pset2: Crypto

Tommy MacWilliam

tmacwilliam@cs50.net

September 18, 2011

Today's Music

- Rehab
  - Scarecrow
  - Storm Chaser
  - 1980
  - Graffiti the World
  - Running out of Time

BEFORE YOU DO ANYTHING


```
jharvard@appliance (~): sudo yum -y update
Password: crimson
```

- do this before you do anything with `submit50`!
  - you don't see your password, but you are indeed inputting it!

Backing up Code

- `submit50` saves your code on the CS50 site
  - we only grade your latest submission, so `submit50` often to back up!
- Dropbox (`http://dropbox.com`) already integrated into the appliance
  - automatically backs up your code to Dropbox's site

Getting Code off the Appliance

- Mac
  - select “Connect to Server” from Finder’s “Go” menu
  - input `smb://192.168.56.50` under “Server Address”
- Windows
  - open Windows Explorer, aka My Computer
  - input `\\192.168.56.50\jharvard` into the address bar

This old man


```
jharvard@appliance (~/pset2): ./oldman
This old man, he played one
He played knick-knack on my thumb
Knick-knack paddywhack, give your dog a bone
This old man came rolling home
```

TODO

1. loop over verses
2. display each verse

Loops

- 10 verses, each slightly different
- can store verses in variables
  - verses are only slightly different, so avoid repetition!
- can use conditions
  - different text is displayed depending on verse number

TODO

1. ~~loop over verses~~
2. display each verse

Functions

- function: block of code aimed at accomplishing a single task
  - take input, produce output
- task: display a verse
  - input: which verse to display
  - output: text of verse

TODO

1. ~~loop over verses~~
2. ~~display each verse~~

Caesar


```
jharvard@appliance (~/pset2): ./caesar 13
This is CS50.
Guvf vf PF50.
```

TODO

1. get *k* from command line and convert to int
2. prompt for string to encode
3. loop over each character of the string
4. output each encoded letter, making sure to not encode non-letters

Getting Input

- `argc`: number of arguments given
- `argv[]`: array of strings
- `./caesar 13`
  - `argc == 2`
  - `argv[0] == "caesar"`
  - `argv[1] == "13"`

atoi

- converts a string to an integer

```
string a = "50";
int i = atoi(a);
```

Using Command-Line Arguments

- example time!
  - `args.c`

TODO

1. ~~get *k* from command-line and convert to int~~
2. prompt for string to encode
3. loop over each character of the string
4. output each encoded letter, making sure to not encode non-letters

Strings

- pset1 had numerical input, now we have words
- `string`: sequence of characters

```
string name = GetString();
printf("Your name is %s\n", name);
```

TODO

1. ~~get *k* from command-line and convert to int~~
2. ~~prompt for string to encode~~
3. loop over each character of the string
4. output each encoded letter, making sure to not encode non-letters

Strings Again

- `char`: single character, type just like `int` or `float`
- strings are just `char` arrays
  - `strlen`: get length of string

```
string word = GetString();
int length = strlen(word);
for (int i = 0; i < length; i++)
    printf("%c", word[i]);
```

TODO

1. ~~get *k* from command-line and convert to int~~
2. ~~prompt for string to encode~~
3. ~~loop over each character of the string~~
4. output each encoded letter, making sure to not encode non-letters

Caesar Cipher

- $c_i = (p_i + k) \,\%\, 26$
  - c_i: i^{th} character in the ciphertext
  - p_i: i^{th} character in the cleartext
  - k: number of rotations (user's input)
  - $\%\, 26$: `Z` should wrap to `A`
- `http://en.wikipedia.org/wiki/Caesar_cipher`

Caesar Cipher

T	h	i	s		i	s		C	S	5	0	.
+	+	+	+		+	+		+	+			
13	13	13	13		13	13		13	13			
↓	↓	↓	↓		↓	↓		↓	↓	↓	↓	↓
G	u	v	f		v	f		P	F	5	0	.

ASCII

- `http://www.asciitable.com/`
- ASCII maps characters to numbers
  - `'A' = 65`
  - `'a' = 97`

ASCII and You

- example time!
  - `ascii.c`

ASCII and Caesar

- don't forget to %!
- however: `('Z' + 2) % 26 == 20`
  - should be `'B'`, or 67!

Keep in Mind

- capitalization must be preserved
- letters should never become symbols
- symbols should not be changed

TODO

1. ~~get *k* from command-line and convert to int~~
2. ~~prompt for string to encode~~
3. ~~loop over each character of the string~~
4. ~~output each encoded letter, making sure to not encode non-letters~~

Vigenere


```
jharvard@appliance (~/pset2): ./vigenere tommy
This is CS50.
Mvue gl QE50.
```

TODO

1. read keyword from command-line
2. prompt for string to encode
3. loop over string
4. loop over keyword, making sure to restart when end of keyword reached
5. output each encoded letter, making sure to not encode non-letters

Getting Input

- word taken at command line instead of integer
  - `argv[]` already contains strings, so no need to `atoi`!
- prompting for plaintext? `GetString()`, just like before

TODO

1. ~~read keyword from command-line~~
2. ~~prompt for string to encode~~
3. loop over string
4. loop over keyword, making sure to restart when end of keyword reached
5. output each encoded letter, making sure to not encode non-letters

Vigenere Cipher

- $c_i = (p_i + k_j) \,\%\, 26$
  - c_i: i^{th} character in the ciphertext
  - p_i: i^{th} character in the plaintext
  - k_j: j^{th} character in the keyword (user's input)
    - keyword can have different length than p!
  - $\%\, 26$: `Z` should wrap to `A`

Vigenere Cipher


```
T  h  i  s  !     i  s     C  S  5  0  .
+  +  +  +        +  +     +  +
t  o  m  m        y  t     o  m
↓  ↓  ↓  ↓  ↓     ↓  ↓     ↓  ↓  ↓  ↓  ↓
M  v  u  e  !     g  l     Q  E  5  0  .
```

Vigenere Cipher

- rotate each character by a different amount!
  - after each letter, go to next letter in keyword
  - but, don't go to next letter in keyword if character in plaintext is a symbol
  - at end of keyword, go back to beginning of keyword
- need to keep track of position in plaintext AND position in keyword

TODO

1. ~~read keyword from command-line~~
2. ~~prompt for string to encode~~
3. ~~loop over string~~
4. ~~loop over keyword, making sure to restart when end of keyword reached~~
5. ~~output each encoded letter, making sure to not encode non-letters~~

Style

Good code style is STILL serious business.

But so is design

- DRY: **D**on't **R**epeat **Y**ourself
  - copy/pasting code? bad idea
  - rewriting the same logic several times? bad idea

Functions

- functions allow you to reuse code
- break up large problems into smaller problems
- organize your code

One More Thing

- `https://www.cs50.net/resources/cppreference.com/stdstring/`
- don't rewrite functions that already exist!
  - I mean, someone else probably worked really hard on them

BEFORE YOU GO ANYWHERE


```
jharvard@appliance (~): sudo yum -y update
Password: crimson
```

- do this before you do anything with `submit50`!
  - you don't see your password, but you are indeed inputting it!