SAARLAND UNIVERSITY
COMPUTER SCIENCE

Arbeitsgruppe Programmierung
Jun.-Prof. Dr. Sebastian Hack
Dipl.-Inform. Christoph Mallon

C/C++-Kurs — Übungsblatt 1

Aufgabe 1: Idiome

Finden Sie heraus, wozu folgende Programmschnipsel dienen.

1. ```
   int f(int x) { return !!x; }
   ```

2. ```
   int g(int a, int b) { return (a > b) - (a < b); }
   ```

3. ```
   char* h(char* d, char* s)
   {
     char* o = d;
     while (*d++ = *s++);
     return o;
   }
   ```

Aufgabe 2: Bitfi...

Implementieren Sie einige Funktionen zur effizienten Kodierung von x/y-Koordinaten in einer Ganzzahl.

1. Die beiden Koordinaten liegen im Wertebereich 0 (einschließlich) und 160 (ausschließlich).
2. Die beiden Koordinaten sollen gemeinsam in einer Ganzzahl kodiert werden.
3. Um spätere Erweiterungen zu erleichtern, sollen der Datentyp, in dem die Koordinaten kodiert werden, und notwendige Details der Implementierung leicht abänderbar sein.
4. Verwenden sie Zusicherungen (assert), um Bedienungsfehler festzustellen.

Es soll folgende Funktionalität zur Verfügung gestellt werden:

1. `EncodeXY()`: Nimmt zwei Koordinaten entgegen und liefert als Ergebnis die gepackte Form.
2. `GetX()`, `GetY()`: Nehmen gepackte Koordinaten entgegen und liefern die x/y-Koordinate.
3. `SetX()`, `SetY()`: Nehmen gepackte Koordinaten und eine neue x/y-Koordinate entgegen und erzeugen neue gepackte Koordinaten bei denen x/y durch den neuen Wert ausgetauscht wurde.
   ```

Aufgabe 3: sizeof

Erklären sie die jeweiligen Ausgaben.

Hinweis: `%zu` ist die Formatangabe, um eine Zahl vom Ergebnistyp von `sizeof` auszugeben. Unter Windows wird dies allerdings nicht unterstützt. Verwenden sie stattdessen `%Iu`.

```
#include <stdio.h>

void f(short a[10], short* b)
{
  short s;
  printf("%zu␣%zu␣%zu\n", sizeof(s), sizeof(+s), sizeof(-s));

  printf("%zu␣%zu\n", sizeof(a), sizeof(b));

  short c[10];
  short* d = c;
  printf("%zu␣%zu␣%zu\n", sizeof(c), sizeof(d), sizeof(c[1000]);

  short (*e)[10] = &c;
  printf("%zu␣%zu␣%zu\n", sizeof(e), sizeof(*e), sizeof(**e));
}

int main(void)
{
  short arr[20];
  f(arr, arr);
  return 0;
}
```

Aufgabe 4: void∗

Sortieren Sie die Reihung mit Elementtyp `int`, welche mit den Werten 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5 initialisiert ist, absteigend mittels der Funktion `qsort()`, welche in `<stdlib.h>` zu finden ist. Dokumentation zu `qsort()` finden Sie u.a. in der C99-Norm im Abschnitt §7.20.5.2. Mit `printf("%d", x);` können Sie jeweils einen `int` ausgeben.

Aufgabe 5: Präprozessor

1. Ein Programmierer möchte einen Datentyp `string_t` als Zeiger auf konstanten `char` definieren. Er betrachtet folgende beiden Möglichkeiten:

   ```
   #define string_t char const*
   typedef char const* string_t;
   ```

   Geben Sie eine Benutzung des Typen `string_t` an, die bei den beiden Definitionen unterschiedlichen Effekt hat.

2. Nehmen Sie an, ein Übersetzer verstünde erst seit Version 4 `void*`. Vor Version 4 benutzte man in den Fällen, in denen man heute `void*` benutzen würde, `char*`. Wie kann man diesem Unterschied mithilfe des Präprozessors begegnen? Schreiben Sie eine Präprozessordefinition und eine beispielhafte Funktionsdeklaration. Die Versionsnummer ist in einem vordefinierten Makro `MAJOR_VERSION` definiert.

3. Vervollständigen Sie folgende Definition:

```
#ifdef DEBUG
# define INC(x) /* hier einsetzen */
#else
# define INC(x) (++(x))
#endif
```

   Dabei soll gelten: Ist `DEBUG` definiert und ist zusätzlich die Variable `debug_level` nicht 0, so soll außer `x` auch `debug_count` inkrementiert werden; `x` wird in jedem Fall inkrementiert. Achten Sie darauf, daß durch die Definition von `DEBUG` kein zuvor funktionierender Code ungültig wird (z. B. `y = INC(x);`).

Aufgabe 6: struct

Betrachten Sie folgende Strukturdefinitionen:

```
struct {
  int a[2];
  int b;
} struct1 = { { -1 }, -1 };

struct {
  int a, b, c;
} struct2 = { { 1 }, { 2 },{ 3 } },
  struct3 = {(1, 2, 3)},
  struct4 = {{ 1, 2, 3 }};

struct {
  int a,b;
} structs1[2] = { 5, 7, 9, 10 },
  structs2[2] = { { .a = 9, .b = 10 }, { .b = 9 } };

struct {
  int a[3];
  struct {
    int a[2];
  } b;
} structs3[2] = { 1, 2, 3, { 4 }, { 5, 6 }, { 7, 8 } };
```

Geben Sie für jede Variable explizit deren vollständige Werte an. Wenn eine Variablendefinition fehlerhaft ist, implementationsabhängiges oder undefiniertes Verhalten hevorruft, so schreiben Sie stattdessen dies, mit Begründung.

Aufgabe 7: switch

1. Was ist und wie funktioniert Duff's Device?

2. Erklären Sie die Verschränkung von `switch` und `if` in folgendem Fragment zum Zerteilen (parsing) von Anweisungen aus einem Java-Übersetzer. Bei `T_...` handelt es sich im Zahlenkonstanten, welche Token repräsentieren: `T_int` → Schlüsselwort `int`, `T_LBRACE` → `{`, `T_IDENT` → Ein Bezeichner. `lookahead[0]` ist das aktuell betrachtete und `lookahead[1]` das darauffolgende Token in der Eingabe.

```
static Statement* parse_block_statement()
{
  switch (lookahead[0])
  {
    case T_LBRACE: return parse_compound_statement(); break;
    case T_if:     return parse_if();                 break;
    case T_return: return parse_return();             break;
    case T_while:  return parse_while();              break;

    case T_IDENT:
      if (lookahead[1] == T_IDENT)
      {
    case T_boolean:
    case T_int:
    case T_void:
        return parse_local_var_declaration();
      }
      else
      {
    default:
        return parse_expression_statement();
      }
  }
}
```