C++ DAP2 Praktikum für ETIT und IKT, SoSe 2018, Kurzaufgabe K1

Fällig am 16.04.2018 um 18:00

Kriterien für eine erfolgreiche Teilnahme am Praktikum:

- Abwesenheit an maximal einem der Termine Nr. 1-12, unabhängig ob entschuldigt oder unentschuldigt.
- Erreichen von in Summe 6 von 12 Punkten bei den 6 Kurzaufgaben
- Erreichen von in Summe 6 von 12 Punkten bei den 6 Langaufgaben

Anmerkungen zum Ablauf des Praktikums

- Verspätungen über 10 Minuten werden als „nicht anwesend“ gewertet.
- Nur bei Anwesenheit können Sie Punkte erreichen.
- Die erfolgreiche Bearbeitung einer Aufgabe besteht aus
    - der selbständigen Programmierung in akzeptablem Programmierstil,
    - der erfolgreichen Übersetzung durch den Compiler,
    - dem erfolgreichen Aufruf des Programms,
    - den richtigen Ausgaben des Programms (auch bei von der Aufgabenstellung abweichenden Aufrufparametern),
    - der sinnvollen Reaktion des Programms auch auf unsinnige Aufrufparameter,
    - der Verwendung von Schutzmechanismen, die unvorhergesehene Zustände des Programms handhaben,
    - dem Erklären des Programmtextes und
    - der Beantwortung von Fragen zum Programm.
- Für den Test des Programms sind die Teilnehmer selbst verantwortlich.
- Kopierte (Teil-)Lösungen oder die Weitergabe eigener Lösungen können nach Maßgabe des Betreuers zum Punktentzug führen.
- Probleme beim Erklären des eigenen Programmtextes führen zum Punktabzug.
- Langaufgaben dürfen und sollen auch außerhalb der Praktikumszeit bearbeitet werden. Langaufgaben werden zum Fälligkeitstermin ab Beginn des Praktikums kontrolliert.
- Es gibt für jedes Aufgabenblatt nur eine Kontrolle. Ein „iteratives“ Kontrollieren ist nicht vorgesehen.
- Kurzaufgaben werden am Tag der Ausgabe kontrolliert.
- Die Kontrolle der Kurzaufgaben erfolgt in Reihenfolge der Anzeige der Fertigstellung. Konnte Ihre Lösung nicht bis zum Ende der Praktikumszeit kontrolliert werden, so erhalten Sie keine Punkte. Beeilen Sie sich daher mit der Lösung um zu verhindern, dass Sie hinten in der Reihenfolge stehen und dem Betreuer keine Zeit verbleibt, Ihre Lösung zu kontrollieren.
- Je nach Auslastung werden die Aufgaben allein oder in 2er-Gruppen bearbeitet.
- Die Teilnehmer einer 2er-Gruppe erhalten dieselbe Punktzahl für Ihre Lösung.
- Es wird nur eine Lösung für jede 2er-Gruppe kontrolliert.
- Der Betreuer nimmt die Gruppeneinteilung vor.
- Da das Praktikum eine Vorlesung und deren Übung begleitet, führen wir keine Kontrolle auf Vorbereitung auf das Praktikum durch Befragung der Studieren-

den durch. Sie sind erwachsene Menschen. Bitte bereiten Sie sich selbständig durch die Vorlesung und die Übungen vor.
- Für die Aufgabenblätter und Quelltexte, die Sie als Gerüst zur Erledigung Ihrer Aufgabe erhalten, besitzen entweder die Fak. ETIT und/oder die Fak. Informatik der TU-Dortmund das Recht zur Vervielfältigung und Nutzung. Sie dürfen das Gerüst für sich persönlich nur für das Praktikum DAP2 des SoSe2018 nutzen. Eine Weitergabe an Dritte oder eine Speicherung auf Medien, auf die Dritte Zugriff haben könnten, ist nicht erlaubt und verpflichtet Sie zur Zahlung von Schadensersatzanspruch. Mit Teilnahme am Praktikum erklären Sie sich hiermit einverstanden.
- Bei Kurzaufgaben können Sie das Praktikum nach Kontrolle Ihrer Lösung verlassen.
- Bei den Langaufgaben bleiben Sie bitte bis zum Ende der Praktikumszeit anwesend.
- Es wird in C/C++ unter Microsoft Visual Studio programmiert.

