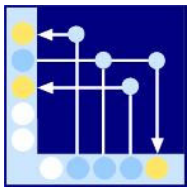
Fakultät Elektronik und Informatik
Studiengang Informatik



Programmieren in C++

Vorlesung im Sommersemester 2018
Prof. Dr. habil. Christian Heinlein

1. Übungsblatt (15. März 2018)

Aufgabe 1: Ganze Zahlen

Teilaufgabe 1.a)

Finden Sie heraus, wie groß die Typen `short`, `int`, `long` und `long long` auf Ihrem System sind und ermitteln Sie ihre minimalen und maximalen Werte!
Was lässt sich damit über die interne Darstellung negativer Werte aussagen?

Finden Sie außerdem heraus, welche Typen sich hinter den Aliasen `int_least16_t` und `int_fast16_t` verbergen!
Tipp: Übergeben Sie Werte dieser Typen jeweils an eine Funktion `test`, die für alle in Frage kommenden Typen überladen ist!

Teilaufgabe 1.b)

Überprüfen Sie die in §2.1.4 der Vorlesungsfolien genannten Regeln, nach denen der Typ eines ganzzahligen Literals ermittelt wird, indem sie gezielt dezimale, oktale, hexadezimale und duale Literale mit bestimmten Werten erstellen und ihren Typ ermitteln!

Teilaufgabe 1.c)

Überprüfen Sie die in §2.1.8 genannten Regeln, nach denen bei einer arithmetischen Operation der gemeinsame Typ der Operanden – der gleichzeitig auch der Resultattyp der Operation ist – ermittelt wird!

Wie lautet demnach das Ergebnis von `1U - 2U` und das von `(unsigned short)1 - (unsigned short)2`?

Teilaufgabe 1.d)

Für welche `int`-Werte `x` und `y` liegt `x/y` nicht im Wertebereich von `int`, sofern `int`-Werte im Zweierkomplement dargestellt werden?

✎

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

// Makro, um Schreibarbeit zu sparen und Codeverdopplung zu vermeiden.
// (Alternativ könnte man test als Funktionsschablone definieren.)
// #T wird vom Präprozessor durch den Parameter T als Zeichenkette
// ersetzt.
#define TEST(T) \
    void test (T x) { \
        cout << #T << " " << x << " (" << sizeof(T) << " bytes) [" \
            << numeric_limits<T>::min() << ", " \
            << numeric_limits<T>::max() << "]" << endl; \
    }

// Definition der Funktion test mit verschiedenen Parametertypen.
TEST(short)
TEST(int)
TEST(unsigned int)
TEST(long)
TEST(unsigned long)
TEST(long long)
TEST(unsigned long long)
TEST(double)

int main () {
    // Aufrufe von test für Werte der Typen short, int, long
    // und long long.
    test(short(1));
    test(1);
    test(1L);
    test(1LL);

    // Aufrufe von test für Werte der Typen int_least16_t
    // und int_fast16_t.
    test(int_least16_t(1));
    test(int_fast16_t(1));

    // Aufrufe von test für unterschiedliche ganzzahlige Literale.
    // Annahme: int hat 32 Bit, long und long long haben 64 Bit.
    // Dann kann int Werte kleiner als 2 hoch 31 darstellen,
    // unsigned int Werte kleiner als 2 hoch 32.
    // 2 hoch 31 ist etwas größer als 2'000'000'000,
    // 2 hoch 32 ist etwas größer als 4'000'000'000.

    // Für dezimale Literale ohne Suffix u oder U
    // werden keine vorzeichenlosen Typen verwendet.
    test(2'000'000'000);        // int
    test(4'000'000'000);        // long
    test(8'000'000'000);        // long
```

```
    // Für oktale, hexadezimale und duale Literale ggf. schon.
    test(   0B100'0000'0000'0000'0000'0000'0000'0000);  // int
    test(  0B1000'0000'0000'0000'0000'0000'0000'0000);  // unsigned int
    test(0B1'0000'0000'0000'0000'0000'0000'0000'0000);  // long

    // Aufrufe von test für das Ergebnis arithmetischer Operationen
    // mit unterschiedlichen Operandentypen.
    test(short(1) + 1U);        // short + unsigned int -> unsigned int
    test(1 + 1U);               // int + unsigned int -> unsigned int
    test('x' + 1.0);            // char + double -> double

    cout << 1U - 2U << endl;    // -1 mod (2 hoch 32) = 4294967295
    cout << (unsigned short)1 - (unsigned short)2 << endl; // -1
}
```

Wenn der Wertebereich eines ganzzahligen Typs asymmetrisch ist (z. B. `-32768` bis `32767` bei `short`), dann werden negative Werte offenbar im Zweierkomplement dargestellt.

Auf einem 64-Bit-PC ist `int_least16_t` normalerweise gleich `short` (16 Bit), aber `int_fast16_t` möglicherweise gleich `long` (64 Bit), weil sich 64-Bit-Werte dort schneller verarbeiten lassen als 16-Bit-Werte.

Für `y = -1` ist `x/y` normalerweise gleich `-x`.
Wenn `int`-Werte im Zweierkomplement dargestellt werden, liegt dieser Wert für `x = numeric_limits<int>::min()` jedoch nicht im Wertebereich von `int`.