

# 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 защищённый 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 произвольных знаков
-l	column	Вывести пароли в столбик
-h	help	Вывести справку/помощь по программе

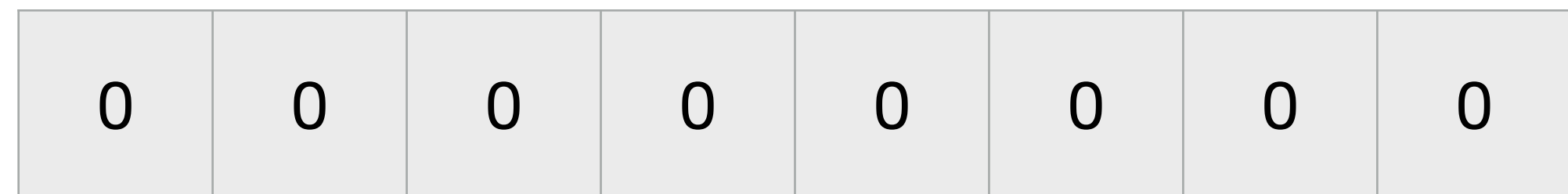
# Реализация

*\$ pawg*

`char password[8]`



`int cells[8]`



# Реализация

*\$ pawg*

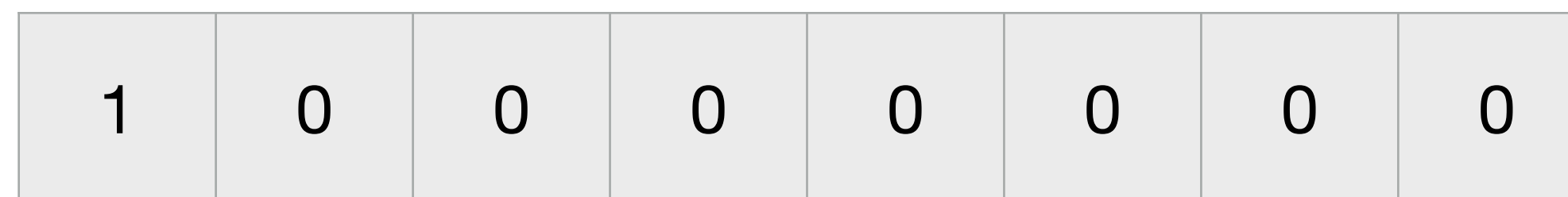
`char password[8]`

`int cells[8]`

getLowercase()



getRandom('a','z')



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

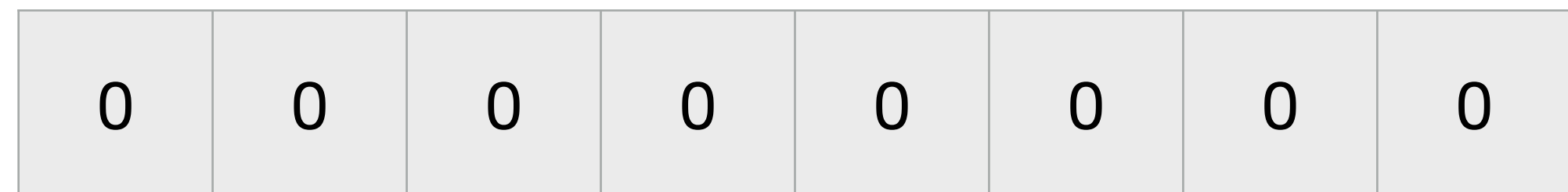
*\$ pawg -c*

*guaranteedCapital = getGuaranteedIndex(cells) → getRandom(0,7) → 7*

*char password[8]*



*int cells[8]*



is cell filled?

