```cpp
// chario: Character I/O
#include <iostream>    // cin, cout, get()
using namespace std;

int main()
{
    char ch;

    cout << "Loop while character is not a pound sign:\n";
    cin >> ch;      // Line-buffered input
    while (ch != '#')
    {
        cout << ch;
        cin >> ch;
    }

    cout << "\n\nLoop while character is not a newline:\n";
    while ((ch = cin.get()) != '\n')   // int from get() is demoted to char
        cout << ch;
    cout << "\n\n";
    return 0;
}

// Note: Entered data is in **bold** type.

Output:
Loop while character is not a pound sign:
On **Tuesday**, **bring a #2 pencil.**
OnTuesday,**bring a #2 pencil.**

Loop while character is not a newline:
2 pencil.

Input Buffer:
On^Tuesday,^bring^a^#2^pencil.\n```
// chario: Character I/O  
#include <iostream>   // cin, cout, ignore(), get()  
using namespace std;  

int main()  
{  
  char ch;  
  cout << "Loop while character is not a pound sign:\n";  
  cin >> ch;  
  while (ch != '#')  
  {  
    cout << ch;  
    cin >> ch;  
  }  
  cin.ignore(80, '
');   // discards next 80 characters or  
                       // up thru first newline, whichever comes first  
  cout << "\n\nLoop while character is not a newline:\n";  
  while ((ch = cin.get()) != '\n')  
  {  
    cout << ch;  
    cin.get(ch);  
  }  
  cout << endl;  
  return 0;  
}  

Output:  
Loop while character is not a pound sign:  
On Tuesday, bring a #2 pencil.  
On Tuesday, bring a #2 pencil.  

Loop while character is not a newline:  
On Tuesday, bring a #2 pencil.  
On Tuesday, bring a #2 pencil.  

Loop while character is not a newline:  
That's it.  
That's it.  

Input Buffer:  
On"Tuesday,"bring"a"#2"pencil.\n
1. Buffered input: Entered characters are stored in a buffer. Pressing <Enter> makes characters available to the program.

Line-Buffered: flush buffer at newline

Fully-Buffered: flush buffer when full (Usually done with files)

Advantages:
   a. Faster to transmit many characters as a group as oppose to one by one.
   b. Can backspace and make corrections

2. char ch;
   cin >> ch;

Extractor skips whitespace; only takes non-ws characters

3. int get(); // prototype

gets the next character (any character, including ws) and returns it as an int.

Return EOF at eof; EOF is in iostream header file

Advantage: C to C++ conversion, C's `getchar()` becomes C++'s `get()`