Folgende Termine sind vorgesehen, es können sich aber Änderungen ergeben:

Termin Nr.	Datum	Ausgabe und Kontrolle	Ausgabe	Kontrolle
1	16.04.	Kurzaufgabe K1		
2	23.04.		Langaufgabe L1	
3	30.04.	Kurzaufgabe K2		
4	07.05.		Langaufgabe L2	Langaufgabe L1
5	14.05.	Kurzaufgabe K3		
	21.05.	Pfingsten		
6	28.05.		Langaufgabe L3	Langaufgabe L2
7	04.06.	Kurzaufgabe K4		
8	11.06.		Langaufgabe L4	Langaufgabe L3
9	18.06.	Kurzaufgabe K5		
10	25.06.		Langaufgabe L5	Langaufgabe L4
11	02.07.	Kurzaufgabe K6	Langaufgabe L6	
12	09.07			Langaufgabe L5
13	16.07			Langaufgabe L6

Regeln für die Programmierung

Nachfolgende Regeln gelten (sofern nichts anderes gesagt wird) auch für den Rest des Praktikums. Wenn das Programm diesen Bedingungen nicht genügt, erfolgt Punktabzug.

- Alle Programme dieses Praktikums sind nicht interaktiv. Sie werden in einer Konsole (aka „Eingabeaufforderung“ „`cmd`“) mit entsprechenden Parametern gestartet. Nach dem Aufruf erfolgen keine weiteren Eingaben.
- Der Aufruf eines Programms mit fehlenden oder mit falschen Parametern soll eine sinnvolle Fehlermeldung erzeugen.
- Die Reihenfolge der Schalter in den Argumenten soll beliebig sein.
- Programme liefern am Ende einen *exit code* ab. Dieser *exit code* hat im Erfolgsfall den Wert *0*, im Fehlerfall wollen wir eine *1* zurückliefern. Dazu kann

im Fehlerfall das Programm mit `exit(1)` beendet werden. Ist alles korrekt, endet das Hauptprogramm `main` mit `return 0;`

- Trennen Sie grundsätzlich die Auswertung der Aufrufparameter sowie die Ausgaben des Programms vom eigentlichen Algorithmus oder den Teilalgorithmen, den/die Sie als Unterprogramm realisieren.
- In allen zu erstellenden Unterprogrammen oder Klassen darf nichts mit `cout` oder `cerr` ausgegeben werden. Unterprogramme und Klassen erledigen still ihre Arbeit. Nur das Hauptprogramm macht Ausgaben auf die Konsole, wenn nicht ausdrücklich erwähnt wurde, dass die Klasse eine Ausgabe erzeugen soll.
- Falls sich während eines Unterprogrammes bzw. Methodenaufrufes herausstellt, dass sich durch die Aufrufparameter ein Fehler ergibt, so wird eine Exception erzeugt, die das aufrufende Programm handhabt und ggfs. den Benutzer informiert. Ein Unterprogramm bzw. eine Methode beendet (Ausnahme `assert`) nie das Programm.
- Die zu implementierenden Programme verwenden `assert` als letzte Notbremse um einen fehlerhaften Programmablauf mit Gewalt augenblicklich zu beenden. Beispiel in der heutigen Aufgabe: Obwohl das Hauptprogramm den Eingabestring übergeben sollte, überprüft das Unterprogramm mit `assert(Input != 0)` dass keine Nullpointer übergeben werden (Test der *Preconditions*).
- Überprüfen Sie mit `assert` auch, ob die Ergebnisse korrekt sind. Beispiel: Bei einem Sortierprogramm sollten Sie *nach* dem Sortieren überprüfen, dass das jeweils folgende Element größer oder gleich dem aktuellen ist. Falls diese Überprüfung fehlschlägt, so ist der Algorithmus offenbar defekt, und Sie beenden das Programm über `assert` (Test der *Postconditions*).
- Im Idealfall überprüfen Sie auch die *Invarianten* mit `assert`. Dies sind Bedingungen, die auch nach (oder sogar während) der Ausführung eines Algorithmus eingehalten werden müssen. Bei einem Sortierverfahren könnte dies entsprechen, das Sie nach Ablauf des Algorithmus die Summe der Werte aller Elemente berechnen. Diese sollte sich im Vergleich zum Wert vor dem Sortiervorgang nicht ändern.
- `assert` ersetzt keine sinnvolle Fehlerbehandlung mit Information des Benutzers über den (Eingabe-) Fehler. `assert` ist nämlich nur in der Debug-Release aktiv! Es soll nur verhindert werden, dass ein scheinbar korrektes Programm unbemerkt falsche Ergebnisse erzeugt.
- Zum Test sollten Sie (zur Not durch temporäre Modifikation des Programmtextes) während der Debug-Phase entsprechende Fehler provozieren.
- Bei dynamisch alloziertem Speicher ist der Erfolg zu überprüfen. Hier gibt es zwei Versionen: Im Normalfall wird bei fehlgeschlagener Allokation unter C++ eine Exception erzeugt, außer man erzwingt das klassische Verhalten durch den Zusatz `(nothrow)` beim `new` . Für uns bedeutet dies, dass wir dynamischen Speicher entweder „klassisch“

```
char * wurst = new (nothrow) char[size_t(n)];
if (0==wurst) throw "No Memory";
```

