Junioraufgabe 1: **Passwörter**

Team-ID: **00255**

Team: **Solo**

Bearbeiter dieser Aufgabe: **Walther Trgovac**

15. November 2020

Inhaltsverzeichnis

Lösungsidee

Das Ziel der Aufgabe ist es, ein Passwort zu generieren und die Merkbarkeit des Passworts soll so gut sein wie möglich. Diese Methode, die ich jetzt vorstellen werde, verwende ich seit Jahren für meine Passwörter, aber ich habe noch nie ein Programm geschrieben, das diese Methode verwendet und so generiert meine Passwörter. Die folgenden Regeln gelten für meine Idee:

Normale Wörter

Die Grundlage dieser Methode sind die normalen Wörter. Das Einfachste, was man sich am besten merken kann, sind, meiner Meinung nach, die normalen Wörter. Weil nur normale Wörter als Passwort sehr schwach sind, werde ich mein Passwort noch verstärken.

Wörter reimen sich

Den Text eines Liedes kann man sich einfach merken, weil die Wörter sich reimen. Deswegen macht mein Programm das Selbe. 2 Wörter werden zufällig ausgesucht und die reimen sich.

Mit Hilfe des Alphabets wird das Passwort stärker

Jeder weiß auswendig, wie das Alphabet lautet, zumindest die ersten paar Buchstaben, die man in diesem Fall am meisten braucht. Nach jedem Buchstaben “A”, “E”, “I”, “O” oder “U” (Selbstlaute) wird ein Buchstabe aus dem Alphabet in der richtigen Reihenfolge ausgesucht und addiert nach diesen Buchstaben. Falls der erste Buchstabe im Passwort “A” ist, wird einfach der nächste Buchstabe aus dem Alphabet ausgesucht und dazu addiert, damit es nicht zu Doppel-Buchstaben kommt.

Das Passwort am Ende sieht sehr komisch aus, aber das ganze Passwort macht Sinn, solange man die Methode kennt. Die Buchstaben aus dem Alphabet sind alle groß, damit das Passwort schwieriger zu knacken ist. Jedes Wort ist auch mindestens 6 Buchstaben lang, damit das Passwort nicht zu kurz ist.

Umsetzung

Das Programm habe ich in C++ geschrieben. Im Internet habe ich ein Wörterbuch als Textdatei gefunden und mit Hilfe von ihm finde ich die passenden Wörter. Das Programm funktioniert mit jedem solchen Wörterbuch. In diesem Fall habe ich mich für ein Wörterbuch mit englischen Wörtern entschieden.

Erstmal werden alle Wörter in einen C++ Vector gespeichert. Dann wird das erste zufällige Wort aus dem Wörterbuch ausgesucht und dem String “Passwort” addiert. Das zweite danach, aber unter der Bedingung, dass die letzten drei Buchstaben gleich sind wie bei dem ersten Wort. Inzwischen wird auch überprüft, ob ein Wort Doppel-Buchstaben hat, falls ja, wird ein neues Wort gesucht.

Danach wird der String Passwort in einen Vector gespeichert, damit man mit Hilfe der Funktion “insert” den richtigen Buchstaben aus dem Alphabet an die richtige Position addieren kann.

Beispiele

Die Grundlage des Passworts lautet: moderators,regulators

Das komplette Passwort lautet: **moAdeBraCtoDrsreFguGlaHtoIrs**

Die Grundlage des Passworts lautet: faltering,inverting

Das komplette Passwort lautet: **faBlteCriDngiEnveFrtiGng**

Die Grundlage des Passworts lautet: activated,subverted

Das komplette Passwort lautet: **aBctiCvaDteFdsuGbveHrteId**

Die Grundlage des Passworts lautet: inscription,conjunction

Das komplette Passwort lautet: **iAnscriBptiCoDncoEnjuFnctiGoHn**

Die Grundlage des Passworts lautet: personalised,familiarised

Das komplette Passwort lautet: **peArsoBnaCliDseFdfaGmiHliJaKriLseMd**

Quellcode

```
//Ein Vector, um Wörter zu speichern
vector <string> v;
for (int i = 0; i < 64622; i++){
	string s;
	getline(cin, s);
	v.push_back(s);
}
```

```
// Loop, um die richtigen Woerter zu finden
while (ok){ /
	// Der erste Fall, das erste Wort wird gesucht
	if (cnt == 0){
	string wort1 = v[uni(rng)];
	// Ueberpruefen, ob es Doppel-Buchstaben im Wort gibt, falls ja, ein
Neues finden
	bool repeat = false;
	for (int i = 0; i < wort1.size()-1; i++){
		if (wort1[i] == wort1[i+1]){
			repeat = true;
		}
	} if (!repeat){
		// Hier gibt es ein paar Bedinungen für das erste Wort
		// Das Wort muss länger als 6 Buchstaben sein
		// In meinem Worterbuch, in dem sich die Wörter befinden, aus
		irgendwelchem Grund gibt es am Ende mancher Wörter Zeichen wie
		"+", "&", "^"...
		// Diese Wörter werden ignoriert
		if (wort1.size() > 6 && wort1[wort1.size()-1] != '&' &&
	wort1[wort1.size()-1] != '+' && wort1[wort1.size()-1] != '$' &&
	wort1[wort1.size()-1] != '^'){
			passwort+=wort1;
			passwort+=',';
			wort1copy = wort1;
			cnt++;
		}
		}
	}
	// Der zweite Fall, das zweite Wort wird gesucht
	if (cnt == 1){
	string wort2 = v[uni(rng)];
	// Ueberpruefen, ob es Doppel-Buchstaben im Wort gibt, falls ja, ein
	Neues finden
	bool repeat = false;
	for (int i = 0; i < wort2.size()-1; i++){
		if (wort2[i] == wort2[i+1]){
			repeat = true;
		}
```

```
	} if (!repeat){
		// Das zweite Wort ist genauso lang wie das Erste und die
	letzen drei Buchstaben des zweiten Wortes entsprechen den drei letzen
	Buchstaben des ersten Wortes
		if (wort2.size() == wort1copy.size() && wort2[0] !=
		wort1copy[wort1copy.size()-1] && wort2 != wort1copy &&
		wort2[wort2.size()-1] == wort1copy[wort1copy.size()-1] &&
		wort2[wort2.size()-2] == wort1copy[wort1copy.size()-2] &&
		wort2[wort2.size()-3] == wort1copy[wort1copy.size()-3]){
		passwort+=wort2;
		ok = false; }
		}
	}
	// Falls es zu viele Versuche gibt, die richtigen Wörter zu finden,
	wird der Loop wiederholt
	versuche++;
	if (versuche > 500){
		versuche = 0;
		cnt = 0;
		passwort = "";
	}
}



// Loop, um die richtigen Buchstaben zu addieren
for (int i = 0; i < passwortv.size(); i++){
	if (passwortv[i] == 'a' || passwortv[i] == 'e' || passwortv[i] == 'i'
	|| passwortv[i] == 'o' || passwortv[i] == 'u'){ if (alphabet[cur] ==
	passwortv[i]){
		// Falls der selbe Buchstabe addiert werden soll, wird der
		naechste Buchstabe addiert, damit es nicht zu Doppel-Buchstaben
		kommt
		passwortv.insert(passwortv.begin()+i+1,
		toupper(alphabet[cur+1]));
		cur+=2; }
	else {
	// Falls nicht, einfach einen Buchstaben addieren, der Buchstabe wird
	mit Hilfe der Variable "cur" ausgesucht
		passwortv.insert(passwortv.begin()+i+1,
	toupper(alphabet[cur]));
		cur++; }
	}
}
```