4.2 Programmiersprache C

4.2.1. Elementare Datentypen

- (signed) **int**, **unsigned int**, (signed) **short** (int), **unsigned short** (int), (signed) **long** (int), **unsigned long** (int)
  Elementare Datentypen für ganze Zahlen mit oder ohne Vorzeichen. Neben dem Vorzeichen unterscheiden sie sich im Zahlenbereich:
  sizeof(short) <= sizeof(int) <= sizeof(long)
- **float**, **double**
  Elementare Datentypen für Fließkommazahlen. float hat die geringere Genauigkeit und geringeren Speicherbedarf.
- **char**
  Elementarer Datentyp für ASCII-Zeichen (andere Zeichentabellen mit 256 Einträgen möglich). Er enthält stets ein Zeichen.

Beispiele für Variablendeklarationen:

```
int a;
unsigned short s;
float f;
char c;
```

4.2 Programmiersprache C

- Aufzählungen: **enum** typename { value1, value2, ...};
  Aufzählungen werden als Typ definiert und dann genutzt. Dies kann auch in einer Anweisung geschehen.
  Beispiele: (1) enum Tehestand {ledig, verheiratet, geschieden, verwitwet};
  enum Tehestand stand;
  stand = ledig;
  (2) enum Tehestand {ledig, verheiratet, geschieden, verwitwet} stand;
  stand = ledig;

- **void**
  Nichttyp – wird genutzt, um anzugeben, dass eine Funktion keinen Wert zurückliefert, also eine Prozedur ist.

4.2.2. Operatoren auf elementaren Datentypen

- Operationen für ganze Zahlen (int, short, long,...)
  Postfix-Inkrement name**++**, Postfix-Dekrement name**--**,
  Präfix-Inkrement **++**name, Präfix-Dekrement **--**name,
  Vorzeichen **-**name, Speichergröße **sizeof**(name), Adresse **&**,
  Umwandlung **(**typename**)** name,
  Multiplikation *, ganzzahlige Division **/**, Rest bei Division **%**,
  Addition **+**, Substraktion **-**,
  bitweise Verschiebung **<< >>** (nicht mit short als erstem Operanden),
  Vergleiche **< > <= >= == !=** ,
  Bitoperationen **& ^ |**,
  logische Negation **!**, logisches Und **&&**, logisches Oder **||**,
  Bedingungsoperator **?:**,
  Zuweisungen **= += -= *= /= %= <<= >>= &= ^= |=**,
  Sequentielle Auswertung **,**

Bemerkung: Logische Werte werden in C als ganze Zahlen betrachtet, wobei 0 für falsch(false) und alle anderen Werte für richtig(true) stehen.

4.2 Programmiersprache C

- Operationen für Fließkommazahlen (float, double)
  Vorzeichen **-**name, Speichergröße **sizeof**(name), Adresse **&**,
  Umwandlung **(**typename**)** name,
  Multiplikation *, Division **/**,
  Addition **+**, Substraktion **-**,
  Vergleiche **< > <= >= == !=** ,
  Zuweisungen **= += -= *= /=**,
  Sequentielle Auswertung **,**
- Operationen auf Zeichen (char)
  Speichergröße **sizeof**(name), Adresse **&**,
  Umwandlung **(**typename**)** name,
  Vergleiche **< > <= >= == !=** ,
  Zuweisung **=**,
  Sequentielle Auswertung **,**

4.2 Programmiersprache C

- Operationen auf Aufzählöungstypen (enum)
  Speichergröße **sizeof**(name), Adresse **&**,
  Umwandlung **(**typename**)** name,
  Vergleiche **< > <= >= == !=** ,
  Zuweisung **=**,
  Sequentielle Auswertung **,**

4.2.3. Bausteine für strukturierte Datentypen

- **Felder, Arrays**: type name**[**length**]**
  Ein Array mit fester Länge wird durch Angabe des Typs und des Namens vereinbart, wobei hinter dem Namen in eckigen Klammern die Länge vorgegeben wird. Der Zugriff erfolgt dann über eckige Klammern.
  Beispiel: int a[5];
  a[3] = 4;

- **Multidimensionale Felder**
  Diese Felder werden einfach durch mehrfache eckige Klammern deklariert.
  Beispiel: double matrix[3][3];

- **Datensätze**: **struct** typename { type1 component1; type2 component2; ...};
  Datensätze oder Rekords werden mit Hilfe von struct definiert. Als Zugriffsoperator dient der Punkt.
  Beispiel:
  struct Tstudent {char vorname[20]; char nachname[20]; unsigned int alter;};
  struct Tstudent student;
  student.vorname = 'Martin';

4.2 Programmiersprache C

- **Zeiger**: type *name;
  Zeiger sind ein elementarer Datentyp, der auf einen Speicherbereich zeigt, der gerade so groß wie der angegebenen Typ ist. Der Stern * in der Deklaration zeigt an, dass ein Pointer und keine Variable deklariert werden soll.
  Operationen: *, &, ++, --, (Typname), +, -, <, >, >=, >=, ==, !=, =, +=, -=, ,

  Der * dient als rechtsseitiger Operator auch zum Referenzieren. Zeigt der Zeiger auf ein struct und soll auf eine Komponente zugegriffen werden, kann statt **(*zeiger).komponente** auch **zeiger->komponente** geschrieben werden. Der rechtsseitige Adressoperator **&** liefert die Adresse eines Datenobjekts und somit einen Wert für die Zuweisung zum Zeiger. Die Zusatzfunktionen **malloc()** und **free()** dienen dem dynamischen Erstellen und Löschen von Datenobjekten. Zeiger werden zum Aufbau von dynmia Listen, Stapeln, Schlangen, Bäumen, dynamischen Arrays usw. benutzt.

