Password Generator

(pawg)

студенты группы ИС-142:
Григорьев Юрий, Наумов Алексей

Новосибирск 2022

Постановка задачи

Реализуемый продукт?

Генератор устойчивых паролей
с различными вариациями символов

Интерфейс?

Не имеется
(приложение для терминала/командной строки)

Методы работы с приложением?

Аргументы (флаги) при запуске программы

Формат входных данных

$ pawg **[N] [length] [args]**

N — количество паролей

length — длина каждого пароля

args — аргументы/флаги при запуске

Аргументы командной строки

Флаг	Расшифровка	Назначение
-c	capitalized	Включить в пароль хотя бы одну заглавную букву
-n	numeric	Включить в пароль хотя бы одну цифру
-y	special symbols	Включить в пароль хотя бы один спецсимвол
-s	secure	Сгенерировать защищённых пароль (alias: -c -n -y)

Аргументы командной строки

Флаг	Расшифровка	Назначение
-H -sha1	hash SHA-1	Использовать хеш SHA-1 заданного сида (строки) как генератор непроизвольных знаков
-1	column	Вывести пароли в столбик
-h	help	Вывести справку/помощь по программе

Options

```c
struct Option {
    int character_options;
    int numeric;
    int capitalized;
    int special;
    int size;
    int count;
    int column;
    int hash;
    char* seed;
};
```

```c
struct Option initOptions()
{
    struct Option option;
    option.size = 8;
    option.count = 1;
    option.numeric = 0;
    option.capitalized = 0;
    option.special = 0;
    option.column = 0;
    option.character_options = 0;
    option.seed = malloc(sizeof(char));
    return option;
}

struct Option getOptions(struct Option option, int argc, char** argv)
{
    if (argc > 1) {
        for (int i = 1; i < argc; i++) {
            if (strcmp(argv[i], "-c") == 0) {
                option.capitalized = 1;
            … < another arguments checking >
```

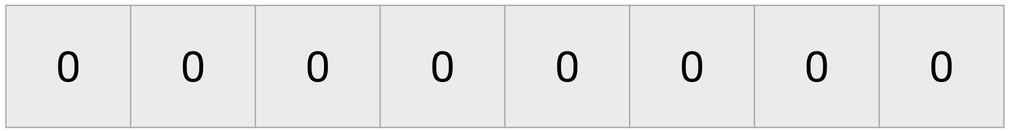
(перебор введённых аргументов)

Реализация

$ pawg

`char password[8]`

`int cells[8]`



Реализация

$ pawg

getLowercase()

getRandom(‘a’,’z’)