if *cells[7] == 0*  
    *password[7] = getCapital()*

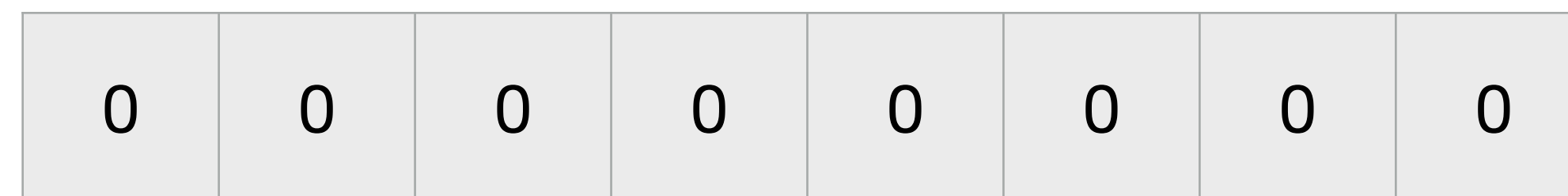
*getRandom('A','Z')*

# Реализация

*\$ pawg -c*

```
char password[8]
```

```
int cells[8]
```



password[7] = getCapital()

getRandom('A','Z')

if cells[7] == 0

is cell filled?

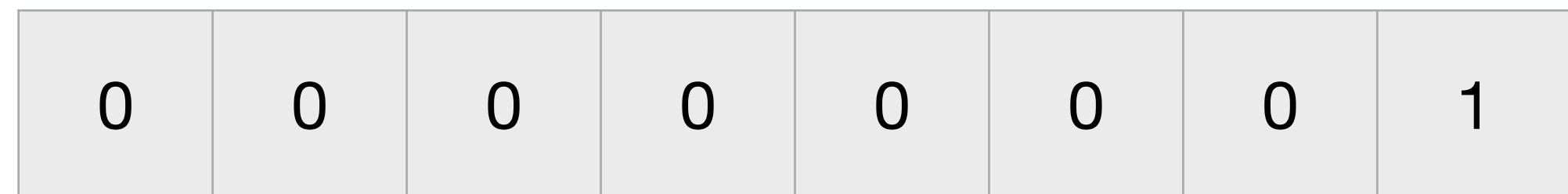
# Реализация

*\$ 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]`

`int cells[8]`

b	6		!				H
---	---	--	---	--	--	--	---

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]`

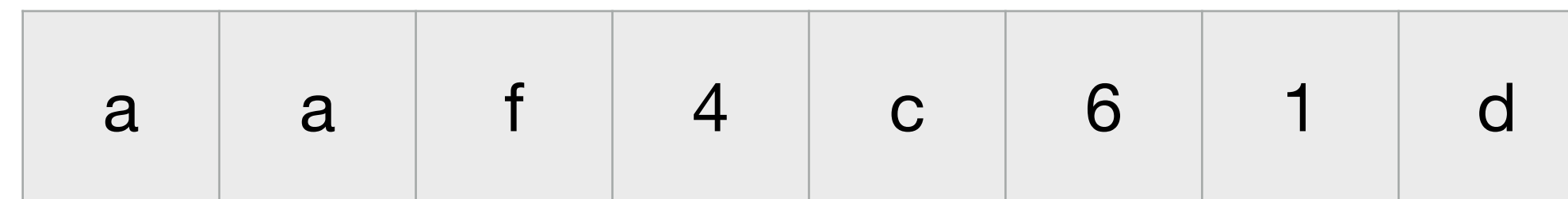
1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---



# Реализация

*\$ pawg 2 -sha1 hello*

char password[8]



**while not enough passwords {**

char hash[40] = "hello"

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

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

password[i] = hash[i]

printf("%s", password)

