C++ for C Coders & Data Structures
Lecture Notes by idebtor@gmail.com

There are many ways to get user input such as `cin`, `cin.get()`, `cin.getline()`, `std::getline()` and more, including the command line arguments as well.

Using std::getline()
It is a standard library function in C++ and is used to read a string or a line from input stream. It is defined in <string> header file.

The general syntax is:
```
istream& getline (istream& is, string& str, char delim);

istream& getline (istream& is, string& str);
```
The second declaration is almost same as that of the first one. The only difference is, it does not accept any delimitation character. This function consider new line or `\n` character as the delimitation character.

So basically, what the getline function does is extracts characters from the input stream and appends it to the string object until the delimiting character is encountered.

__JoyNote__: Coders prefer `std::getline()` to `cin` to get user input. Stay tune and keep on reading for the reason behind.

__JoyNote__: The previous stored value in the string object `str` will be replaced by the input string if any.

Example:
```C++
#include <iostream>
#include <string>
using namespace std;

int main () {
  string str;
  cout << "Please enter your name: ";
  getline (cin, str);
  if (str.length() > 0)
    cout << "Hello, " << str << ", Welcome to C++CC!\n";
  else
    cout << "Hello World!";
  return 0;
}

```
__Input__:
```
C Coders
```
__Output__:
```
Hello, C Coders, Welcome to C++CC!
```

Using std::getline() and stringstream object
In C++, you may use `getline()` and `stringstream` object that works like `sscanf()` in C. A `stringstream` associates a `string` object with a stream allowing you to read from the `string` as if it were a stream. This example tokenizes a line of string into tokens with respect to a delimiter.

```C++
#include <iostream>
```

```C++
#include <sstream>
#include <string>
#include <vector>
using namespace std;

int main() {
  string line = "God is good all the time.";
  vector <string> tokens; // Vector of string to save tokens

  stringstream ss(line);   // stringstream object ss
  string word;
  while(getline(ss, word, ' '))  // Tokenize w.r.t. space ' '
      tokens.push_back(word);

  for(int i = 0; i < tokens.size(); i++) // print the tokens
      cout << tokens[i] << '\n';
}
```

Using cin and >>
It is defined in <iostream> header file. The "c" in cin refers to "character" and 'in' means "input", hence cin means "character input".

The `cin` object is used along with the extraction operator `>>` in order to receive a stream of characters. The general syntax is:

```
cin varName;
```
The extraction operator can be used more than once to accept multiple inputs as:
```
cin >> var1 >> var2 >> … >> varN;
```

The `cin` object can also be used with other member functions such as `getline()`, `read()`, etc. Some of the commonly used member functions are:

- `cin.get(char &ch)`: Reads an input character and store it in ch.
- `cin.getline(char *buffer, int length)`: Reads a stream of characters into the buffer which is `char*`. It stops when
  - it has read length-1 characters or
  - when it finds an end-of-line character (`\n`) or the end of the file.
  - For C++ string object, but use `std::getline()`.
- `cin.ignore(int n)`: Ignores the next n characters from the input stream.-
- `cin.eof()`: Returns a nonzero value if the end of file `eof` is reached.

Example
```C++
#include <iostream>
using namespace std;

int main() {
  int x, y, z;
  cout << "Enter a number: "; // for single input
  cin >> x;

  cout << "Enter 2 numbers: "; // for multiple inputs
  cin >> y >> z;

  cout << "Sum = " << (x+y+z);
  return 0;
}
```

```

When you run the program, a possible output will be:

```
Enter a number: 9
Enter 2 numbers: 1 5
Sum = 15
```

## Differences between `cin >>`, `std::getline()` and `cin.getline()`

`std::getline()` reads till a newline is found and `cin >>` operator of `std::istream` reads till a space (as defined by `std::isspace`) and is found. Both remove their respective delimiter (separator) from the stream but don't put it in the output buffer. The default separator of `std::getline()` is `\n`.

The input operator `>>` is for reading __formatted input__.  It will get input in any primitive data types that you define, but it will stop at the first non data type char encountered such as ` ` or `\n` . For example, if you run