`char password[8]`

k							

`int cells[8]`

1	0	0	0	0	0	0	0

cell is filled

Реализация

$ pawg

one second later

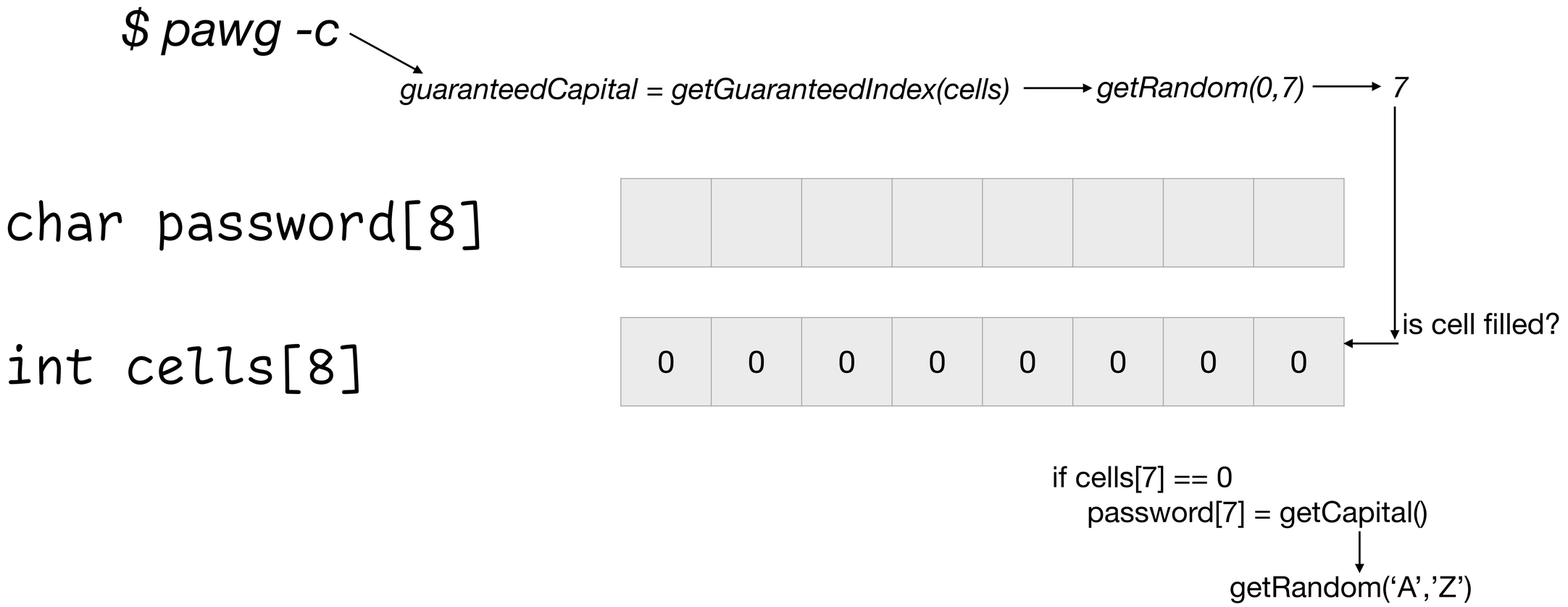
`char password[8]`

k	o	f	g	a	p	e	s

`int cells[8]`

1	1	1	1	1	1	1	1

Реализация


$ pawg -c
guaranteedCapital = getGuaranteedIndex(cells)
getRandom(0,7)
7
char password[8]
int cells[8]
0 0 0 0 0 0 0 0
is cell filled?
if cells[7] == 0
password[7] = getCapital()
getRandom('A','Z')

Реализация

$ pawg -c

```
char password[8]
```

```
int cells[8]
```

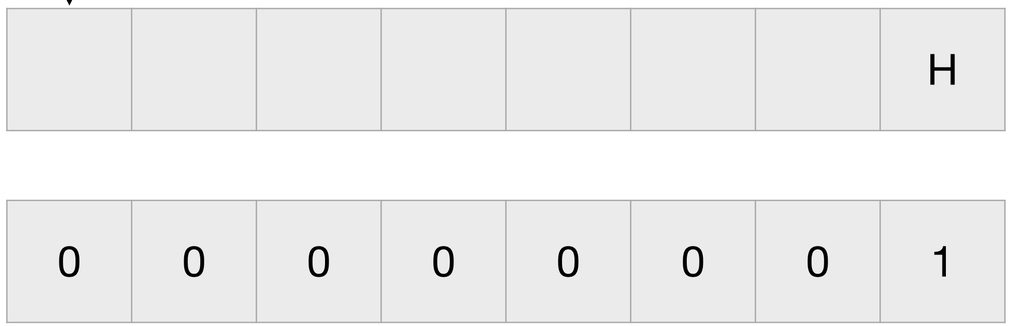
Реализация

$ pawg -c

```
char password[8]

int cells[8]
```

```
if cells[i] == 0
    getFromAlphabet(options)
```




```
charType = getRandom(0,1)

if charType == 1
    getCapital()
else
    getLowercase()
```