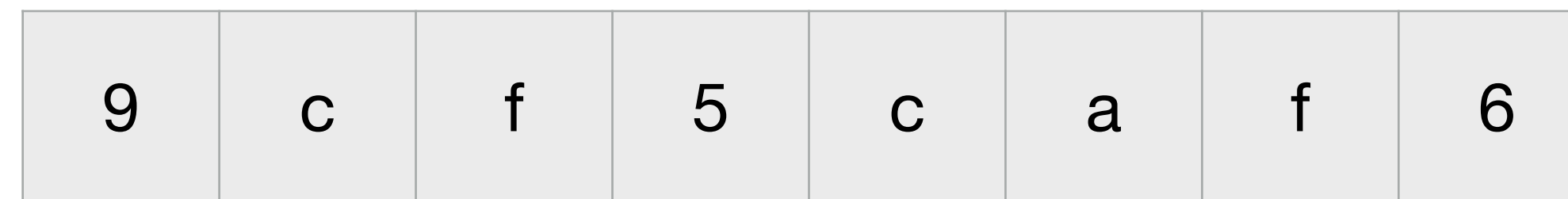
**}**

(aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d)

# Реализация

*\$ pawg 2 -sha1 hello*

char password[8]



**while not enough passwords {**

char hash[40] = "aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d"

hash = getHashFromString(hash) —————> "sha1.h" —————> MessageDigest[20][2]

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

password[i] = hash[i]

printf("%s", password)

**}**

(9cf5caf6c36f5cccde8c73fad8894c958f4983da)

# Сценарии использования

✓ Run the -help flag

```
1 ▶ Run bin/pawg -h
4
5 Usage: pawg [N] [L] [args...]
6      (to generate N passwords with L-digit length and specific options included in args)
7
8 Arguments:
9      -h      : display help
10     -c      : include at least one uppercase letter
11     -n      : include at least one number
12     -y      : include at least one special character
13     -s      : generate a secure password including all characters (alias for -c -n -y)
14     -H <seed> : generate password based on sha1-hash of seed
15
```

✓ Run program with invalid input

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

✓ Generate standard password

```
1 ▶ Run bin/pawg
4 hvxblopa
```

✓ Generate 1 password with capital letters

```
1 ▶ Run bin/pawg -c
4 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 hvxblopa sqkvx
5 qksdkgvtzhcx
6 stoirzbcwiwlw
7 tmfcqmvvnofvq
```

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

```
1 ▶ Run bin/pawg 5 6 -1 -c
4 Hvxb10
5 Q1pHas
6 VqBkFU
7 xqEPFk
8 dkgGYv
```

✓ Generate 1 password including numbers

```
1 ▶ Run bin/pawg -n
4 v35xb133
```

✓ Generate 5 passwords with numbers and capital letters

```
1 ▶ Run bin/pawg 5 -n -c
4 v35xb103 NH9a6034 0kFUFxqE Kfksd1k6 14Qzh41Z
```

✓ Generate 9 secure passwords, output in column

```
1 ▶ Run bin/pawg 9 10 -s -1
4 3!%x!MP33o
5 *9a60"V4)x
6 "1-vENqE7/
7 sd1!j6bG14
8 )9%VZ8l+,E
9 oi/r7,6$bc
10 M7K%wL.w95
11 %&KXLPc$%&
12 &0+0A.S40v
```

✓ Generate 5 passwords with special symbols

```
1 ▶ Run bin/pawg 5 5 -y
4 %&%x! *op#! .gs"" q)($k vx)ps
```

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

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

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

```
1 ▶ Run bin/pawg 5 20 -sha1 scrubs
4 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
```

# Разбиение задач

Григорьев Юрий

Наумов Алексей

- Постановка задач на проект
- Разработка архитектуры приложения  
(filling cells array / guaranteed-index method)
- Структурирование проекта  
(external modules, thirdparty libraries)
- Разработка Makefile
- Разработка генерации паролей, основанных на хеш-функции SHA-1
- Разработка генерации символов из заданных алфавитов (a-z, A-Z, 0-9, !-/)
- Codestyle

- Техническое задание
- Аргументы командной строки
- Разработка генерации паролей с цифрами
- Разработка функций подбора случайного символа
- Bug fixes
- Unit-тесты
- Multiplatform-тесты
- Разработка сценариев для автоматических сборок в GitHub Actions



**Спасибо за внимание!**