

Variables and Constants

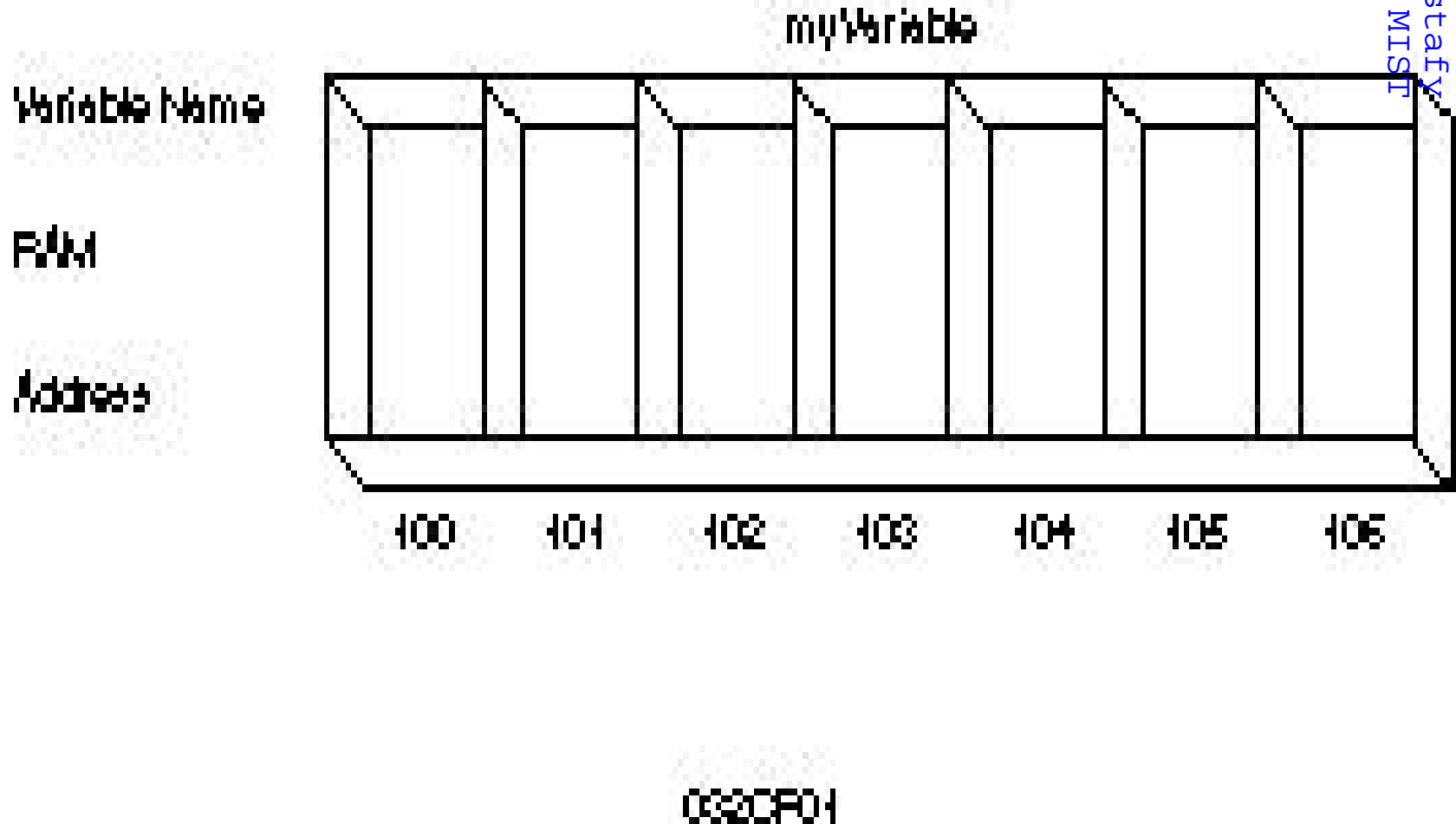
Tanvir Mustafy
Lecturer, MIST

Variable

- A location in computer's memory
- in which a value may be stored
- from which that value may be retrieved

