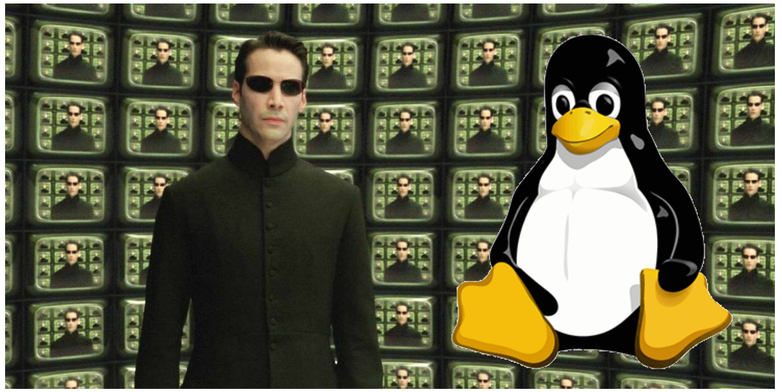
26.06.20 08:32

CSCI 125
Introduction to Computer Science and Programming II
Lecture 5: Char Array and String 2



Jetic Gū

2020 Summer Semester (S2)

Overview

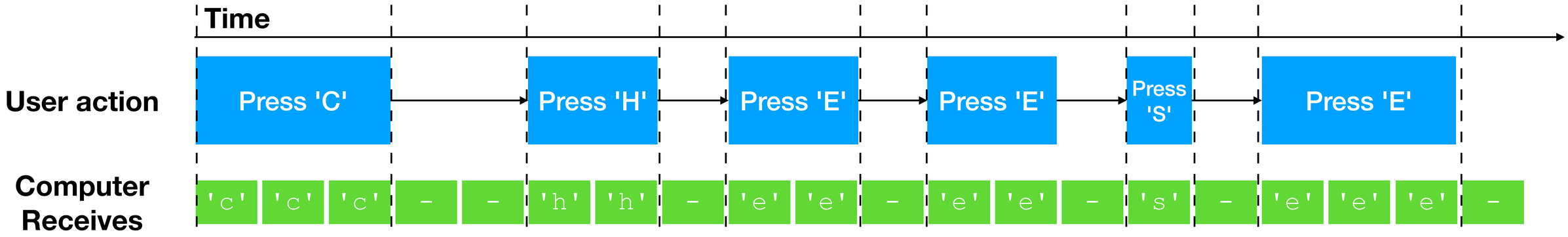
- Focus: Basic C/C++ Syntax
- Architecture: Linux/Unix OS
- Core Ideas:
    1. Stream I/O Operations, File I/O stream
    2. Linux/Unix-Specific Terminal Control Sequence

Stream I/O Operations

Including File I/O

P1
Stream I/O

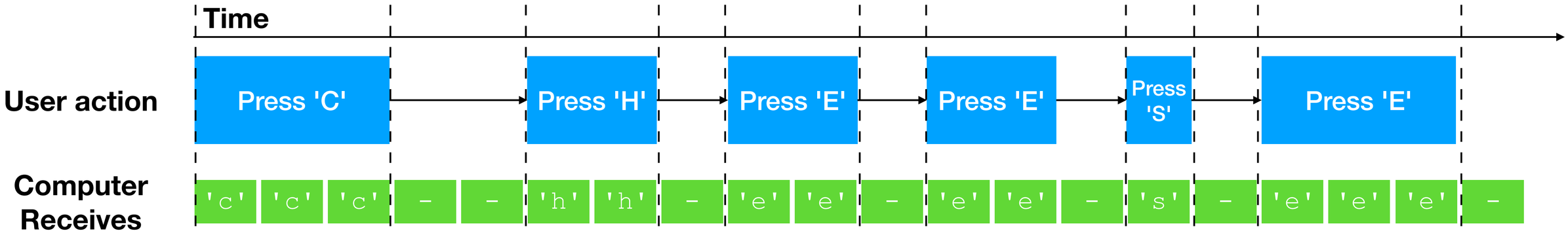
I/O Streams



- Computer: continuously processes information at a **sample rate**
  - Monitor: refresh rate 60Hz - 120Hz; HDMI TV 30Hz - 60Hz (1Hz = 1 cycle per second)
  - Keyboard: 100 bytes per second, i.e. press key for a second, the computer sees 100 keys
    - >200 samples of 'i' -> single click
      After 200, every 50 samples equal a single click

Concept

I/O Streams



- **I/O Stream**: sequence of characters from source to destination
- **Input stream**: from an input device to the computer
- **Output stream**: from the computer to an output device

Concept

I/O Streams

- `iostream` contains definitions of two data types for standard I/O
  - `istream` - input stream
  - `ostream` - output stream
- Has two **variables** declared directly in `iostream`:
  - `istream cin` - stands for common input (stdin, keyboard)
  - `ostream cout` - stands for common output (stdout, onscreen)
- Operators: e.g. >>, << (these are functions/methods!)

