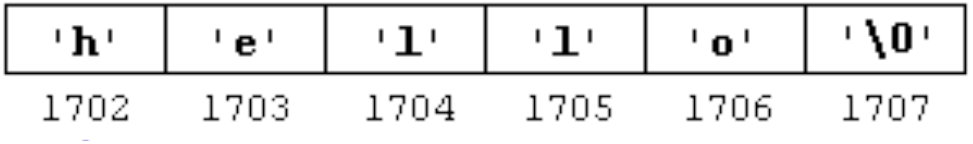
Memory

- 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



- Addressing
  - Sequential locations where data can be stored
  - Each location has an ***address*** (an integer)
  - A computer with 1 Gigabyte of RAM has a billion 1-byte locations to store data
    - The first location is at address 0
    - The last location is around address 999,999,999
- 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
  - That name is called a *variable*
  - The program uses the variable to store and retrieve data

string1 = "hello";

- This is done with a *variable declaration* statement
  - Made up of a ***type*** and a ***name***
  - For example:

```
string string1;
string1 = "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
string	String of characters

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
`employee Salary`	There can be no space between `employee` and `Salary`.
`Hello!`	The exclamation mark cannot be used in an identifier.
`one+two`	The symbol + cannot be used in an identifier.
`2nd`	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;
int y;

cout << "Hello";
cout << endl;
x = 1 + 4 - 3;
cout << x;
cout << endl;
y = x - 1;
y = y + 5;
cout << y;
cout << endl;
```

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

Combining Operations

- Given this statement, what will the computer do?

```
cout << 10 + 11 + 12 - 3;
```

Combining Operations

- Assume that the computer can only do ***one operation*** at a time
  - What is the sequence of operations the computer will perform?
  - How many reasonable sequences can you come up with?

```
cout << 10 + 11 + 12 - 3;
```

Combining Operations

- Assume that the computer can only do one operation at a time
  - What is the sequence of operations the computer will perform?
  - How many reasonable sequences can you come up with?

```
cout << 10 + 11 + 12 - 3;
```

1. Add 10 and 11
2. Then add 12
3. Then subtract 3
4. Then print it to the screen

Operator priority:
1. ()
2. *, /, %
3. +, -

- Why does it have to be in that order?

Expressions

- An operator with its arguments (also called *operands)* is called an ***expression***

```
5 + 6
17 - 8
```

- Every ***expression evaluates to a value***, a piece of data

`5 + 6` evaluates to `11`

`17 - 8` evaluates to `9`

- Syntax rule: an operand can be any expression that evaluates to the right type of data
- Necessary rule: an operator *must* evaluate it's operands before it can execute

Using Expressions

- I want to print to the screen
- Start with the right operator
  - Insertion (`<<`)
- Then give it the correct arguments
  - LHS (Left Hand Side) is `cout`, meaning print to the screen
  - RHS (Right Hand Side) can be any piece of data that I want to print
    - E.g. `cout << 5;`
- *Before* the operator can print, it has to evaluate the RHS to see what it is supposed to print
    - It could be a number: `cout << 5;`
    - Or, an expression that evaluates to a number: `cout << 2 + 3;`