

Memory

- Computers get their powerful flexibility from the ability to store and retrieve data
- Data is stored in *main memory*, also known as *Random Access Memory (RAM)*

Memory

'h'	'e'	'l'	'l'	'o'	'\0'
1702	1703	1704	1705	1706	1707

- Addressing
 - Sequential locations where data can be stored
 - Each location has an *address* (an integer)
- Data is *stored and retrieved* by address
 - Could use the actual location number
 - Easier if we are able to name locations
 - “store this data at location *stan*”
 - “get whatever data is a location *stan*”



Allocating Space

- Before a program can store a piece of data, it has to **allocate** space in main memory
 - This involves reserving memory at some location and giving that location a name `a = 3;`
 - That name is called a *variable* `string1 = "hello";`
 - The program uses the variable to store and retrieve data
- This is done with a *variable declaration* statement
 - Made up of a **type** and a **name**
 - For example:

```
string str1;  
str1 = "hello";
```
 - Where `string` is the type and `string1` is the name

Data Types

- Every variable has a type
 - Just like every piece of data (integer, real number, character, string)
 - A variable can only store data of the same type
- Some C++ data types:

int	Integer (whole number, positive or negative)
float	Real number (includes decimal, positive or negative)
char	Character (one letter 'a', 'b')
string	String of characters or a sentence ("hello world!")



Identifiers (Variable Names)

- Consist of letters, digits, and the underscore character (_); **NO SPACE**
- *Must begin with a letter or underscore*
- C++ is *case sensitive*
 - NUMBER is not the same as number



Identifiers (continued)

- The following are legal identifiers in C++:
 - `first`
 - `conversion`
 - `payRate`
1. consist of letters, digits, and the underscore character (`_`); **NO SPACE**
 2. *Must begin with a letter or underscore*
 3. C++ is *case sensitive*

TABLE 2-1 Examples of Illegal Identifiers

Illegal Identifier	Description
<code>employee Salary</code>	There can be no space between <code>employee</code> and <code>Salary</code> .
<code>Hello!</code>	The exclamation mark cannot be used in an identifier.
<code>one+two</code>	The symbol <code>+</code> cannot be used in an identifier.
<code>2nd</code>	An identifier cannot begin with a digit.

Assignment Operator

- The assignment operator (=) stores a piece of data in a memory location
 - LHS argument: the variable where you want to store it
 - RHS argument: the data to store
- For example, storing the number 4 (an integer) is a two-step process:
 - First, **allocate memory** by declaring a variable of type integer

```
int myVariable;
```
 - Then, **assign** it the piece of data

```
myVariable = 4;
```
 - We say that `myVariable` has the *value* 4

Assignment and Expressions

- Remember that an operand can be any expression that evaluates to the right type of data
- So the RHS (right-hand side) of an assignment can be an expression:

```
int x;
```

```
x = 5 + 6 - 7;
```

- The RHS of an assignment is always evaluated first, then the resulting value is stored

Using Stored Data

- Variable are used to store and retrieve data
- Up to this point we have used *literal* data in our expressions:

```
cout << 41;
```

- Instead, we can use a stored piece of data:

```
int x;
```

```
x = 41;
```

```
cout << x;
```

See coding

- A variable can go in any expression where a literal piece of data could go

Using Stored Data

- Variables are reusable
 - Each assignment stores a new value and *over writes* the old

```
int x;
```

```
x = 5;
```

```
cout << x;
```

```
x = 6;
```

```
cout << x;
```

```
x = x + 10;
```

```
cout << x;
```

Check your knowledge

- What gets printed on the screen when this code runs?

```
int x; // declare a variable (named x) as an integer type
int y; // declare a variable (named y) as an integer type

// print out the string "hello" to the terminal
cout << "Hello";
cout << endl; // start with a new line
// calculate the RHS and assign the results to the LHS
x = 1 + 4 - 3;
cout << x; // print out the value of the (variable) x, which is 2.
cout << endl; // start a new line
// calculate the RHS 2-1 => 1; then assign the RHS 1 to LHS y.
y = x - 1; // the y value should be 1.
y = y + 5; // RHS: 1 + 5 => 6; LHS: assign 6 to the LHS
cout << y; // print out the value of y, which is 6.
cout << endl; // start a new line.
```

Function scope

- Variables are declared *inside* functions
 - We say they have function *scope*
- Those variables are only valid inside the function they are declared in
 - The name is meaningless outside
- Think of memory as being divided up between the functions
 - Each function gets its own chunk of memory
 - Variables declared in a function allocate memory in that chunk
 - When the function is done, that chunk is erased