Technische Universität München
Institut für Informatik I-16
Dr. Tobias Lasser
Jakob Vogel

WiSe 2012/13
Übungsblatt 2
22. Oktober 2012

Algorithmen und Datenstrukturen

Aufgabe 1 Zahldarstellung

In den bisherigen Aufgaben haben Sie bereits Variablen des Types `int` verwendet, wobei wir darunter zunächst nur abstrakt einen Speicherbereich verstanden haben, in dem eine Vorzeichenbehaftete Ganzzahl abgelegt werden kann. In der Vorlesung wurde nun besprochen, dass derartige Zahlen (in einzelnen Bits) als Binär-Koeffizienten abgespeichert werden und die Anzahl dieser Bits das darstellbare Intervall definieren.

a) Welche primitiven Datentypen für Ganzzahlen können Sie auf einem 64-bit-System mindestens erwarten?

b) Es gibt eine Reihe von sogenannten *bitweisen Operationen*, die direkt mit den Binärkoeffizienten arbeiten. Diese Operationen sind normalerweise sehr schnell im Vergleich zu arithmetischen Operationen, da sie extrem gut in Hardware implementiert werden können.

Die wichtigsten Operationen sind die folgenden:

- *Bitweises NOT.* Alle Koeffizienten (bei gegebener maximaler Bitanzahl) einer Zahl werden betrachtet, und eine neue Zahl wird generiert, bei der alle Koeffizienten invers gesetzt sind:

$$\neg\ 0001\ 0111_2 \ \rightsquigarrow\ 1110\ 1000_2$$

In C/C++ erzeugt man ein entsprechendes Ergebnis durch `ergebnis = ~wert;`. Dieser Befehl ist **nicht** mit dem *logischen NOT* `ergebnis = !wert;` zu **verwechseln**!

- *Bitweises AND, OR, XOR.* Der Computer betrachtet jeweils die Koeffizienten zweier Zahlen und erzeugt bitweise eine dritte.

Bei *AND* werden nur diejenigen Koeffizienten auf 1 gesetzt sind, bei denen in **beiden** Ausgangszahlen eine 1 gesetzt war:

$$0001\ 0111_2 \ \wedge\ 0000\ 1011_2 \ \rightsquigarrow\ 0000\ 0011_2$$

In C/C++ schreiben wir `ergebnis = wert1 & wert2;`.

Bei *OR* genügt dagegen schon eine 1 bei **mindestens einer** der Zahlen für einen Koeffizienten 1:

$$0001\ 0111_2 \ \vee\ 0000\ 1011_2 \ \rightsquigarrow\ 0001\ 1111_2$$

In C/C++ schreiben wir `ergebnis = wert1 | wert2;`.

Bei *XOR* (Exklusives ODER) werden zuletzt diejenigen Koeffizienten auf 1 gesetzt, bei denen **entweder** die eine **oder** die andere Zahl eine 1 gesetzt hat:

$$0001\ 0111_2 \ \oplus\ 0000\ 1011_2 \ \rightsquigarrow\ 0001\ 1100_2$$

Die Operation in C/C++ ist `ergebnis = wert1 ^ wert2;`.

- *Bitshift nach rechts.* Hierbei werden alle Koeffizienten einer Zahl um eine oder mehrere Stellen nach „rechts“, also in Richtung der niedrigeren Potenz verschoben, wobei „links“ mit 0 aufgefüllt wird und „rechts“ die Werte entfallen. Beispielsweise könnte eine Verschiebung um eine Stelle nach rechts aussehen wie folgt:

  $$0001\,0111_2 \;\rightsquigarrow\; 0000\,1011_2$$

  In C/C++ wird diese Operation mit `ergebnis = wert >> stellen;` durchgeführt.

