

2. Übungsblatt zur Vorlesung „Einführung in die Programmiersprache C++“

Wintersemester 2022 / 2023

Aufgabe 2.1: Berechnung des Datums von Ostern (20 Punkte)

Schreiben Sie ein Programm `easter`, das für gegebene Jahreszahlen berechnet, auf welchen Tag Ostern fällt. Der Algorithmus dazu findet sich in Donald Knuth's Buch "The Art of Computer Programming".

Neue Konzepte dieser Aufgabe:

- Operatoren
- Funktionen, Funktionsprototypen
- Umleitung von I/O-Operationen (Unix)

Beschreibung des Algorithmus:

GIVEN: Y: the year for which the date of easter is to be determined
FIND: The date (month and day) of Easter
STEP E1: Set G to $(Y \bmod 19) + 1$
[G is the „golden year“ in the 19-year Metonic cycle.]
STEP E2: Set C to $(Y / 100) + 1$
[C is the century]
STEP E3: Set X to $(3C / 4) - 12$
[X is the skipped leap years]
Set Z to $((8C + 5) / 25) - 5$
[Z ist a correction factor fort he moon's orbit.]
STEP E4: Set D to $(5Y / 4) - X - 10$
[March $((-D) \bmod 7 + 7)$ is a sunday]
STEP E5: Set E to $(11G + 20 + Z - X) \bmod 30$
If E is 25 and G is greater than 11 or if E is 24 increment E
[E is the „epact“ which specifies when a full moon occurs.]
STEP E6: Set N to $44 - E$
If N is less than 21 add 30 to N
[March N is a „calendar full moon“.]
STEP E7: Set N to $N + 7 - ((D + N) \bmod 7)$
[N is a Sunday after full moon.]
STEP E8: If $N > 31$ the date is APRIL $(N - 31)$ otherwise the date is MARCH N

Bemerkung: Alle Divisionsoperationen in diesem Algorithmus sind Integer-Divisionen, was bedeutet, dass die Reste nicht beachtet werden.

Bemerkung: Der Kommentar „March $((-D) \bmod 7 + 7)$ is a sunday“ ist eigentlich nur für die Jahre nach 1752 wahr, weil in diesem Jahr eine 11-tägige Datumskorrektur stattgefunden hat. Dies muss im Programm nicht beachtet werden.

Bemerkung: Obwohl Donald Knuth den Algorithmus geschrieben hat, bedeutet dies nicht, dass er ihn in einer schönen Art und Weise aufgeschrieben hat. Konkret: Die Benutzung von einzelnen Zeichen als Variablennamen ist für gewöhnlich eine schlechte Idee (weil diese Zeichen keine Bedeutung für Personen haben, die den Programmcode verstehen sollen). Ihr Programm `easter` soll vernünftige Bezeichner enthalten.

Dazu Folgende aus dem Internet stammende Erklärung zum Algorithmus:

But the trouble with Easter is that it has to fall on Sunday. I mean, if you don't have that, all the other non-fixed holidays get all screwed up. Who ever heard of having Ash Wednesday on a Saturday, or Good Friday on Thursday? If the Christian church had gone and made a foolish mistake like that, they'd have been the laughingstock of all the other major religions everywhere.

So the Church leaders hemmed and hawwed and finally defined Easter to fall on the first Sunday after the first full moon after the vernal equinox. I guess that edict must not have been too well-received, or something, because they then went on to define the vernal equinox as March 21st, which simplified matters quite a bit, since the astronomers of the time weren't really sure that they were up to the task of finding the date of the real vernal equinox for any given year other than the current one, and often not even that. So far so good.

The tricky part all comes from this business about the full moon. The astronomers of the time weren't too great at predicting that either, though usually they could get it right to within a reasonable amount, if you didn't want a prediction that was too far into the future. Since the Church really kinda needed to be able to predict the date of Easter more than a few days in advance, it went with the best full-moon-prediction algorithm available, and defined "first full moon after the vernal equinox" in terms of that. This is called the Paschal Full Moon, and it's where all the wacky epacts and Metonic cycles come from. So what's a Metonic Cycle?

A Metonic cycle is 19 years. The reason for the number 19 is the following, little-known fact: If you look up in the sky on January 1 and see a full moon, then look again on the same day precisely 19 years later, you'll see another full moon. In the meantime, there will have been 234 other full moons, but none of them will have occurred on January 1st.