```cpp
int getchar(void); // Prototype for getchar()
```

4. 20 while ((ch = cin.get()) != 'n') // () are needed
    assignment expression

   != has higher precedence than =; Without ()

   20 while (ch = (cin.get() != 'n')) // () are redundant
5. **istream& get(char&); // prototype**

Gets next character (any character, including ws) and assigns it to its argument

Returns a reference to istream object used to invoke it

a. `char c1, c2, c3; cin.get(c1).get(c2) >> c3;`  // inputs c1, c2, and c3

   Chained function calls; associates L – R;
   c1 and c2 could get ws, but c3 could not.

b. `char c1, c2, c3; cin.get().get() >> c3;`  // Invalid

   returns int, not cin

6. `char ch;  
   ...  
   while (cin.get(ch))`

There is a function in the istream class that will convert the istream return value to bool when appropriate. This bool value is true if the attempt to read was successful and false otherwise.

What would cause the above call to fail?

ctrl+Z will simulate eof at the keyboard in a DOS environment;
ctrl+D in Unix.
1. `cin.ignore(80, "\n");`

   Reads and discards next 80 characters or up thru 1\textsuperscript{st} newline, whichever comes first.

   `cin.ignore(1024, "\n");`  // empties buffer

2. `istream& ignore(int = 1, int = EOF);`  // prototype

   Reads and discards the next 1 character or thru eof

   `cin.ignore();`  // `cin.ignore(1, EOF);`

3. Assume input buffer contains: `abcdefg\n`

   ```
   10 char letter;
   20 cout << "Please input some characters:\n";
   30 cin.ignore(2, 'd');  // ignores ab; pointer moves to c
   40 cin.get(letter);  // gets c
   50 cout << letter;
   60 cin.ignore(10, 'f');  // ignores def
   70 cin.get(letter);  // gets g
   80 cout << letter << endl;
   ```

   Output:
   Please input some characters:
   `abcdefg`
   `cg`

Note:
1. If the input buffer is empty, `ignore()` allows you to enter characters.

2. file pointer or input pointer or file position indicator
// CstringConstant: C-style String Constants

#include <iostream> // cout
#include <cstring> // strlen()

using namespace std;

int main()
{
    cout << "The C strings:
" // string concatenation
    "Oh no!" "\t" "Maybe so..." "\n\n";

    cout << "The lengths of the strings:
" << strlen("Oh no!") << "\t" << strlen("Maybe so...") << "\n\n";

    cout << "The sizes of the strings (includes null characters):
" << sizeof "Oh no!" << "\t" << sizeof "Maybe so..." << "\n\n";

    cout << "Strings	Addresses	Pointed-to Objects
" << "Oh no!" << "\t" << static_cast<void*> ("Oh no!") << "\n"
    << "Maybe so..." << "\t" << static_cast<void*> ("Maybe so...") << "\n"
    << "\t" << "\n";
    return 0;
}

Output:
The C strings:
Oh no!      Maybe so...
The lengths of the strings:
6           11
The sizes of the strings (includes null characters):
7           12

Strings     Addresses    Pointed-to Objects
Oh no!      00451F4C     O
Maybe so... 00451F54     M

Note:
void* is a general-purpose pointer type that can point to any data type. All
pointer types can be assigned to void* without casting. void* must be cast
to another pointer type. void* cannot be dereferenced.

void* pv;       // pv can contain the address of any data type
void fun(void*); // fun() operates independently of a specific data type
int* pi;
...
pv = pi;       // no cast required
pi = static_cast<int*> (pv); // cast required
cout << *pv;   // Invalid
1. C++ offers: C-style strings or the string class

2. **String constant:** Zero or more characters enclosed in double quotes (delimiters)

   Enclosed characters, plus terminating null character are stored in contiguous bytes in memory.

   "Oh^no!" stored as
   
   Oh ^ no ! \0

   "Maybe^so..." stored as
   
   M a y b e ^ s o . . . \0

   **Null character:** '\0', used to mark the end of a C string
   
   a. non-printable character whose ASCII code value is 0
   b. inserted in the C string constant by the compiler

3. String constants are stored in external static storage class.
   (permanent storage, file scope, static duration, and internal linkage)

   Stored just once and lasts for the duration of the program.

4. ```
   size_t strlen(const char* string);
   // prototype in cstring
   ```

   returns the length of a string, not counting the null character.

   ```
   size_t is the type generated by the sizeof operator;
   usually, unsigned int, but can vary.
   ```

   ```
   typedef unsigned size_t;
   ```

   Using `size_t` rather than `unsigned` aids portability.

   Note: C++ imposes no limits on the length of a string.
5. Entire quoted string acts as pointer to the first character of the string.

Similar to array name acting as pointer to first element.

```cpp
    cout << "Oh no!";                // outputs Oh no!
    cout << static_cast<void*>("Oh no!"); // outputs address
```

Normally if `cout` is given a pointer, it prints the address. If it is `char*`, `cout` displays the pointed-to-string

```cpp
    cout << *"Oh no!";          // outputs pointed-to-char
```

6. Possible definition for strlen():

```cpp
    size_t strlen(const char* s) {
        size_t i = 0;

        while (*s) // or for ( ; s[i]; ++i) // continue;
            ++i;
        ++s;

        return i;
    }
```
// StringVariables: "String" Variables
// (There really are no C-string variables.)

#include <iostream> // cin, cout, ignore()
#include <iomanip>   // setw()

using namespace std;

const int SIZE = 10;

int main()
{
    char name[SIZE]; // To store a string

    for (int i = 1; i <= 3; i++) // Three examples
    {
        cout << "Enter a name: ";
        cin >> name; // cin stores the null character in the array
        cin.ignore(80, '\n'); // flush the input buffer (input queue)
        cout << "The name is " << name << endl;
        cout << "ASCII codes of the " << SIZE << " characters stored in the array are:
";
        for (int j = 0; j < SIZE; j++)
            cout << setw(7) << static_cast<int>(name[j]); // Note the cast
        cout << endl;
    }

    return 0;
}

Output:
Enter a name: Albert
The name is Albert
ASCII codes of the 10 characters stored in the array are:
       65 108  98 101 114 116 0 0 -8  83

Enter a name: Pat Wong
The name is Pat
ASCII codes of the 10 characters stored in the array are:
        80  97 116  0 114 116 0 0 -8  83

Enter a name: ALBERT
The name is ALBERT
ASCII codes of the 10 characters stored in the array are:
       65  76  66  69  82  84 0 0 -8  83
1. There is no built-in string data type.