- *Bitshift nach rechts.* Analog zu oben geschilderter Operation werden hierbei alle Koeffizienten in Richtung der höheren Potenz verschoben, wobei nun „links“ die Werte entfallen und „rechts“ mit 0 aufgefüllt wird. Das entsprechende Beispiel ist dann:

  $$0001\,0111_2 \;\rightsquigarrow\; 0010\,1110_2$$

  In C/C++ lautet der Befehl `ergebnis = wert << stellen;`.

Wählen Sie nun eine beliebige Zahl $\geq 8_{10}$ und berechnen Sie die Binärkoeffizienten. Wenden Sie nun dreimal hintereinander einen Bitshift nach rechts um jeweils eine Stelle an, und berechnen Sie die Dezimaldarstellungen der Resultate. Was ist Ihre Beobachtung?

Schieben Sie diese Zahl nun wieder dreimal hintereinander um jeweils eine Stelle nach links und berechnen Sie die Dezimaldarstellungen. Was folgern Sie?

c) Wählen Sie nun eine neue Zahl $\geq 32_{10}$ und berechnen Sie das Resultat der „Ver-UND-ung“ mit 31_{10}. Was ist der Effekt? Aufbauend darauf und auf die Aufgabe im letzten Übungsblatt, wie können Sie schneller als mit dem Modulo-Operator berechnen ob eine Zahl gerade oder ungerade ist?

d) Bilder werden auf Computern häufig angenähert durch einen Raster von Farbwerten, die linear hintereinander im Speicher abgelegt werden. Um den Farbwert für das Pixel (x/y) aus einem Bild der Größe $\mathit{breite} \times \mathit{höhe}$ im Speicher zu finden, muss der lineare Index $x + y \cdot \mathit{breite}$ berechnet werden.

Texturen in Computerspielen werden genauso abgespeichert. Die Dimensionen dieser Texturbilder sind dann sehr oft Zweierpotenzen, also etwa 256×128 oder 512×512. Können Sie sich vorstellen, warum?

e) Zur Darstellung negativer Zahlen wird das Zweierkomplement verwendet, bei dem die Verrechnung einfach und die Darstellung der 0 eindeutig ist. Insbesondere besteht der Unterschied zwischen einem Vorzeichen-behafteten und einem Vorzeichen-losen Typen nicht in einem unterschiedlichen Datenformat, sondern lediglich in einer anderen Interpretation der 0-1-Folge.

Wie können Sie bei einem `signed` 8-bit-Typen ohne Vergleich mit 0 schnell feststellen, ob der Wert negativ ist? Wie kann man seinen Absolutbetrag nur mit bitweisen Operationen berechnen?

Was halten Sie von dieser Schleife?

```
for (unsigned int i = 50; i >= 0; --i)
    std::cout << "i = " << i << std::endl;
```

f) Neben Ganzzahlen existieren weiterhin spezielle Formate für Fließkommazahlen, wobei hier die Zahl in Exponentialschreibweise durch drei Ganzzahlen (Vorzeichen, Mantisse und Exponent) beschrieben und gespeichert wird.

Wie schätzen Sie die Anforderungen von Gleitkomma-Arithmetik im Vergleich zu Ganzzahl-Arithmetik ein? Sind Ihre zuvor erarbeiteten Erkenntnisse über bitweise Operationen auch für Fließkomma-Darstellungen gültig?

Aufgabe 2 (P) Arrays

Die bisher verwendeten Variablen kann man als einzelne „Schubladen“ verstehen, in denen genau ein Wert eines speziellen Typs hinterlegt werden kann. Damit sind bereits sehr mächtige Programmabläufe realisierbar, doch müssen bisweilen große Datenmengen verarbeitet werden, wofür statt einzelner „Schubladen“ sozusagen ganze „Regale“ oder gar „Lagerhäuser“ notwendig werden.

Im Rahmen der Veranstaltung werden wir uns mit verschiedenen Datenstrukturen zur effizienten Speicherung und Verarbeitung beschäftigen. Eine sehr grundliegende Struktur, die bereits in den meisten Programmiersprachen vorgesehen ist, ist das Array (die „Reihung“). Dabei wird statt einer einzigen gleich eine ganze Menge „Schubladen“ eingerichtet, die alle Werte eines einzelnen Typs enthalten können und über einen ganzzahligen Index identifiziert werden. Letzteres ist natürlich auch algorithmisch möglich, so dass automatisiert zugegriffen werden kann.