```
int i;
std::cin >> i;
```
it will try to convert the input into an integer whereas the `std::getline()` will read its characters as a string.

By the way, the another stream `cin.getline()` function takes as a parameter a `char*` array into which it reads the characters and the max number of characters. The free function `std::getline()` takes as a parameter a `std::string` so it resizes the string when it is needed. Both functions can take optionally a third parameter, which is a delimiter ('\n' by default).

In conclusion, use `std::getline()` instead of 'cin >>'.

## Avoid using cin >> if you can
Using `cin` to get user input is convenient sometimes since we can specify a primitive data type. However, it is __notorious__ at causing input issues because it doesn't remove the newline character from the stream or do type-checking. So anyone using `cin >> var;` and following it up with another `cin >> stringtype;` or `std::getline();` will receive empty inputs. It's the best practice __not__ to mix the different types of input methods from `cin`.

Another disadvantage of using `cin >> stringvar;` is that `cin` has no checks for length, and it will break on a space. So you enter something that is more than one word, only the first word is going to be loaded. Leaving the space, and following word still in the input stream.

__JoyNote__: A more elegant solution, and __much easier to use is the `std::getline()` function__.

The example below shows getting user input using `std::getline()`, and convert it between types.

```C++
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

int main() {
  string input = "";
  cout << "Enter a sentence(with spaces):\n>";
  getline(cin, input);
  cout << "You entered: " << input << endl << endl;
```
```

```
  int num = 0;
  while (true) {
    cout << "Please enter a number: ";
    getline(cin, input);
    // This code convert from string to number safely.
    stringstream ss(input);
    if (ss >> num) break;
    cout << "Invalid number, please try again" << endl;
  }
  cout << "You entered: " << num << endl << endl;

  char ch  = {0};
  while (true) {
    cout << "Please enter one char: ";
    getline(cin, input);
    if (input.length() == 1) {
      ch = input[0];
      break;
    }
    cout << "Invalid char, please try again" << endl;
  }
  cout << "You entered: " << ch << endl << endl;
  cout << "All done. And without using the >> operator" << endl;
  return 0;
}
```

Convert string to int, long int, or double etc.

The following functions convert string or char array into numeric number types.

```
// C++ std functions and string based:
int stoi (const string&  str, size_t* idx = 0, int base = 10);
long stol (const string&  str, size_t* idx = 0, int base = 10);
Note: stoll(), stoul()

// char array based
long int strtol (const char* str, char** endptr, int base);
double strtod (const char* str, char** endptr);
```

Getting an integer input from an argument (char array).

You can convert the command line arguments (char array) to integer using `strtol()` or to floating-point using `strtod()`. They are defined in `<stdlib.h>`. For example,

```
a=strtol(argv[1], NULL, 0);
b=strtod(argv[2], NULL, 0);
```

__Error checking__:

The function gives you a pointer to the location in the string where parsing terminated. If that points to the terminating NULL, parsing was successful. And of course it is good that you verify `argc` to make sure the user provided enough parameters, and avoid trying to read missing parameters from `argv`. Here is an example:

```
int main(int argc, char *argv[]) {
// The strtol() converts the contents of a string as an integral
// number of the specified base and return its value as a long int.
```

```
// s: specifies the string which has the representation of an int.
// end: refers to an already allocated object of type char*.
//      The value of the end is set by the function to the next
//      character in s after the last valid character. It can
//      also be a null pointer, in which case it is not used.
// b: specifies to the base of the integral value.
// Return Value : The function returns value of two types :
//   *end is NULL. If not, then 0 is returned and *end is not NULL.
  int N = 0;
  char *end;
  if (argc == 2)
    N = strtol(argv[1], &end, 10);

  if (N <= 0 || *end != '\0') {
    std::cout << "Usage: randomN N\n";
    return EXIT_FAILURE;
  }
  // On success, do something with N
}
```

Convert int to string

1. Using C standard library sprintf()
```
char out_string[MAX_BUFFER_SIZE];
sprintf(out_string, "%s%d", "using sprintf: ", 123);
```

2. Using C++ standard library std::stringstream
```
#include <sstream>
std::string out_string;
std::stringstream ss;
ss << 123;
out_string = ss.str();
```

3. Using C++ standard library std::to_string() since C++11
```
#include <iostream>
#include <string>
std::string out_string = std::to_string(123);
```

Reference:
- [Learn C++](https://www.programiz.com/cpp-programming/library-function/iostream/cin)
- [cpluscplus-Using cin to get user input](http://www.cplusplus.com/forum/articles/6046/)
- [stringstream] https://doc.bccnsoft.com/docs/cppreference_en/cppsstream/all.html
- [stringstream example] https://word.tistory.com/m/24?category=539983

One thing I know, I was blind but now I see. John 9:25