What the ancient astronomers didn't realize, and what makes the formula slightly inaccurate, is that the moon only really goes around the earth about 234.997 times in 19 years, instead of exactly 235 times. Still, it's pretty close - and without computers, or even slide rules, or even pencils, you were happy enough to use that nice, convenient 19-year figure, and not worry too much about some 0.003-cycle inaccuracy that you didn't really have the time or instruments to measure correctly anyway. Okay, how about this Golden Number business then?

It's just a name people used for how many years into the Metonic cycle you were. Say you're walking down the street in Medieval Europe, and someone asks you what the Golden Number was. Just think back to when the last 19-year Metonic cycle started, and start counting from there. If this is the first year of the cycle, it means that the Golden Number is 1; if it's the 5th, the Golden number is 5; and soon. Okay, so what's this Epact thing?

In the Gregorian calendar, the Epact is just the age of the moon at the beginning of the year. No, the age of the moon is not five billion years - not here, anyway. Back in those days, when you talked about the age of the moon, you meant the number of days since the moon was "new". So if there was a new moon on January 1st of this year, the Epact is zero (because the moon is new, i.e. zero days „old“); if the moon was new three days before, the Epact is three; and so on.

When Easter was first introduced, the calculation for the Epact was very simple -- since the phases of the moon repeated themselves every 19 years, or close enough, the Epact was really easy to calculate from the Golden Number. Of course, this was the same calendar system that had one leap year every 4 years, which turned out to be too many, so the farmers ended up planting the fields at the wrong times, and life just started to suck.

Pope Gregory makes things more complicated. You probably already know about the changes Pope Gregory XIII made in 1582 with respect to leap years. No more of this „one leap year every four years“ business like that Julius guy said. Nowadays, you get one leap year every four years unless the current year is a multiple of 100, in which case you don't -- unless the current year is also a multiple of 400, in which case you do anyway. That's why 2000 was a leap year, even though 1900 wasn't (I'm sure many of you were bothered by this at the turn of the millenium).

Well, it turns out that the other thing Pope Gregory did, while he was at it, was to fix this Metonic Cycle-based Easter formula which, quite frankly, had a few bugs in it -- like the fact that Easter kept moving around, bumping into other holidays, occuring at the wrong time of year, and generally making a nuisance of itself.

Unfortunately, Pope Gregory had not been throwing out the old, poorly-designed code and building a new design from scratch, he sort of patched up the old version of the program (this is common even in modern times). While he was at it, he changed the definition of Epact slightly. Don't worry about it, though -- the definition above is the new, correct, Gregorian version.

This is why you'll see Knuth calculating the Epact in terms of the Golden Number, and then applying a „correction“ of sorts afterwards: Gregory defined the Epact, and therefore Easter, in terms of the old definition with the Metonic cycles in it. Knuth is just the messenger here. So what is this thing with „Z“ and the moon's orbit?

It's just the „correction“ factor which the Pope introduced (and Knuth later simplified) to account for the fact that the moon doesn't really orbit the earth exactly 235 times in 19 years. It's analogous to the „correction factor“ he introduced in the leap years -- the new formula is based on the old one, is reasonably simple for people who don't like fractions, is also kind of arbitrary in some sense, and comes out much closer to reality, but still isn't perfect. What about all the rest of that stuff?

Beschreibung des Programms easter:

Schreiben Sie ein Programm, dass eine Serie von Jahren aus einem Textfile einliest und das Datum von Ostern ausgibt. Benutzen Sie die Unix Ein- und Ausgabeumleitung, um die Eingabe aus einer Datei zu lesen und die Ausgabe in eine Datei zu schreiben, d.h., das Programm soll wie folgt aufgerufen werden:

```
% easter < infile > outfile
```

Die Datei infile enthält eine Liste von Jahren, z.B.:

```
1994
1995
1998
```

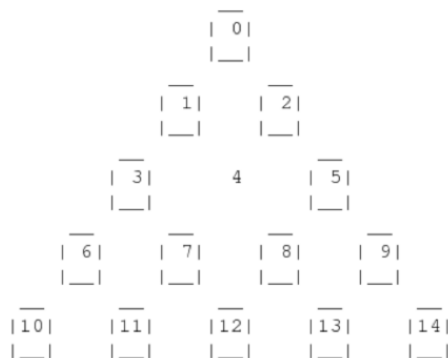
und das outfile soll eine Liste von Jahreszahlen und Datumsangaben enthalten, z.B.:

```
1994 - April 3
1995 - April 16
1998 - April 12
```

