the wheel when others have done it for you?

- Use the STD Math Library by adding `#include <cmath>` at the top of you program

Math Functions Example

Implement a program that calculates the distance between two points.

$$Distance = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Table 2.10.1: A few common math functions from the math library.

Function	Behavior	Example
sqrt(x)	Square root of x	sqrt(9.0) evaluates to 3.0.
pow(x, y)	Power: x^y	pow(6.0, 2.0) evaluates to 36.0.
fabs(x)	Absolute value of x	fabs(-99.5) evaluates to 99.5.

examples: math.cpp

Random Numbers

Use the ***cstdlib*** library to import the `rand()` function which returns a random number between 0 and RAND_MAX.

- You can scale the range of random numbers by using the modulo operator (%)
- The rand() function does not generate ***truly*** random numbers. It relies on a ***seed*** to generate a random number

```
int random = rand();
int random = rand() % 10;

srand(time(NULL));
```

examples: random.cpp

Overflow

Overflow occurs when the value held inside a variable exceeds the amount of space allocated for that a variable in memory.

- It is important in C++ to allocate enough memory for your variables so you do not get ***unexpected behavior*** or a ***runtime error.***

```
int value = 9999999999; // overflow
```

examples: overflow.cpp

Type Casting

IMPLICIT CONVERSION

Casting that does not require an operator. The type is a cast automatically when assigned to a compatible type.

```
double numA = 10.5;
int numB = numA; //numB = 10
```

```
int numA = 10;
double quarter = numA * 0.25; //numB = 2.5
```

EXPLICIT CONVERSION

Using an operator or function to explicitly cast a data type to a different data type.

```
static_cast<type>
```

```
int numPlayers = 10;
int totalScore = 23;
int avg = numPlayers / totalScore;
cout << avg << endl;

double avg_decimal = (double)(numPlayers / totalScore);
cout << avg_decimal << endl;
```

examples: casting.cpp
note_frequencies.cpp

Other Data Types

Intro to Strings and Cstrings

Strings

Strings are a data type that represents a sequence of characters.

- ***String literals*** are surrounded in double quotes.

  Ex:

  “Elroy was meowing loudly”
  “Computer”

- NOTE: A string is ***not*** a primitive data type. A string is an ***object*** that is provided by the string library.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  string name = "Erik Gabrielsen";
  cout << "My name is " << name << endl;

  return 0;
}
```

Strings Continued...

- A string is initialized to an empty string (“”) by default:

```
string name;
```

- We can re-assign a string to one of a larger, or smaller size. The library is in charge of allocating the correct amount of memory.
- Strings may include whitespace characters. i.e “ “, “\t”, “\n”
- Read in strings using `getline()` and `cin`

examples: getline.cpp

C-Strings

Cstrings are an array of characters that are always terminated by the null character i.e. `\0`

- Initializing a cstring requires you to know how many characters are in the string at ***compile time.***

```
char greeting[6] = "hello";
```

- We can reference individual characters of a cstring by providing an index.

```
cout << greeting[0]; // 'h'
```

NOTE: Remember that indexes in C++ and Java begin with ***0***

examples: cstring.cpp