Реализация

$ pawg -c

result

`char password[8]`

B	t	y	J	a	a	l	H

`int cells[8]`

1	1	1	1	1	1	1	1

Реализация

$ pawg -s

($ pawg -c -n -y)

same thing

got the guaranteed

`char password[8]`

	6		!				H

`int cells[8]`

0	1	0	1	0	0	0	1

Реализация

$ pawg -s

($ pawg -c -n -y)

same thing

```
char password[8]
```

b	6		!				H

```
int cells[8]
```

1	1	0	1	0	0	0	1

```
charType = getRandom(0,3)

switch(charType)
case 1:
    getCapital()
case 2:
    getNumeric()
case 3:
    getSpecial()
default:
    getLowercase()
```

Реализация

$ pawg -s

($ pawg -c -n -y)

same thing

`char password[8]`

b	6	8	!	O	0	u	H

`int cells[8]`

1	1	1	1	1	1	1	1

Реализация

$ pawg 2 -sha1 hello

`char password[8]`

a	a	f	4	c	6	1	d

while not enough passwords {

char hash[40] = “hello”

hash = getHashFromString(hash) → “sha1.h” → MessageDigest[20][2]

(aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d)

for(i = 0; i < 8; i++)
password[i] = hash[i]

printf(“%s”, password)

}

Реализация

$ pawg 2 -sha1 hello

`char password[8]`

9	c	f	5	c	a	f	6

while not enough passwords {

char hash[40] = “aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d”

hash = getHashFromString(hash) → “sha1.h” → MessageDigest[20][2]

(9cf5caf6c36f5cccde8c73fad8894c958f4983da)

for(i = 0; i < 8; i++)

password[i] = hash[i]

printf(“%s”, password)

}

Сценарии использования

Run the -help flag

```
▶ Run bin/pawg -h

 Usage: pawg [N] [L] [args...]
        (to generate N passwords with L-digit length and specific options included in args)

 Arguments:
        -h        : display help
        -c        : include at least one uppercase letter
        -n        : include at least one number
        -y        : include at least one special character
        -s        : generate a secure password including all characters (alias for -c -n -y)
        -H <seed> : generate password based on sha1-hash of seed

```

Run program with invalid input

```
▶ Run bin/pawg -asd askdfj
can't recognise parameter -asd : skipping it
can't recognise parameter askdfj : skipping it
hvxblopa
```

Generate standard password

```
▶ Run bin/pawg
hvxblopa
```

Generate 1 password with capital letters

```
▶ Run bin/pawg -c
HvxblOQl
```

Generate 5 standard passwords

```
1  ▶ Run bin/pawg 5
4  hvxblopa sqkvxqsk sdkgvtzh cxstoirz bcwiwlwt
```

Generate 4 passwords in columnm, sized 13

```
1  ▶ Run bin/pawg 4 13 -1
4  hvxblopasqkvx
5  qsksdkgvtzhcx
6  stoirzbcwiwlw
7  tmfcqmvvnofvq
```

Generate 5 passwords sized 6 with capital letters, output in column

```
1  ▶ Run bin/pawg 5 6 -1 -c
4  HvxblO
5  QlpHas
6  VqBkFU
7  xqEPFk
8  dkgGYv
```

Generate 1 password including numbers

```
1  ▶ Run bin/pawg -n
4  v35xbl33
```

Generate 5 passwords with numbers and capital letters

```
1  ▶ Run bin/pawg 5 -n -c
4  v35xblO3 NH9a6034 0kFUFxqE KFksd1k6 14Qzh41Z
```

Generate 9 secure passwords, output in column

```
▶ Run bin/pawg 9 10 -s -1
3!%x!MP33o
*9a60"V4)x
"1-vENqE7/
sd1!j6bG14
)9%VZ8l+,E
oi/r7,6$bc
M7K%wl.w95
%&KXLPc$%&
&0+OA.S4Ov
```

Generate 5 passwords with special symbols

```
▶ Run bin/pawg 5 5 -y
%&%x! *op#! .gs"" q)($k vx)ps
```

Generate full-length sha-1 (40) hash password using seed "hello"

```
▶ Run bin/pawg 1 40 -H hello
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
```

Generate 5 hash-based segmented (size 20) passwords using seed "scrubs"

```
▶ Run bin/pawg 5 20 -sha1 scrubs
89cdb34afa779741f55d 351f5802c7a21905504f da5bd315de38b51bc8ac d8a6e8965592a627271f f6a03b4aa3a517ff5823
```