Storage of Variables

Tanvir Mustafiz
Lecturer, MIST



Setting Aside Memory

Defining a variable in C++

- Telling the compiler what kind of variable it is: integer, float, double, character, boolean, string etc
- Tells the compiler how much room (bytes) to set aside in memory and what kind of value you want to store in your variable
- Computers use bits and bytes to represent values, and memory is measured in bytes

Binary and Hexadecimal

Base 10 or Decimal Number System

- Base 10 uses the digits 0-9.
- The columns are powers of ten: $1(10^0)$, $10(10^1)$, $100(10^2)$
- With n columns you can represent 0 to $(10^n - 1)$. Thus, with 3 columns you can represent 0 to $(10^3 - 1)$ or 0-999.

Binary and Hexadecimal

Base 8 or Octal Number System

- Base 8 uses the digits 0-7.
- The columns are powers of eight: 1s ($1(8^0)$), $8(8^1)$, $64(8^2)$
- With n columns you can represent 0 to (8^n-1)
- To represent the number 15_{10} in base 8 you would write 17_8 . This is read "one, seven, base eight."

Binary

- There are only two digits: 0 and 1

- 01011000

- 87654321

- $1 * 64(2^{(7-1)}) = 64$

$$0 * 32(2^{(6-1)}) = 0$$

$$1 * 16(2^{(5-1)}) = 16$$

$$1 * 8(2^{(4-1)}) = 8$$

$$0 * 4(2^{(3-1)}) = 0$$

$$0 * 2(2^{(2-1)}) = 0$$

$$0 * 1(2^{(1-1)}) = 0$$

$$= 88$$

Why Base 2?

Tanvir Mustafy
Lecturer, MIST

- It corresponds so cleanly to what a computer needs to represent
- Computers do not really know anything at all about letters, numerals, instructions, or programs
- Computers have just circuitry, and at a given juncture they recognize either absence or presence of power or magnetism

Bits and Bytes

- One binary digit, 1s and 0s (or bits)
- Early computers could send 8 bits at a time
- Code was written using 8-bit numbers--called bytes
- With 8 binary digits you can represent up to 256 different values
- If all 8 bits are set (1), the value is 255. If none is set (all the bits are clear or zero) the value is 0. 0-255 is 256 possible states.

Kilobytes (KB), Megabyte (MB) and Gigabyte (GB)

- 2^{10} (1,024) is roughly equal to 10^3 (1,000)
- Scientists started referring to 2^{10} bytes as 1KB or 1 kilobyte
- $1024 * 1024$ (1,048,576) is close enough to one million to receive the designation 1MB or 1 megabyte
- 1,024 megabytes is called 1 gigabyte (giga implies thousand-million or billion).

Base 16 or Hexadecimal

- In base 16 there are sixteen numerals: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F
- F8C hex
- 123
- $F * 16^{(3-1)} = 15 * 16^2 = 15 * 256 = 3840$
- $8 * 16^{(2-1)} = 8 * 16^1 = 8 * 16 = 128$
- $C * 16^{(1-1)} = 12 * 16^0 = 12 * 1 = 12$
- The number is 3980

Converting Binary to Hexadecimal

Tanvir Mustafy
Lecturer, MIST

- 11111100
- Treat this binary number as two sets of 4 digits
- 1111 1100
- The right set is 1100. In decimal that is 12, or in hexadecimal it is C
- The left set is 1111, which in base 10 is 15, or in hex is F
- 1111 1100 is
- FC

Size of Variables

- On any one computer, each variable type takes up a single, unchanging amount of memory
- That is, an integer might be two bytes on one machine, and four on another
- But on either computer it is always the same, day in and day out

Fundamental Variable Types

Tanvir Mustafy
Lecturer, MIST

- integer variables
 - signed (either negative or positive)
 - Short (represent numbers from -32,768 to 32,767)
 - Long
 - unsigned (always positive)
 - Short (represent numbers from 0 to 65,535)
 - Long
- floating-point variables: values that can be expressed as fractions--that is, they are real numbers
- character variables: hold a single byte and are used for holding the 256 characters and symbols of the ASCII and extended ASCII character sets.