oder mit

```
char *wurst;
try { wurst=new char[size_t(n)]; }catch(...){ throw "No Memory"; };
```

überprüfen.^Mitlerweile wird die zweite Version bevorzugt.

- Die zu implementierenden Unterprogramme dürfen keine (potentiellen) Speicherlöcher erzeugen.
- C-typische Konstrukte wie `open`, `fopen`, `sscanf`, `scanf`, `atoi`, `printf`, `fprintf` sind verboten.
- Verwenden Sie sinnvoll Kommentare. Falsche Kommentare sind schlimmer als keine Kommentare. Schreiben Sie keine Selbstverständlichkeiten in Kommentare. Sinnvolle Variablen sind besser als schlechte Kommentare.

```
i=i+1; // Increase i by 1                                  <- Bad!

ai=(i>0)?i:0; // ai is absolute value of i                 <- Worse!

X3=x1*x2; x_4=sqrt(X3); // geometric mean of x1 and x2 <- Obfuscated

GeometricMean = sqrt(x1*x2); <- Much better! Needs no comment at all.

/*--------------------------------------*/
/* main starts here                     */               <- Bad
/*--------------------------------------*/
int main( ...

// See: http://en.wikipedia.org/wiki/Volume (of Sphere) <- Good
Volume = 4.0/3.0*M_PI*pow(Radius,3);
```

Allgemeine Hilfestellungen

- Die Feldbreite bei der Ausgabe kann mit `setw` gesteuert werden. Beispiel:

```
cout << setw(4) << 14   << setw(4) << 1   << endl
     << setw(4) << "XX" << setw(4) << 104 << endl;
```

ergibt die vertikal übersichtlich angeordnete Ausgabe

```
  14   1
  XX 104
```

- Für die Formatierung von Fließkommazahlen bei der Ausgabe über einen C++-Stream gibt es zwei nützliche Funktionen, nämlich `setiosflags` und `setprecision`, welche beide direkt in den Ausgabe-Stream eingefügt werden. Mit `setiosflags` kann man wissenschaftliche (z.B. `1.2e+000`) oder Festpunktnotation (z.B. `1.2`) wählen, und mit `setprecision` die Präzision der Ausgabe angeben. Bei Wahl von wissenschaftlicher oder Festpunknotation gibt die Präzision die Anzahl der Nachkommastellen an. Beispiele

```
// Standard mit 3 signifikanten Stellen
// Ausgabe: 1.2
cout << setprecision(3) << 1.2 << endl;

// Festpunktnotation mit 3 Nachkommastellen
// Ausgabe: 1.200
cout << setiosflags(ios::fixed) << setprecision(3) << 1.2 << endl;

// Flag ios::fixed wieder löschen
```

```
cout << resetiosflags(ios::fixed);

// wissenschaftliche Notation mit 3 Nachkommastellen
// Ausgabe: 1.200e+000
cout << setiosflags(ios::scientific) << setprecision(3) << 1.2 <<
endl;
```

