Variables

Objectives:

- By the end of this section you should:
  - Understand what the variables are and why they are used.
  - Use C++ built in data types to create program variables.
  - Apply C++ syntax rules to declare variables, initialize them.
  - Understand memory allocation process.
  - Apply C++ syntax rules to read user input using `cin`.
  - Understand assignment statements
  - Understand the scope of variables
  - Differentiate between local and global variables
  - Use C++ arithmetic and logical operators to create expressions

Recall:

- Write a program that displays the sum of two integers entered at the keyboard:
  - Define the problem precisely.
  - Write the psudocode that will solve the problem
  - Use an Editor to create source code in C++.
  - Use the compiler to
    - Check that source code obeys the language rules.
  - If no errors: Execute your program.

Variables

- In order to define places in memory where the three values will be stored, we need to defined three variables for number1, number2 and sum.

Variables

- Variable:
  - Location on computer's memory to store data then use and change its value in a program.
- Name(Identifier)
  - Series of letters, digits, underscores
  - Not a keyword(int, float, double char, void, return main)
  - Start with a letter
  - Case sensitive
  - Meaningful
- Type:
  - Programmer defined
  - Built-in

What is a data type?

- When we wish to store data in a C++ program, we have to tell the compiler which type of data we want to store.
- The data type will have characteristics such as:
  - The range of values that can be stored.
  - and the operations that can be performed on variables of that type.

C++ Built-in Data Types

- Called fundamental types or primitives types:
  - Numerical (integer and floating point)
  - Character
  - Logical (Boolean)

C++ Built-in Data Types

Type	size
bool	1 byte
unsigned short int	2 bytes
short int	2 bytes
unsigned long int	4 bytes
long int	4 bytes
int	2 bytes
char	1 byte
float	4 bytes
double	8 bytes

bool Data type

- Has two values (true) and (false).
- Manipulate logical expressions.
- true and false are called logical values.
- bool, ture, and fasle are reserved words.
- For example:

```
bool isEven = false;
bool keyFound = true;
```

char Data Type

- Used for characters: letters, digits, and special symbols.
- Each character is enclosed in single quotes.
- Some of the values belonging to char data type are:

  ‘A’,’a’,’0’,’*’,’+’,’$’,’&’.
- A blank space is a character and is written ‘ ‘, with a space left between the single quotes.

int Data Type

- The integer type is used for storing whole numbers. We can use signed, unsigned or plain integer values as follows:

```
signed int index = 4182;
signed int temperature = -32;
unsigned int count = 0;
int height = 100;
int balance = -67;
```

Floating-Point Types

- Floating point types can contain decimal numbers.
  - Examples: 1.23, -.087.
- There are three sizes:
  - float (single-precision)
  - double (double-precision)
  - and long double (extended-precision).
- Examples:
  float Temp= 37.623;
  double fahrenheit = 98.415;
  long double accountBalance = 1897.23;

Variable declaration

```
Data type VarName;
```

- All variables must be declared anywhere in program with a name and data type before they used.
- Begin with a data type then variable name.
- Variables of the same type can be declared in
  - Multiple lines
  - One line separated by commas.

```
int num1;
int num2;
int num3;
```

```
int num1,num2,num3;
```

Initializing Variables

- Variables can be initialized when declared:

```
int first=13, second=10;
char ch= ‘ ‘;
double x=12.6, y=123.456;
```

- `first` and second are `int` variables with the values 13 and 10 respectively.
- `ch` is a char variable whose value is empty.
- `x` and `y` are double variables with 12.6 and 123.456 respectively.

Memory Concepts

- Variable names such as : number1, number2 and sum correspond to locations in the computer's memory.
- Every variable has four parts:
  - Type, name, size and value.
  - Example:
  - char letter='A';
    - Type? Name? Size? Value?

Using cin

- Namespace:
  - std::
    - Specifies using a name that belong to “namespace” std
    - Can be removed through use of using statement.
  - Standard input stream object.
    - std::cin
    - Connected to keyboard
    - Defined in input/output stream library <iostream>

Using `cin`

- Stream extraction operator >>
  - Value to left (left operand) inserted into right operand.
  - Waits for user to input value then press enter key
    - Examples:

      ```
      cin>>num1;
      ```
    - Inserts the standard input from keyboard into the variable `num1`.
- Prints message before `cin` statement to direct the user to take a specification called prompt.
- `cin` and `cout` facilitate interaction between user and program.

Examples:

```
#include <stream>
int main
{
    cout << "If this text",
    cout >> " appears on your display, ";
    cout << " endl;"
    cout << 'you can pat yourself on '
    << " the back!" << endl.
    return 0;
)
```

In the above program, find the errors (if any) then, give the program output.

Scope Of Variable

- The scope of variable is the portion of the program where the variable can be used.
- Scope can be:
  - Local
  - Global
- Local variables:
  - Defined within a module
  - Can be seen and used only by the module itself
  - Store temporally in memory
  - Erased when he module terminates

Scope Of Variable

- Global variables:
  - Defined outside any module.
  - Used an seen by all modules
- Variable name can be duplicated within and outside a modules
  - Differentiate between them by using unary scope resolution operator (::)

Examples:

```
int main()
{
  int i;
  char a;
}
```

i: Local variable

```
int i;
int main()
{
  char a;
}
```

i: Global variable

Unary Scope Resolution Operator

- Denoted as (::)
- Used to declare local and global variables have a same name.
  - To avoid conflicts.
- Syntax: :: variable
  - Example: y= ::x+3
- Not needed if names are different