ASCII

- American Standard Code for Information Interchange
- every computer operating system supports ASCII
- the set of characters standardized for use on computers
- ASCII character set and its ISO (International Standards Organization) equivalent are a way to encode all the letters, numerals, and punctuation marks

Variable Types

• unsigned short int	2 bytes	0 to 65,535
• short int	2 bytes	-32,768 to 32,767
• unsigned long int	4 bytes	0 to 4,294,967,295
• long int	4 bytes	-2,147,483,648 to 2,147,483,647
• int (16 bit)	2 bytes	-32,768 to 32,767
• int (32 bit)	4 bytes	-2,147,483,648 to 2,147,483,647
• unsigned int (16 bit)	2 bytes	0 to 65,535
• unsigned int (32 bit)	2 bytes	0 to 4,294,967,295
• Char	1 byte	256 character values
• Float	4 bytes	1.2e-38 to 3.4e38
• Double	8 bytes	2.2e-308 to 1.8e308

Defining a Variable

- `int myAge;`
- `unsigned int;`
- `unsigned short x;`
- `unsigned int x, y;`
- `long x;`
- `char x;`
- `typedef unsigned short int USHORT`
 - `USHORT x;`

Assigning Values to Your Variables

- unsigned short Width;
- Width = 5;
- OR
- unsigned short Width = 5;
- int myAge = 39, yourAge, hisAge = 40;

When to Use short and When to Use long

Tanvir Mustafy
Lecturer, MIST

- If there is any chance that the value you'll want to put into your variable will be too big for its type, use a larger type
- Unsigned short integers, assuming that they are two bytes, can hold a value only up to 65,535
- Signed short integers can hold only half that
- unsigned long integers can hold an extremely large number (4,294,967,295)
- If you need a larger number, you'll have to go to float or double

Keywords

- Some words are reserved by C++
- These may not be used as variable names.
- These are keywords used by the compiler to control your program
- Keywords include if, while, for, and main
- The compiler manual should provide a complete list of keywords

Wrapping Around an unsigned Integer

Tanvir Mustafy
Lecturer, MIST

- unsigned long integers have a limit to the values they can hold
- When an unsigned integer reaches its maximum value, it wraps around and starts over

Wrapping Around a signed Integer

- A signed integer is different from an unsigned integer, in that half of the values you can represent are negative
- When you run out of positive numbers, you run right into the largest negative numbers and then count back down to 0

Characters and Numbers

Tanvir Musaffy
Lecturer, MIST

- When you put a character, for example, ``a'`, into a char variable, what is really there is just a number between 0 and 255
- compiler knows, however, how to translate back and forth between characters
- the value/letter relationship is arbitrary; there is no particular reason that the lowercase "a" is assigned the value 97
- There is a big difference between the value 5 and the character ``5'`
- The latter is actually valued at 53

Special Printing Characters

Tanvir Mustafy
Lecturer, MIST

- The C++ compiler recognizes some special characters for formatting
- put these into your code by typing the backslash (called the escape character), followed by the character
- An *escape character* changes the meaning of the character that follows it
- normally the character n means the letter n, but when it is preceded by the escape character (\) it means new line

Special Printing Characters

Tanvir Mustafy
Lecturer, MIST

- to put a tab character into your code, you would enter a single quotation mark, the slash, the letter t, and then a closing single quotation mark:
- `char tabCharacter = '\t';`
- This example declares a char variable (tabCharacter) and initializes it with the character value \t, which is recognized as a tab