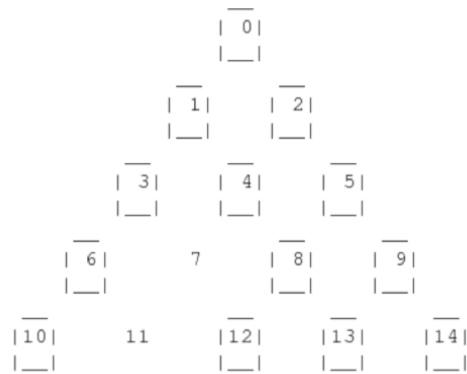
Die Ein- und Ausgabe aus Dateien ist auch möglich. Hier sollen Sie aber explizit die `stdin` und `stdout`, d.h. `scanf` und `printf`, benutzen! Für das Lösen der Aufgaben wird Ihnen ein Makefile zur Verfügung gestellt. Benutzen Sie dieses Makefile zusammen mit dem Programm `make` um Ihr Programm zu übersetzen und die Berechnungen zu testen. Die dazu benötigten Dateien werden Ihnen auf der Vorlesungsseite in StudIP zur Verfügung gestellt.

Aufgabe 2.2: Pegboard-Puzzle (80 Punkte)

In dieser Aufgabe werden Sie ein Programm zur Lösung eines Pegboard-Puzzles schreiben. Dabei werden Sie etwas über Rekursion und effiziente Problemdarstellungen lernen. Das Pegboard-Game wird auf einem dreieckigen Brett mit 15 Einträgen gespielt. Die folgende Abbildung zeigt wie die Einträge angeordnet und nummeriert sind:



Der Spielablauf ist wie folgt: Zunächst sind 14 Einträge besetzt, einer ist leer. In der Abbildung sind besetzte Einträge mit einem Kästchen um die Nummer versehen. Einträge können sich durch Springen ähnlich dem Spiel Dame verändern. Es sind allerdings nur horizontale und diagonale Sprünge erlaubt, keine vertikalen. So ergibt der Sprung von Kästchen 11 über Kästchen 7 folgende Konfiguration:



Nun könnte es mit Eintrag 9 weitergehen, indem über 8 gesprungen wird. Leere Felder dürfen nicht übersprungen werden. Das Ziel des Spiels besteht darin, alle Einträge bis auf einen abzuräumen. Im Folgenden soll der Terminus "Board" eine bestimmte Spielkonfiguration bezeichnen. Ein "Move" stellt einen Spielzug dar und ist erlaubt, wenn er den Regeln entspricht.

Beschreibung des Programms:

Schreiben Sie ein Programm `triangle_game`, das ein initiales Board vom Benutzer einliest. Ihr Programm soll dann eine Sequenz von Zügen generieren, die zur Lösung des Problems führen falls eine existiert. Zur Erleichterung der Programmierung sollen die Züge in umgekehrter Reihenfolge ausgegeben werden. Für diese Aufgabe stellen wir Ihnen ein Makefile sowie Funktionen zur Ein- und Ausgabe der Boards zur Verfügung. Die Erzeugung der Lösung soll durch einen rekursiven Backtracking Algorithmus erfolgen.

Problemrepräsentation:

Die Eingabefunktion `triangle_input()` stellt Ihnen ein 15-Stelliges Integer-Array zur Verfügung, das die Spielfelder entsprechend der oben gezeigten Nummerierung darstellt. Der Wert 1 markiert eine besetzte Zelle, unbesetzte Zellen bekommen den Wert 0. Überführen sie die Dreiecksdarstellung mit 15 Einträgen analog zum folgenden Beispiel in ein Array mit 25-Einträgen:

Die Ursprüngliche Darstellung

```

      1
     1 1
    1 0 1
   1 1 1 1
  1 1 1 1 1

```

soll nun folgendermaßen dargestellt werden:

```

1 2 2 2 2
1 1 2 2 2
1 0 1 2 2
1 1 1 1 2
1 1 1 1 1