- Zeitmessungen können Sie auf folgende Weise ausführen:

```
double MeasuredTime = double(clock());
// Calculate something...
MeasuredTime=(double(clock())-MeasuredTime)/CLOCKS_PER_SEC;
```

- Machen Sie sich ein System zum Parsen der Aufrufparameter, da Sie dieses in Zukunft immer wieder benötigen werden. Eine Möglichkeit ist es, folgendes Schema zu erweitern:

```
char *SomeString=0;
int SomeInteger;
bool SomeFlag=false;

for(int i=1;i<argc;i++)  { // parse options
      if (!string(argv[i]).compare("-t"))  {
            // option "t" is active!
            SomeFlag=true;
            continue;
      }
      if (!string(argv[i]).compare("-l"))  {
            // option "l" is active,
            // but an integer must follow option -l
            if (++i == argc )
                  throw  "-l Must be followed by some integer." ;
            if (!(istringstream(argv[i]) >> dec >> SomeInteger))
                  throw "Malformed number for SomeInteger." ;
            continue;
      }
      SomeString = argv[i]; // if you trickle down here,
                            // this must be SomeString
} // for
```

Schauen Sie nach: Für die heutige Aufgabe passt das obige ganz gut!

Kurzaufgabe K1.1 (1,5 Punkte)

Schreiben Sie ein Programm, das eine einfache polyalphabetische Substitutionsverschlüsselung durchführt. Der Algorithmus ist an die Enigma-M3 angelehnt. Allerdings verwendet er nur eine – statt wie beim Original drei – Substitutionsrunden, was ihn zum kryptografischen Fingerspiel degradiert. Als Substitutionstabelle verwendet er die „Walze VI“ der Enigma M3. Immerhin vermeidet der Algorithmus die involutorische Verschlüsselung (beim Entschlüsseln muss daher eine andere Walze verwendet werden). Die involutorische Verschlüsselung war ein Schwachpunkt der Enigma, der das zuverlässige Brechen der Schlüssel ermöglichte.

Die beiden Zahlen `InitialRotation` und `Rotation` stellen den geheimen Schlüssel dar. Der Algorithmus wird durch diesen Pseudocode erklärt:

```
Offset ← InitialRotation;

// Original       ABCDEFGHIJKLMNOPQRSTUVWXYZ
Table[0..25] ← "JPGVOUMFYQBENHZRDKASXLICTW" // Walze No. VI


for i in all Characters Positions of Input
       ActualTable ← Rotate(Table by Offset);
       C ← Input[i];
       //Replace C with Substitute from ActualTable:
       R ← ActualTable[ C – 'A' ];
       Output[i] ← R;

       Offset ← Offset+Rotation;

endfor
```

Die Funktion **`Rotate`** wird durch folgende Tabelle erläutert:

Function Call	ActualTable
...	...
Rotate(Table by -2)	TWJPGVOUMFYQBENHZRDKASXLIC
Rotate(Table by -1)	WJPGVOUMFYQBENHZRDKASXLICT
Rotate(Table by 0)	JPGVOUMFYQBENHZRDKASXLICTW
Rotate(Table by 1)	PGVOUMFYQBENHZRDKASXLICTWJ
Rotate(Table by 2)	GVOUMFYQBENHZRDKASXLICTWJP
...	...
Rotate(Table by 25)	WJPGVOUMFYQBENHZRDKASXLICT
Rotate(Table by 26)	JPGVOUMFYQBENHZRDKASXLICTW
Rotate(Table by 27)	PGVOUMFYQBENHZRDKASXLICTWJ
...	...
… and so on …	

Bitte beachten Sie:

