Crashkurs

Aufgaben: http://icpc.ira.uka.de/public/wettbewerbe/getconnected12
Einsendung: http://domjudge.ira.uka.de/team/

1 Kompilieren auf der Konsole

Zuerst legt man in einem Editor seiner Wahl die zu kompilierende Quelldatei an. Üblicherweise hängt man für C-Dateien `.c`, für C++-Dateien `.cpp` (andere übliche Endungen: `.cc`, `.C`, `.cxx`, `.cc`) und für Java-Dateien `.java` als Dateierweiterung an, damit der Editor das richtige Syntaxhighlighting wählt. Man beachte auch, dass in Linux Groß- und Kleinschreibung unterschieden wird. Außerdem bereiten Leerzeichen einige Problem auf der Konsole. Es empfiehlt sich also, alle Dateinamen kleinzuschreiben und keine Leerzeichen zu verwenden.

Bei Java sollte weiterhin der Dateiname der öffentlichen Klassen in der Datei entsprechen. Eine Datei `Foo.java` sollte also nur eine öffentliche Klasse `Foo` beinhalten.

Zum eigentlichen Kompilieren öffne man dann eine Konsole. Man wird mit einem Prompt der Form

```
name@rechner(~)$
```

begrüßt. Vor dem Dollar sieht man den aktuellen Pfad in runden Klammer. Dabei steht ~ für das Home-Verzeichnis. Mit `cd` wechselt man das aktuelle Verzeichnis und mit `ls` listet man den Inhalt des aktuellen Verzeichnisses auf. Zum Beispiel wechselt man mit

```
cd foo
```

in das Unterverzeichnis `foo` des aktuellen Verzeichnisses. Zum Kompilieren gehe man nun in das Verzeichnis, in dem die Quelldateien liegen.

Zum Übersetzen von C++-Programmen gibt man dann folgendes ein:

```
g++ -Wall foo.cpp -o foo
```

C-Programme werden sehr ähnlich übersetzt:

```
gcc -Wall foo.c -o foo
```

In beiden Fällen wird eine ausführbare Datei namens `foo` erzeugt. Der Parameter `-Wall` sorgt dafür, dass Warnungen über fragwürdigen Code ausgeben werden.

Zum Ausführen verwendet man folgenden Befehl:

```
./foo
```

Man kann die Ein- und Ausgabe auch umlenken:

```
./foo < in > out
```

Hier liest das Programm nun die Eingabe aus der Datei `in` ein und gibt die Ausgabe in die Datei `out` aus. Man kann auch nur die Eingabe oder nur die Ausgabe umlenken. Falls nicht umgelenkt wird, ist die Eingabe oder die Ausgabe an die Konsole gebunden.

Java Programme werden ähnlich kompiliert:

```
javac Foo.java
```

Zum Ausführen benutzt im Gegensatz aber folgendes:

```
java Foo
```

Auch bei Java kann man die Ein- und Ausgabe umlenken:

```
java Foo < in
java Foo > out
java Foo < in > out
```

2 Weitere Hinweise

2.1 Arbeit mit der Konsole

Um Dateinamen automatisch zu vervollständigen, drücke man Tabulator. Man sollte schon reflexhaft Tabulator drücken, um sich lästige Tipparbeit zu sparen. Mit den Pfeiltasten kann man Kommandos wiederholen.

2.2 Vergleich mit der erwarteten Ausgabe

Um sicherzustellen, dass ein Programm die erwartete Ausgabe hat, sollte man `diff` verwenden, da ein manueller Vergleich korrekte Whitespaces und kleine Zahlendreher nicht mit Sicherheit erkennen kann.

Um zum Beispiel das Programm `foo` mit der Eingabe `sample.in` aufzurufen und die Ausgabe mit der erwarteten Ausgabe in `sample.out` zu vergleichen, benutzt man folgenden Befehl:

```
./foo < sample.in | diff - sample.out
```

Das Programm `diff` vergleicht die beiden angegeben Dateien, wobei der Bindestrich bedeutet, dass die eine Datei aus der Standardeingabe genommen wird. Es gibt nichts (!) aus, falls die Dateien übereinstimmen. Andernfalls werden die Unterschiede angezeigt.

In obiger Kommandozeile bedeutet das Symbol `|`, dass die Ausgabe des linkes Programmes (`./foo`) genommen und in die Eingabe des rechten (`diff`) gesteckt wird.

2.3 Mehrere Kommandos ausführen

Mit && kann man 2 Befehle nacheinander ausführen, wobei der zweite Befehl nur dann ausgeführt wird, falls der erste erfolgreich war. Mit der oben erläuterten Verwendung von `diff` ergibt sich damit die folgende Kommandozeile zum Testen eines Programmes:

```
g++ -Wall -o foo foo.cpp && ./foo < sample.in | diff - sample.out
```

2.4 Sinnvolle Fehlermeldungen

Die Fehlermeldungen von `g++` sind (insbesondere bei Verwendung von Templates) zum Teil sehr obskur. Für lesbarere Fehlermeldungen kann man den `clang++` Compiler (mit den gleichen Optionen wie `g++`) benutzen. (Momentan funktioniert das in der ATIS nicht.)

2.5 Compilerflags

Im Judge werden folgende Compiler- und Ausführflags genommen:

```
gcc -Wall -O2 -static -pipe -lm
g++ -Wall -O2 -static -pipe
javac
java -Xrs -Xss8m -Xmx300000k
```

Diese Flags sollten das Programmverhalten aber nicht wirklich verändern, d. h es ist auch nicht wichtig, zu verstehen, was sie bewirken.

3 Beispielprogramme

Hier folgen nun noch einige Beispielprogramme zur Lösung des `SimpleSum`-Problems, die insbesondere die Ein- und Ausgabe in der jeweiligen Sprache illustrieren.

3.1 C

In C verwendet man `scanf` und `printf` aus `stdio.h` zur Ein- und Ausgabe:

```c
#include <stdio.h>

int main() {
        int sum = 0, n, tmp, i;
        scanf("%d", &n);
        for (i = 0; i < n; ++i) {
                scanf("%d", &tmp);
                sum += tmp;
        }
        printf("%d\n", sum);

        return 0;
}
```

3.2 C++

In C++ kann man die C-Funktionen verwenden. Alternativ nimmt man `cout` und `cin` aus `iostream`:

```
#include <iostream>
using namespace std;

int main() {
        int n, sum = 0;
        cin >> n;
        for (int i = 0; i < n; ++i) {
                int tmp;
                cin >> tmp;
                sum += tmp;
        }
        cout << sum << "\n";

        return 0;
}
```

3.3 Java

Die einfachste Möglichkeit zum Parsen von Eingaben in Java ist die `Scanner`-Klasse:

```
import java.util.Scanner;

class Main {
        public static void main(String[] args) {
                Scanner s = new Scanner(System.in);
                int n = s.nextInt();
                int sum = 0;
                for (int i = 0; i < n; ++i) {
                        sum += s.nextInt();
                }

                System.out.println(sum);
        }
}
```