Example

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

int count = 100;
int main()
{
   int count = 10;
   int Second_count = 50;
   cout << "Local  count = " << count << endl;
   cout << "Global count = " << ::count << endl;
   cout << "Local Second count = " << Second_count
<<endl;

   return 0;
}
```

Assignment Statement

- The assignment statement takes the form:
  - Variable= expression;
- In C++ (=) is called the assignment operator.
- Has two operands (Binary operator)
- Expression is evaluated and its values is assigned to the variable on the left side.

Assigning Data: Shorthand Notations

- Syntax:
  - Variable =variable operator expression;
    - Example: c=c+3;
  - Variable operator= expression.
    - Example: c+=3;

Assignment Operators

Assignment operator	Sample expression	Explanation	Assigns
Assume : int c=3, d=5,e=4,f=6,g=12			
+=	c+=7	c=c+7	10 to c
-=	d-=4	d=d-4	1 to d
*=	e*=5	e=e*5	20 to e
/=	f/=3	f=f/3	2 to f
%=	g%=9	g=g%9	3 to g

C++ Operators

- Data connectors within expression or equation
- Concept related
  - Operand: data that operator connects and processes
  - Resultant: answer when operation is completed
- Operator types:
  - Arithmetic: addition ‘+’ , subtraction ‘–’, modulo division ‘%’ ,… etc.
  - Relational: equal to ‘=‘ , less than ‘<’, grater than ‘>‘, … etc
  - Logical: NOT ‘!’ , AND ‘&&’, OR ‘||’

Arithmetic Operators

- All of them are binary operators

C++ Operation	C++ arithmetic operators	Algebraic expression	C++ expression
Addition	+	a+7	a+7
Subtraction	-	a-b	a-b
Multiplication	*	ab	a*b
Division	/	a/b	a/b
Modulus	%	a mod b	a%b

Arithmetic Operators

- Arithmetic expressions appear in straight line form.
- Parentheses() are used to maintain priority of manipulation.

Rules of Operator Precedence

- Operators in parentheses evaluated first
  - Nested/embedded parentheses.
    - (a * (b + c))
  - Operators in innermost pair firs.
  - If nested parentheses-applied from left to right
- Multiplication division, modulus applied next
  - Operators applied from left to right
- Addition, subtraction applied last
  - Operators applied from left to right
- Examples:
  - Algebra: $z = pr \% q + w / x - y$
  - C++: `z=p * r % q + w / x - y;`

Increase and Decrease (++, --)

- The increase operator (++) and the decrease operator (--) increase or reduce by one the value stored in a variable. They are equivalent to +=1 and to -=1, respectively.
  1. c++;
  2. c+=1;
  3. c=c+1;
- The three above expressions are all equivalent in its functionality: the three of them increase by one the value of c.

Increase and Decrease (++, --)

- In the case that the increase operator is used as a prefix (++a) the value is increased **before** the result of the expression is evaluated
  - Therefore the increased value is considered in the outer expression.
- In case that it is used as a suffix (a++) the value stored in a is increased **after** being evaluated
  - Therefore the value stored before the increase operation is evaluated in the outer expression

Example

// apply increment-decrement operators

1.

```
B=3;
A=++B; // A contains 4, B contains 4
```

2.

```
B=3;
A=B++; // A contains 3, B contains 4
```

Relational and Equality Operators

- All of them are binary operators

Algebraic equality or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operator			
>	>	a>b	A is greater than b
<	<	a<b	A is less than b
≥	>=	a>=b	A is greater than or equal to b
≤	<=	a<=b	A is less than or equal to b
Equality Operator			
=	=	a==b	A is equal to b
≠	!=	a!=b	A is not equal to b

Relational and Equality Operators

- Have the same level of precedence
- Applied from left to right
- Used with conditions
- Return the value true or false

Logical Operators

- Used to combine multiple conditions
- && is the logical AND
  - True if both conditions are true
  - Example: age>=50 $$ gender==‘f’
- || is the logical OR
  - True if either of conditions is true
  - Example: age>=50 || hasDiabetes ==‘true’

Logical Operators

- ! (logical NOT, logical negation)
  - Return true when its condition is false and vice versa
  - Example:

```
!(grade==maximumGrade)
```

  - Alternative:

```
grade != maximumGrade
```

Summary of Operator Precedence and Associativity

Operators	Associativity	Type
()	Left to right	parentheses
++ --	Left to right	Postfix increment-decrement
++ --	Right to left	Prefix increment-decrement
* / %	Left to right	Multiplicative
- +	Left to right	Additive
< <= > >=	Left to right	Relational
== !=	Left to right	Equality
&&	Left to right	Logical AND
||	Left to right	Logical OR

Boolean Data Type

```
bool test1,test2,test3;
int x=3,y=6,z=4;
test1=x>y                  // false
test2=!(x==y);                  //true
test3=x<y && x<z;               //true
test3= test1|| test2;           //true
test2=!test1;                   //true
```

Example1

```
// compound assignment operators
#include <iostream>
using namespace std;
int main () {
 int a, b=3; a = b; a+=2; // equivalent to
   a=a+2
cout << a;
 return 0;
 }
```

Example 2

- Convert the following algebraic expression to an arithmetic C++ expressions:

$$m = \frac{a + b + c + d + e}{5}$$

Example 3

- Evaluate the following C++ expression:

```
y= a * x * x + b * x + c;
```

Example 4

```
#include <iostream>
using namespace std;
int main()
{
   double x, y;
   cout << "\nEnter two floating-point values: ";
   cin >> x >> y;
   cout << "The average of the two numbers is: "
        << (x + y)/2.0 << endl;
   return 0;
}
```