```

Überlegen Sie sich, welche Vorteile diese Darstellung zum Auffinden von erlaubten Zügen hat. Die Spielzüge werden sollen jeweils in einem dreistelligen Array `move[]` repräsentiert werden, wobei `move[0]` die Position des Startkästchens, `move[1]` die Position des übersprungenen Kästchens und `move[2]` die Position des Zielkästchens darstellen. Zur Lösung des Problems sollten sie folgende Hilfsfunktionen implementieren:

```

/* Return the number of entries on the board. */
int nentries(int board[]);

```

```

/* Return 1 if the move is valid on this board, otherwise return 0. */
int valid_move(int board[], int move[]);

```

```

/* Make a move on this board. */
void make_move(int board[], int move[]);

/* Unmake this move on this board. */
void unmake_move(int board[], int move[]);

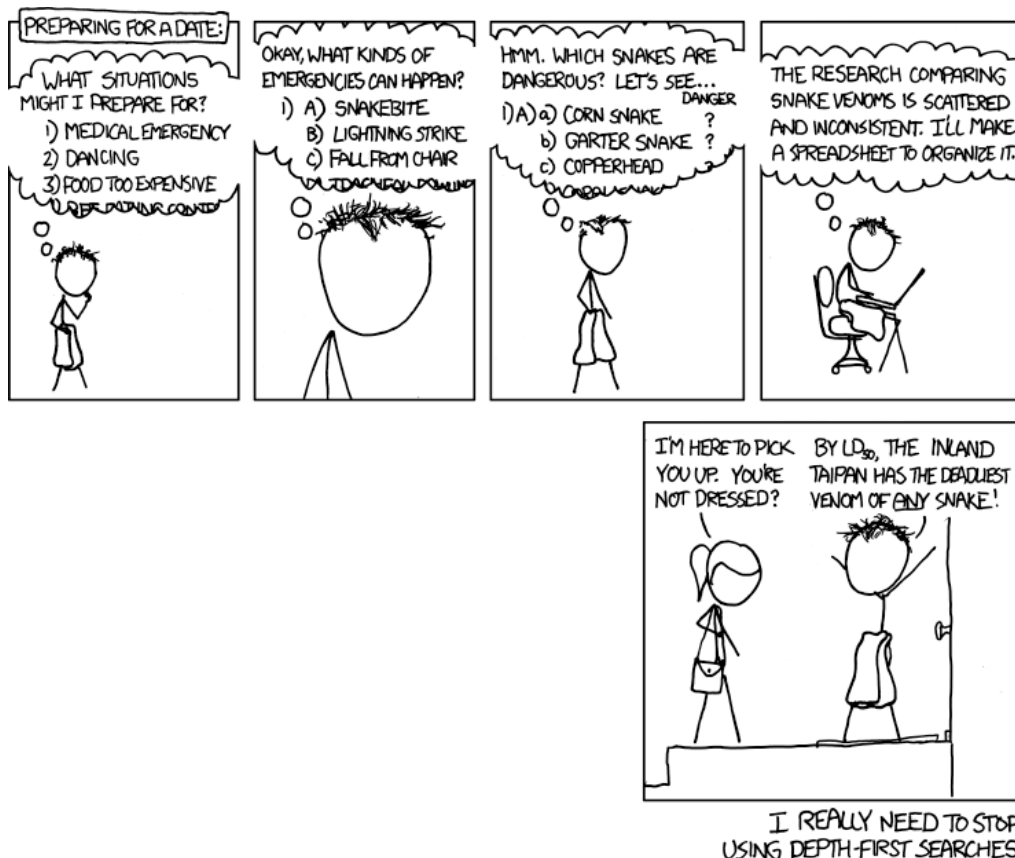
/*
 * Solve the game starting from this board. Return 1 if the game can
 * be solved; otherwise return 0. Do not permanently alter the board
 * passed in. Once a solution is found, print the boards making up
 * the solution in reverse order.
 */
int solve(int board[]);

```

Ein automatisierter Test kann mit `make test` ausgeführt werden.

Abgabe

Checken Sie Ihre Abgaben bis zum 07.11.2022 8:00 Uhr in das GitLab-Repository Ihrer Gruppe ein. Stellen Sie sicher, dass Ihre Programme ohne Fehler und Warnungen übersetzen und den Style-Richtlinien dieser Veranstaltung (siehe StudIP) entsprechen.



“breadth-first search makes a lot of sense for dating in general, actually; it suggests dating a bunch of people casually before getting serious, rather than having a series five-year relationships one after the other.”