- Ihr Programm soll `k1` heißen.
- Der Ver- und Entschlüsselungsalgorithmus wird als Unterprogramm
  ```
  char *Crypt( char *Input,
               bool Encrypt,
               int InitialRotation,
               int Rotation )
  ```
  realisiert. Zunächst wird nur die Verschlüsselung implementiert.
- Das Flag `Encrypt` wählt bei `true` die Verschlüsselung aus, ansonsten die Entschlüsselung.
- Der `Input` wird als als `char*` übergeben. Beachten Sie, dass solche char-Strings mit 0 terminiert sein müssen.
- Mit dem optionalen Schalter `–i InitialRotation` und `–r Rotation` können die beiden Schlüsselkomponenten übergeben werden.
- Mit dem optionalen Schalter `–t` kann die Laufzeit des Algorithmus' (und nur des Algorithmus', das „Beiwerk" wird nicht berücksichtigt) gemessen werden.

- Mit dem optionalen Schalter `-d` wird von Verschlüsseln auf Entschlüsseln umgeschaltet.
- Der Aufruf folgt also der Konvention

```
k1 <string Input> [-t][-d][-i <int Initial Rotation>][-r <int Rotation>]
```

Bespiele:

```
k1 <Return>
Usage: k1 [ -i <int Rotation> ] [ -i <int InitialRotation> ] [ -t ]  [-d ]
<string Input>

Simple Encryption.

Options:
-i InitialRotation: initial rotation the codebook
-r Rotation       : rotation of codebook after every Letter
-d                : decrypt (instead of encrypt)
-t                : Calculate processing time

Options may occur in any order.

k1 "AAAA"
JJJJ

k1 "ZzZZ" -i 1 -r 4
Invalid Chars in Input. Use only A-Z.

k1 "ZZZZ" -i 1 -r 4
JOYN

k1 "JOYN" -i 1 -r 4 -d
ZZZZ
```

Gerüst für k1

Eine mögliche Struktur für `k1` ist nachfolgend gezeigt. Natürlich sind andere Lösungen möglich.

```
#include <iostream>     // headers may maybe oversized
#include <fstream>      // for this example
#include <sstream>
#include <cstdlib>
#include <cstring>
#include <assert.h>
#include <ctime>
using namespace std;

// Code is yet not able to decrypt!

char *Crypt(      char *Input,
                  bool Encrypt,
                  int InitialRotation,
                  int Rotation)
{

      -> Missing Code for memory allocation and sanity checks

      int Offset=InitialRotation;
```

```
	for (int i=0;i<strlen(Input);i++) { //iterate input

		if (Input[i]<'A' || Input[i]>'Z')
			throw("Invalid Chars in Input. Use only A-Z.");


		//                 ABCDEFGHIJKLMNOPQRSTUVWXYZ
		char const *Table="JPGVOUMFYQBENHZRDKASXLICTW";
		// Decrypt:       "SKXQLHCNWARVGMEBJPTYFDZUIO"

		-> Missing Code for rotating table by Offset;

		Result[i] = Table[(Input[i]-'A')];

		-> Missing Code for string termination;

		Offset+=Rotation;
	}
	return Result;
}

int main(int argc, char *argv[]) {

	-> Insert missing Code:
	->    Declare Variables
	->    Read and check parameters;
	->    Garbage in parameters: Report to stderr, then exit 1;

	// Call Crypt; Crypt may throw an error

	try {

		-> Missing code for runtime measurement

		Result= Crypt(Input,Encrypt,InitialRotation,Rotation);

		-> Insert missing Code:
		->    output of result and possibly runtime report
		->    get rid of acquired memory

	} // try

	catch ( const char *Reason ) {

		// We assume Crypt gives exceptions only of type const char *
		// otherwise more complex exception handlers are needed.

		cerr << Reason << endl; // Handle Exception
		exit(1);

	} // catch

	return 0;

} // main
```

Kurzaufgabe K1.2 (0,5 Punkte)

Ergänzen Sie `k1`, so dass das Chiffrat durch Aktivieren mit `-d` wieder entschlüsselt werden kann.