The Escape Characters

- ***Character*** ***What it means***
- ***\n*** ***newline***
- ***\t*** ***tab***
- ***\b*** ***backspace***
- ***\"*** ***double quote***
- ***'*** ***single quote***
- ***\?*** ***question mark***
- ****** ***backslash***

Constants

- Like variables, constants are data storage locations
- Unlike variables, and as the name implies, constants don't change
- a constant must be initialized with a value when created and it cannot be assigned a new value later

Constants

- Symbolic
 - `#define studentsPerClass 15`
 - `const unsigned short int studentsPerClass = 15;`
 - `students = classes * studentsPerClass`
 - Every time the preprocessor sees the word `studentsPerClass`, it puts in the text `15`
 - Constants cannot be changed while the program is running. If you need to change `studentsPerClass`, for example, you need to change the code and recompile
- Literal
 - A literal constant is a value typed directly into the program wherever it is needed
 - `int myAge = 39;` `myAge` is a variable of type `int`; `39` is a literal constant. You can't assign a value to `39`, and its value can't be changed.

Enumerated Constants

- Enumerated constants enable you to create new types
- and then to define variables of those types
- whose values are restricted to a set of possible values
-

Enumerated Constants

- `enum COLOR { RED, BLUE, GREEN, WHITE, BLACK };`
- It makes COLOR the name of an enumeration, that is, a new type.
- It makes RED a symbolic constant with the value 0, BLUE a symbolic constant with the value 1, GREEN a symbolic constant with the value 2, etc

Enumerated Constants

- `enum Color { RED=100, BLUE, GREEN=500, WHITE, BLACK=700 };`
- RED will have the value 100; BLUE, the value 101; GREEN, the value 500; WHITE, the value 501; and BLACK, the value 700

Enumerated Constants

- `enum Days { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, ^_Saturday };`
- `Days DayOff;`

Expressions and Statements

Tanvir Mustafy
Lecturer, MIST

- **Statements**

- **$x = a + b;$**
- Unlike in algebra, this statement does not mean that x equals $a + b$
- This is read, "Assign the value of the sum of a and b to x ," or "Assign to x , $a + b$."
- The assignment operator assigns whatever is on the right side of the equal sign to whatever is on the left side

Whitespace

- Whitespace (tabs, spaces, and newlines) is generally ignored in statements
- *Whitespace characters* (spaces, tabs, and newlines) cannot be seen
- Whitespace can be used to make your programs more readable and easier to maintain

- `x=a+b;`

- `x`

`=`

`a`

`+`

`b ;`

- Although this last variation is perfectly legal, it is also perfectly foolish

Blocks and Compound Statements

Tanyir Mustafy
Lecturer, MIST

- compound statement, also called a block
- A block begins with an opening brace ({) and ends with a closing brace (}).
- Although every statement in the block must end with a semicolon, the block itself does not end with a semicolon

Example of a Block Statement

- ```
{ temp = a;
 a = b;
 b = temp; }
```

# Expressions

Tanvir Mustafy  
Lecturer, MIST

- Anything that evaluates to a value is an expression in C++.
- An expression is said to return a value
- Thus,  $3+2$ ; returns the value 5 and so is an expression
- All expressions are statements

# Expressions

- $x = a + b;$
- $a + b$  is an expression
- Because it is an expression, it can be on the right side of an assignment operator
- $y = x = a + b;$ 
  - This line is evaluated in the following order:  
Add  $a$  to  $b$ .
  - Assign the result of the expression  $a + b$  to  $x$ .
  - Assign the result of the assignment expression  $x = a + b$  to  $y$ .