Strings are stored in a `char` array.

```cpp
13 char name[10];       // in StringVariables
...
17 cout << "Enter a name: ";
18 cin >> name;          // cin stores null character in array
```

a. Buffer contents: **Albert\n**

```plaintext
name array:
A l b e r t \0 g a r b a g e
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
```

Null character (\0) is used to mark the end of the string.

non-printable character whose ASCII code value is 0; Inserted by cin

b. Buffer contents: **^^^^^^^^^Pat^Wong\n**

```plaintext
name array:
P a t \0 r t \0 g a r b a g e
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
```

^Wong\n is left in buffer - Reason for cin.ignore(80,\n');

c. Buffer contents: **ALBERT\n**

```plaintext
name array:
A L B E R T \0 g a r b a g e
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
```
2. What happens if user enters: Christopher (11 characters)?
   Walks off the end of the array
   Array must have enough elements for the characters plus the null character.

3. cin:
   Accepts only non-whitespace characters (no spaces, tabs, or newlines)
   Skips leading whitespace and stops at whitespace
   Reads a single word
   Null character is inserted

4. char type versus C-String
   'A'       "A"
   A  (1 byte) A  \0  (2 bytes)
   65       65 0
   Built-in type (char) Compound type (array of chars)

5. char st[3] = { 'g', 'o', '\0'}, ch[3] = { 'c', 'a', 't'};

   char array        ch[0]  [c]
   ch[1]  [a]
   ch[2]  [t]
   char array        st[0]  [g]
   containing        st[1]  [o]
   string            st[2]  \0

   cout << st;       // Outputs go
   cout << ch;       // Outputs catgo

Is the following declaration equivalent to the above?
// GetString1: Inputting strings

#include <iostream> // cout, getline()

using namespace std;

const int SIZE = 80;

int main()
{
    char name[SIZE]; // To store a string
    for (int i = 1; i <= 3; i++) // Three examples
    {
        cout << "Enter a name: ";
        cin.getline(name, SIZE);
        cout << "The name is " << name << "\n";
    }
    return 0;
}

Output:
Enter a name: Albert C. Finkelstein
The name is Albert C. Finkelstein

Enter a name: Edna Dora Ebert
The name is Edna Dora Ebert

Enter a name: Mr. John Paul Gomez
The name is Mr. John Paul Gomez
1. Declare **char array** large enough to hold the string.

2. **istream& getline(char*, int, char = \n'); // Prototype**

\begin{verbatim}
11 char name[80]; // in GetString1
\end{verbatim}

\begin{verbatim}
15 cout << "Enter a name: ";
16 cin.getline(name, 80);
\end{verbatim}

1\textsuperscript{st} arg: address of location to store string, the base address
2\textsuperscript{nd} arg: 1 greater than maximum number of chars. to be read
3\textsuperscript{rd} arg: termination character causing input to stop; defaults to \n'

Inputs characters until it reads 79 characters, leaving room for null char.
or until it reaches newline, whichever comes first.

Reads **thru** newline (the termination character), replacing newline with
the null char.

3. Recall: Two previous forms of get()

\begin{verbatim}
int get();
istream& get(char&);
\end{verbatim}

Third form:

\begin{verbatim}
istream& get(char*, int, char = \n'); // prototype
\end{verbatim}

\begin{verbatim}
cin.get(name, 80);
\end{verbatim}

Inputs characters until it reads 79 characters, leaving room for null
or until it reaches newline, whichever comes first.

Reads **to** newline, adding the null char to end of string.

Alternative to getline():

\begin{verbatim}
 cin.get(name, 80).get(); // member function concatenation
\end{verbatim}

Equivalent to:

\begin{verbatim}
 cin.get(name, 80);
 cin.get();
\end{verbatim}
4. Classic Problem:

```cpp
10  int age;
20  cout << "Enter age: "
30  cin >> age;
...

40  char name[80];
50  cout << "Enter name;
60  cin.getline(name, 80);
```

What happens?  \n' in buffer from line 30 is read as a null string.

Solution:  
```cpp
30  cin >> age;
35  cin.get();  // remove newline from buffer
       // or cin.ignore();
       // or cin.ignore(1024, "\n");
```

or

```cpp
(cin >> age).get();  // () are needed
```

without():  
```cpp
cin >> (age.get());  // has higher precedence than >>
```