Um also ein Array von Ganzzahlen der Länge `n` namens `ein_array` in C++ zu erzeugen (in C lauten die Befehle anders), schreiben Sie[1]:

```
int* ein_array = new int[n];
```

Der Operator `new` reserviert einen gewissen Speicherbereich und gibt ihn zur entsprechenden Verwendung frei. Bedenken Sie, dass Sie **allen** reservierten Speicher nach Ende der Nutzung wieder freigeben müssen, ansonsten haben Sie ein *Speicherleck* erzeugt. Moderne Betriebssysteme (nicht aber einzelne Programme) sind in dieser Hinsicht zwar sehr robust, doch ist davon vor allem bei Eingebetteten Systemen nicht auszugehen! Zur Freigabe lautet der Befehl:

```
delete[] ein_array;
```

Nach dieser Freigabe darf auf das Array **nicht mehr** zugegriffen werden!

Zwischen diesen Punkten kann das Array verwendet werden. Sie greifen auf das i-te Element zu mittels `ein_array[i]`, wobei $0 \leq i < n$ gelten muss. C/C++ überprüft diese Beschränkung des Indices **nicht**[2], allerdings kann es infolge einer falschen Angabe zu (schwer identifizierbaren) Fehlern und Abstürzen kommen.

Sie können sowohl aus Array-Elementen lesen, als auch in diese schreiben, vergleichbar zu „normalen“ Variablen, also etwa:

```
int wert = ein_array[0];
ein_array[1] = wert + ein_array[2];
ein_array[ein_array[0]] = 5;
```

Bedenken Sie dabei aber, dass Arrays normalerweise zu Beginn **nicht** initialisiert sind, in den Elementen also „Datenmüll“ aus der vorherigen Nutzung des Speichers liegen kann.

a) Schreiben Sie ein Programm, das ein Array von 10 Elementen erzeugt, und die Zellen dann mit den Werten 1 bis 10 initialisiert. Anschliessend laufen Sie das Array von hinten durch und tauschen dabei die Werte zwischen angrenzenden Zellen aus, so dass Sie letztendlich ein Array erhalten, bei dem der erste Wert 10 ist und die Zahlen 1 bis 9 folgen.

   Erweitern Sie Ihr Programm anschließend durch eine weitere Schleife so, dass das komplette Array invers sortiert ist.

[1] Wenn Sie bereits C/C++ beherrschen, werden Sie wissen, dass Arrays auch mit statischer Größe ohne `new` erzeugt werden können. Wir vernachlässigen diese Möglichkeit, da sie für unsere Anwendung zu unflexibel ist.

[2] Dies ist ein klassisches Einfalltor für Computerviren!

b) Eine weitergehende Datenstruktur, die Sie bereits kennengelernt haben, ist der Stack („Keller“ oder „Stapel“). Von abstraktem Standpunkt aus ist dieser eine Black Box, die eine Schnittstelle aus zwei Funktionen hat:

- Über die Funktion `void push(int wert)` kann ein Wert in den Stack abgelegt werden, vorausgesetzt, dass genügend Platz vorhanden ist.
- Über die Funktion `int pop()` kann ein Wert aus dem Stack ausgelesen werden, der dadurch dort gleichzeitig auch gelöscht wird.

Zwischen den beiden Funktionen herrscht der „Constraint“ („Vertrag“), dass die Werte, die zuletzt mittels *push* hinterlegt wurden, als erste wieder durch *pop* entfernt werden. Dies gleicht eben genau einem Stapel, auf dem nur oben abgelegt, und auch nur das oberste Element wieder entfernt werden kann. Man spricht daher auch von sogenanntem *LIFO (Last In, First Out)*.