# Operators

Tanvir Mustafy  
Lecturer, MIST

- An operator is a symbol that causes the compiler to take an action
- Operators act on operands
- in C++ all operands are expressions
- categories of operators
  - Assignment operators.
  - Mathematical operators

# Operators

- Assignment operator (=)
  - causes the operand on the left side of the assignment operator to have its value changed to the value on the right side of the assignment operator
  - $x = a + b$ ; assigns the value that is the result of adding  $a$  and  $b$  to the operand  $x$
  - An operand that legally can be on the left side of an assignment operator is called an lvalue That which can be on the right side is called an rvalue
  -

# Operators

- Constants are r-values. They cannot be l-values. Thus, you can write
- `x = 35; // ok` but you can't legally write
- `35 = x; // error, not an lvalue!`



# Mathematical Operators

Tanvir Mustafy  
Lecturer, MIST

- five mathematical operators:
- addition (+)
- subtraction (-)
- multiplication (\*)
- division (/) and
- modulus (%).

# Combining Assignment and Mathematical Operators

- `int myAge = 5;`  
  `int temp;`  
  `temp = myAge + 2; // add 5 + 2 and put it in temp`  
  `myAge = temp; // put it back in myAge`
- `myAge = myAge + 2;`
- 
- `myAge += 2;`

# Combining Assignment and Mathematical Operators

Tanvir Mustafy  
Lecturer, MIST

- `myAge += 2;`
- The self-assigned addition operator (`+=`) adds the rvalue to the lvalue and then reassigns the result into the lvalue
- If `myAge` had the value 4 to start, it would have 6 after this statement.
- There are self-assigned subtraction (`-=`), division (`/=`), multiplication (`*=`), and modulus (`%=`) operators as well.

# Increment and Decrement

- increasing a value by 1 is called incrementing
- and decreasing by 1 is called decrementing
- There are special operators to perform these actions in C++
- `a = a + 1;`
- `a += 1;`
- `a++;` // Start with a and increment it.

# Prefix and Postfix

- increment operator (++) and the decrement operator(--) come in two varieties:
  - Prefix: `int a = ++x;`//Increment the value and then fetch it
  - Postfix: `int b = x++;` //Fetch the value and then increment the original

# Precedence

- $x = 5 + 3 * 8;$
- Precedence: which is performed first, the addition or the multiplication?
- Every operator has a precedence value
- Nesting Parentheses

# The Nature of Truth

- In C++, zero is considered false
- and all other values are considered true, although true is usually represented by 1
- If a statement is true, all you know is that it is nonzero, and any nonzero statement is true.

# Relational Operators

- The relational operators are used to determine whether two numbers
  - are equal,
  - or if one is greater or less than the other.
  - every relational statement evaluates to either 1 (TRUE) or 0 (FALSE)
  - `myAge == yourAge`; // is the value in myAge the same as in yourAge?
  - `myAge > yourAge`; // is myAge greater than yourAge?



# The Relational Operators

Tanvir Mustafy  
Lecturer, MIST

| <b>• Name</b>                       | <b>Operator</b> | <b>Sample Evaluations</b>                            |
|-------------------------------------|-----------------|------------------------------------------------------|
| <b>• Equals</b>                     | <b>==</b>       | <b>100 == 50;false</b><br><b>50 == 50;true</b>       |
| <b>• Not Equals</b>                 | <b>!=</b>       | <b>100 != 50;true</b><br><b>50 != 50;false</b>       |
| <b>• Greater Than</b>               | <b>&gt;</b>     | <b>100 &gt; 50;true</b><br><b>50 &gt; 50;false</b>   |
| <b>• Greater Than<br/>or Equals</b> | <b>&gt;=</b>    | <b>100 &gt;= 50;true</b><br><b>50 &gt;= 50;true</b>  |
| <b>• Less Than</b>                  | <b>&lt;</b>     | <b>100 &lt; 50;false</b><br><b>50 &lt; 50;false</b>  |
| <b>• Less Than<br/>or Equals</b>    | <b>&lt;=</b>    | <b>100 &lt;= 50;false</b><br><b>50 &lt;= 50;true</b> |