Concept

`cin` and the `get` Function

- The `get` function
  - Inputs next character (including whitespace)
  - Stores in memory location indicated by its argument
- Syntax

```
char varChar;
cin.get(varChar);
```

  - `varChar` is a `char` variable
  - `varChar` is the argument (parameter) of the function

Concept

`cin` and the `ignore` Function

- `ignore`: discards a portion of the input
- Syntax

```
char stopChar; int m;

cin.ignore(m, stopChar);
```

- Ignore the next `m` characters or all characters until the character specified by `stopChar`

Concept

File Input/Output

- Files are also treated as streams in C++

1. **Include** `fstream` header
   stands for file stream

2. **Declare** file stream variables
   `ifstream` class for Input stream, `ofstream` class for Output stream

3. **Open** input/output sources using the variables
   e.g. actual files

4. Use `>>`, `<<`, or other **I/O functions**
   Exactly the same as `cin` and `cout`

5. **Close** the files

Concept

File Input

- #1: Include the header
- #3: all `fstream` stuff are under namespace `std`
- #6 declaring and initialising
  this also opens the file for reading
- #7 read from the file
- #9 closes the file

```
1. #include <fstream>
2. #include <iostream>
3. using namespace std;
4. int main() {
5.     int n;
6.     ifstream infile("1.txt");
7.     infile >> n;
8.     cout << n;
9.     infile.close();
10.    return 0;
11.}
```

Concept

1. Demo1

File Input

- Important Things!
  - Some compilers require the filename to be C string (Char Array)

```
string fileName = "1.txt";
ifstream infile(fileName.c_str());
```

  - Always check if the file was opened correctly

```
ifstream infile("1.txt");

if (infile.is_open()) infile >> n;
```

  - Always close the file after using

```
infile.close()
```

Concept

File Input

- Useful things
  - EOF: check if the file has reached the end (no more things to read)

```
if (infile.eof())
    cout << "File has reached its end!" << endl;
```

  - Read an entire line

```
if (myfile.is_open()) {
    while ( getline(myfile, line) ) { // line is string
        cout << line << '\n';
    }
    myfile.close();
}
```

Concept

File Output

- #1/3: same header and namespace
- #6 declaring and initialising
  this also opens the file for reading
- #8 write to the file
- #9 closes the file

```
1. #include <fstream>
2. #include <iostream>
3. using namespace std;
4. int main() {
5.      int n;
6.      ofstream outfile("2.txt");
7.      cin >> n;
8.      outfile << n << endl;
9.      outfile.close();
10.     return 0;
11.}
```

Concept

1. Demo2

File Output

- Important Things!
  - Some compilers require the filename to be C string (Char Array)

```
string fileName = "1.txt";
ofstream outfile(fileName.c_str());
```

  - Always check if the file was opened correctly

```
ofstream outfile("1.txt");

if (outfile.is_open()) outfile << n;
```

  - Always close the file after using

```
outfile.close()
```

Concept

ANSI
escape sequences

How to make your output look COOL

Summary

P2
ANSI

Introduction to ANSI

- Standard for controlling signals for setting
  cursor location,
  colour,
  and other options
  on text terminals and terminal emulators

- If you want your game to look cool on the terminal, you need to know this!

- This only works on supported terminals! (including default ubuntu/macOS terminals, and iterm2)

Concept

P2
ANSI

ANSI Escape Sequence

- Control signals start with two special characters `'\033'`, which is also the code for `Escape key`
  - There’s multiple modes for subsequent things to do
    - For this lecture, we focus on colour and rewriting previous lines
  - Control Sequence Introducer: `'['`
    - Changing subsequent colour
    - Changing the line you are printing

Concept

P2
ANSI

Changing Colour

- Print the following string on screen `"\033[NNm"`
  NN stands for the colour code
  - This changes the colour for all subsequent prints
  - Available options: text colour (foreground), background colour
  - Also: Bold, Underline, etc.
  - Here we teach you only the most basic 3/4bit colour

Concept

Changing Colour

- Colour changes for ALL subsequent print, even after your programme exits.
- The control sequence itself doesn't get printed, only the normal text. This is because the terminal is interpreting the **RAW** text before displaying

```
1. #include <iostream>
2. using namespace std;
3. int main() {
4.     cout << "Normal colour\n";
5.     cout << "\033[31mCode31\n";
6.     cout << "Same colour\n";
7.     cout << "Then, \033[32mCode32\n";
8.     cout << "\033[35mCode35\n";
9.     cout << "\033[1mBold\n";
10.    cout << "\033[0mBack to norm\n";
11.    return 0;
12.}
```

Concept

1. Demo3

P2
ANSI

3/4 bit Colour Code

Colour	Foreground	Background
Black	30	40
Red	31	41
Green	32	42
Yellow	33	43
Blue	34	44
Magenta	35	45
Cyan	36	46
White	37	47
Bright Black	90	100
Bright Red	91	101
Bright Green	92	102
Bright Yellow	93	103
Bright Blue	94	104
Bright Magenta	95	105
Bright Cyan	96	106
Bright White	97	107

Additional	Code	Note
reset	0	everything back to normal
bold/bright	1	often a brighter shade of the same colour
underline	4	
inverse	7	swap foreground and background colours
bold/bright off	21	turn off bold/bright
underline off	24	turn off underline
inverse off	27	turn off inverse

Technical

P2
ANSI

Overwrite previous lines

- After printing the initial map, you might want to change it after the character moves instead of printing the map again.
- Cursor position: where the next input/output character will be printed on screen
  - Move cursor position up: "`\033[1A`"
  - Clean the current line (and move the cursor to the left): "`\033[2k`"

Concept

1. Demo4

P2
ANSI

Read single char and not Print

- Movement: key pressed, programme do not wait until `enter` to proceed
- Do not print typed key
- Requires terminal to enter raw stty mode!

```
system("stty raw");
system("stty opost");
```

- Return to normal

```
system("stty cooked");
```

```
#include <iostream>
using namespace std;
int main() {
  cout << "Press any key to continue..." << endl;
  // Set terminal to raw mode
  system("stty raw");
  system("stty opost");
  // Wait for single character
  char input = getchar();
  // Echo input:
  cout << "--" << input << "--";
  // Reset terminal to normal "cooked" mode
  system("stty cooked");
  // And we're out of here
  return 0;
}
```

Concept

1. Demo5