

Konzepte der Programmiersprachen

Sommersemester 2007

Eduard Wiebe

Übungsblatt 2

Aufgabe 2.1: Dangling References

1. Was versteht man unter einer *dangling reference*? Geben Sie anhand kommentierter Programmbeispiele zwei unterschiedliche Situationen an, in denen dieser Effekt auftritt.
2. Gegeben sei das folgende Programmfragment:

```
type Ptr is pointer to Some_Type;

procedure Clean_Assign (A: Ptr; B: Ptr) is
begin
  Free(A);
  A := B;
end Clean_Assign;
```

`Free` veranlasst die Freigabe des Speichers für das vom Zeiger designierte Objekt und setzt den Zeigerparameter auf `null`, führt aber keine weiteren Prüfungen oder Maßnahmen durch.

Zeigen Sie durch ein kleines Programmfragment, das die Prozedur `Clean_Assign` aufruft, wie durch den Aufruf von `Clean_Assign` unabhängig vom Parameterübergabemechanismus „dangling references“ im aufrufenden Programm entstehen können.

3. Man kann „dangling references“ in einer Sprache vollständig vermeiden. Wie? Welche Nachteile entstehen dadurch?

Aufgabe 2.2: Aliasing

Im Kapitel über Bindungskonzepte für Programmiersprachen wurde der Begriff *Aliasing* eingeführt. Zwei Bezeichner sind Aliase, wenn sie auf dasselbe Objekt (oder allgemeiner auf im Speicher überlappende Variablen/Objekte) verweisen.

1. Geben Sie zwei Möglichkeiten an, wie durch Referenzparameter Alias-Effekte entstehen können.
2. Durch welche anderen Programmiersprachenkonzepte können Alias-Effekte entstehen?

3. Welche zusätzlichen Probleme entstehen durch das Auftreten von Alias-Effekten?
4. Werden Alias-Effekte grundsätzlich erst durch Eigenschaften der Programmiersprache hervorgerufen, oder gibt es auch Situationen, in denen Aliasing „natürlich“ ist?

Aufgabe 2.3: Zeiger in C und Ada

1. Vergleichen Sie die Semantik von Zeiger-Datentypen in C und Ada 95 hinsichtlich der Möglichkeiten zur Verhinderung illegaler Speicherzugriffe.
2. Gibt es Situationen, in denen die Sprachregeln von C oder Ada 95 auch „harmlose“ Speicherzugriffe verhindern? Wenn ja, welche?

Aufgabe 2.4: Zeigeroperationen in C

Gegeben sei folgendes obskure C-Programm:

```
#include <stdio.h>

char vek[6] = "AEIOU";
char *ptr;

void print(char c)
{
    putchar(c);
    putchar('\n');
}

int main(void)
{
    ptr = vek;
    print(*ptr);
    print(*ptr + 1);
    print(*ptr);
    print(*(ptr + 1));
    print(*ptr);
    print(*ptr++);
    print(*ptr);
    print(*(ptr++));
    print(*ptr);
    print(++ptr);
    print(*ptr);
    print(++ptr);
    print(*ptr);
    return 0;
}
```

Welche Buchstaben geben die Aufrufe der Funktion `print` aus, wenn die Zielmaschine für die Repräsentation von Characters ASCII verwendet?

Beachten Sie dabei insbesondere die Vorrang- und Auswertungsreihenfolgeregeln für Operatoren!

Aufgabe 2.5: Fließkommadatentypen

Gegeben sei ein Fließkommadatentyp `real` mit vier signifikanten Dezimalstellen.

1. Zeigen Sie, dass für diesen Datentyp weder das Assoziativgesetz der Addition gilt noch das Distributivgesetz. Haben diese Tatsachen irgendwelche praktischen Auswirkungen?
2. Gilt für diesen Datentyp das Kommutativgesetz der Addition? Gilt das Kommutativgesetz der Multiplikation?
3. *Zum Knobeln:* Gilt für diesen Datentyp das Assoziativgesetz der Multiplikation? Wenn nicht, wie groß kann der relative Fehler werden?

Hilfe: Versuchen Sie zunächst, die zweite Frage direkt für `a*b` zu beantworten.

Aufgabe 2.6: Exakte Repräsentation

Welche der folgenden Datentypen kann man prinzipiell ohne Fehler und Bereichsüberschreitung repräsentieren?

1. Reelle Zahlen
2. Rationale Zahlen
3. Ganze Zahlen
4. Strings

Aufgabe 2.7: Spass an der Freude

Was liefert dieses APL-Programm :-)

$$(2 = +/[1]0 = (\iota 10) \circ .|(\iota 10))/\iota 10$$