Implementieren Sie also einen `int`-Stack mit definierbarer Kapazität, fügen Sie die beiden Schnittstellenfunktionen hinzu, kümmern Sie sich um die Fehlerfälle, und testen Sie Ihre Implementierung. Nutzen Sie hierzu *globale Variablen*[3]!

Aufgabe 3 (P) Objektorientierung I

Neben primitiven Datentypen und Arrays besteht in vielen Programmiersprachen auch die Möglichkeit, eigene zusammengesetzte Datentypen zu erzeugen. Will man etwa für ein Grafikprogramm Kreise speichern, so könnte man Mittelpunkte und Radien zwar einzeln ablegen, doch wäre es eleganter, sich einen neuen Datentyp „Kreis“ zu erzeugen. In C/C++ könnte das dann etwa so aussehen:

```
struct Kreis
{
    float x, y;
    float radius;
};
```

In C++ – Achtung, hier existieren Unterschiede zu C! – verwendet man den neuen Typen dann etwa wie folgt:

```
Kreis k;
k.x = 5.0f;
k.y = 3.5f;
k.radius = std::sqrt(k.x*k.x + k.y*k.y);
std::cout << "Der Radius ist " << k.radius << std::endl;
```

Je nach Sprache können dann sogar Funktionen angehängt werden. Dies funktioniert in C++ – aber nicht in C! – so:

```
struct Kreis
{
    float x, y;
    float radius;

    Kreis()
    {
```

[3] Diese sind bewusst bisher nicht besprochen worden, bitte versorgen Sie sich in der Bibliothek und im Internet mit den nötigen Informationen!

```
        // Dies ist der Konstruktor, der bei der Erzeugung eines Objekts
        // automatisch ausgefuehrt wird. Hier koennen Initialisierungen und
        // Speicher-Belegungen erfolgen. Der Name muss immer "<Klassenname>()"
        // lauten und es existiert kein Rueckgabetyp (auch nicht void).
    }

    virtual ~Kreis()
    {
        // Dieser Destruktor -- Namensschema "~<Klassenname>()" -- wird beim
        // Loeschen der Variable automatisch ausgefuehrt. Hier kann etwa
        // belegter Speicher wieder freigegeben werden. Wieder existiert
        // kein Rueckgabetyp, "virtual" ist eine Compiler-Instruktion!
    }

    // Dieses ist nun eine wirkliche Funktion, die aus dem Zustand (also ohne
    // Parameter) einen Wert berechnet und zurueckgibt.
    virtual float berechneLaengeDesOrtsvektors()
    {
        return std::sqrt(x*x + y*y);
    }

    // Aehnliches geht natuerlich auch mit Parametern!
    virtual void verschiebeKreis(float dx, float dy)
    {
        x += dx;
        y += dy;
    }
};
```

Instanzfunktionen können Sie so aufrufen:

```
std::cout << "Die Laenge des Ortsvektors ist "
          << k.berechneLaengeDesOrtsvektors() << std::endl;
k.verschiebeKreis(-2, 4);
std::cout << "Der Kreis liegt jetzt bei " << k.x << "/" << k.y << std::endl;
```

Man nennt einen solchen Datentypen eine „Klasse“, und Variablen dieses Typen dann „Instanzen“. Dieses ganze Programmierparadigma wird gemeinhin als „Objektorientierung“ bezeichnet[4] und wird heute – zumindest im Bereich von Consumer-Software – weitgehend als Standard betrachtet.

So werden wir die meisten der Daten-Strukturen aus der Vorlesung mit Klassen umsetzen. Schreiben Sie den Stack aus der vorhergehenden Aufgabe also so um, dass statt globaler Variablen ein Stack-Objekt erzeugt wird, das dann auch die beiden Funktionen `push` und `pop` umfasst.

[4] Zur Objektorientierung kommen jedoch noch einige weitere Konzepte wie Sichtbarkeiten, Klassenmethoden und Vererbung. Manches davon werden wir noch im Laufe der Veranstaltung genauer untersuchen.