Unit-тесты

```
TEST 1/24 check_random:valid [OK]
TEST 2/24 check_random:invalid [OK]
TEST 3/24 check_isnumber:valid [OK]
TEST 4/24 check_isnumber:invalid [OK]
TEST 5/24 check_initoptions:valid [OK]
TEST 6/24 check_initoptions:invalid [OK]
TEST 7/24 check_getoptions:valid [OK]
TEST 8/24 check_getoptions:invalid can't recognise parameter abc : skipping it
[OK]
TEST 9/24 check_sha1:valid [OK]
TEST 10/24 check_sha1:invalid [OK]
TEST 11/24 check_getlowercase:valid [OK]
TEST 12/24 check_getlowercase:invalid [OK]
TEST 13/24 check_getcapital:valid [OK]
TEST 14/24 check_getcapital:invalid [OK]
TEST 15/24 check_getspecial:valid [OK]
TEST 16/24 check_getspecial:invalid [OK]
TEST 17/24 check_getnumber:valid [OK]
TEST 18/24 check_getnumber:invalid [OK]
TEST 19/24 check_getfromalphabet:valid1 [OK]
TEST 20/24 check_getfromalphabet:valid2 [OK]
TEST 21/24 check_getfromalphabet:invalid1 [OK]
TEST 22/24 check_getfromalphabet:invalid2 [OK]
TEST 23/24 check_getguaranteedindex:valid [OK]
TEST 24/24 check_getguaranteedindex:invalid [OK]
RESULTS: 24 tests (24 ok, 0 failed, 0 skipped) ran in 0 ms
```

Unit-тесты

```
 1 #include "random.h"
 2 #include "input.h"
 3 #include <ctest.h>
 4
 5 CTEST(check_random, valid)
 6 {
 7     int value = getRandom(0, 100);
 8     int result = value >= 0 && value <= 100;
 9     int expected = 1;
10     ASSERT_EQUAL(expected, result);
11 }
12
13 CTEST(check_random, invalid)
14 {
15     int value = getRandom(0, 100);
16     int result = value <= 0 || value >= 100;
17     int expected = 1;
18     ASSERT_NOT_EQUAL(expected, result);
19 }
20
```

```
21 CTEST(check_isnumber, valid)
22 {
23     int result = isNumber("1000");
24     int expected = 0;
25     ASSERT_EQUAL(expected, result);
26 }
27
28 CTEST(check_isnumber, invalid)
29 {
30     int result = isNumber("AVC");
31     int expected = 1;
32     ASSERT_EQUAL(expected, result);
33 }
```

Guards

```
 1 if (option.hash == 1 && option.size > 40) {
 2         printf("max hash password length equals to 40 : letters after
   40'th won't generate\n");
 3     }
 4
 5 if (strcmp(argv[i], "-H") == 0 || strcmp(argv[i], "-sha1") == 0) {
 6     if (!argv[i + 1]) {
 7         printHelp();
 8         option.size = 0;
 9     }
10 }
11
12 if (option.character_options > option.size) {
13     printf("size of password is less than character options you've
   specified\n");
14     option.size = 0;
15 }
16
17 else {
18     printf("can't recognise parameter %s : skipping it\n",
19     argv[i]);
20 }
```

Разбиение задач

Григорьев Юрий	Наумов Алексей
Постановка задач на проект	Техническое задание
Разработка архитектуры приложения (filling cells array / guaranteed-index method)	Аргументы командной строки
Структурирование проекта (external modules, thirdparty libraries)	Разработка генерации паролей с цифрами
Разработка Makefile	Разработка функций подбора случайного символа
Разработка генерации паролей, основанных на хеш-функции SHA-1	Bug fixes
Разработка генерации символов из заданных алфавитов (a-z, A-Z, 0-9, !-/)	Unit-тесты
Codestyle	Multiplatform-тесты
	Разработка сценариев для автоматических сборок в GitHub Actions

Спасибо за внимание!