  Beispiel:
```
int *p; int i;
i = 3;
p = &i;
*p = 5;  // setzt i auf 5!
```

4.2.4. Prioritätsebenen für Operatoren

17. Konstantennamen, Indizes [ ], Funktionsaufrufe ( ), Auswahl . ->
16. Postfix-Inkrement, -Dekrement ++ --
15. Präfix-Inkrement ++ --, Vorzeichen -, Speicherbedarf sizeof, Negation !, Indirektion & *
14. Umwandlung (Typname)
13. Multiplikative Operatoren * / %
12. Additive Operatoren + -
11. Verschiebung << >>
10. Relationen < > <= >=
9. Gleichheit == !=

8. bit-weises und &
7. bit-weises exklusives oder ^
6. bit-weises oder |
5. Logisches und &&
4. Logisches oder ||
3. Bedingungsoperator Bedingung ? Wert1 : Wert2
2. Zuweisung = += -= *= /= %= <<= >>= &= ^= |=
1. Sequentielle Auswertung ,

4.2.5. Steueranweisungen

Bemerkungen:

- Jede Anweisung endet mit einem Semikolon.
- Die Zuweisung ist eine Anweisung und ein Operator, der den zugewiesenen Wert zurückgibt.
- Es gibt auch eine goto-Anweisung, die man aber vermeiden sollte.

Komposition

- **Verbundanweisung**: { Anweisung1 Anweisung2 ... }
  Die geschweiften Klammern fassen mehrere Anweisungen zu einer einzigen Anweisung zusammen.

Alternation

- If-Anweisungen:
  if (Bedingung) Anweisung
  if (Bedingung) Anweisung1 **else** Anweisung2
- Switch-Anweisung:
  switch (ganzzahliger Ausdruck**) {**
  case Konstante1 **:** Anweisung1
  case Konstante2 **:** Anweisung2
  ...
  default: AnweisungN
  }

  Vorsicht: Wenn der ganzzahlige Ausdruck Konstante1 ergibt und in Anweisung1 kein **break;** steht, wird Anweisung2 auch noch ausgeführt!

Iteration

- while-Anweisung:
  while (Bedingung**)** Anweisung
  Führe Anweisung aus, wenn und solange Bedingung erfüllt ist!
- do-while-Anweisung:
  do Anweisung **while (**Bedingung**);**
  Führe Anweisung aus und wiederhole dann, solange Bedingung erfüllt ist!
- for-Anweisung:
  for (Ausdruck1**;** Ausdruck2**;** Ausdruck3**)** Anweisung
  Berechne Ausdruck1 (Initialisierung)!
  Wenn Ausdruck2 wahr ist (nicht 0 ergibt), führe Anweisung aus und berechne anschließend Ausdruck3!
  Prüfe nun wieder Ausdruck2, falls wahr: Anweisung, berechne Ausdruck3,...!
- break-Anweisung, continue-Anweisung
  break; bricht die Ausführung der Iteration ab und setzt Ausführung hinter der Iteration fort. **continue;** führt zu sofortigem Beenden der Anweisung und zum Sprung in die Bedingungsprüfung oder zum Berechnen von Ausdruck3.

4.2.6. Unterprogramme

Alle Unterprogramme sind Funktionen. Eine Prozedur ist eine Funktion die den Nichttyp void zurückgibt.

- Definition einer Funktion
  resulttype functionname(type1 parameter1, type2 parameter2,...) { Anweisung};

  Funktionen müssen vor Ihrem Benutzen deklarariert worden sein. Dies geschieht durch ihre Definition oder durch eine Deklaration:

  resulttype functionname(type1 parameter1, type2 parameter2,...);

  Jede Funktion darf jede andere Funktion aufrufen. Die Übergabe von Werten erfolgt durch die Parameter, deren Typ streng geprüft wird, wobei implizite Typumwandlungen häufig sind.

Es gibt nur einen Rückgabewert, dessen Typ der Signatur (resulttype) zu entnehmen ist. Die Zuweisung erfolgt in der Anweisung mittels der Returnanweisung

return Ausdruck;
return (Ausdruck);

wobei dies auch den Rücksprung zur aufrufenden Funktion zur Folge hat. Es dürfen mehrere return-Anweisungen in einer Funktion stehen. In Prozeduren bleibt der Ausdruck leer

return;

4.2.7. Programmstruktur

- Jedes Programm muss eine Funktion main enthalten.
  Die Programmausführung beginnt mit dieser Funktion.
- Man kann ein Programm aus mehreren Dateien aufbauen und damit gliedern.

4.2.8. Beispiel (Primzahltest):

```
#include <stdio.h>      // Hinzufügen der IO-Standardbibliothek

int isprime(int number)
{
      int ist_primzahl, divisor;
      ist_primzahl = 1;
      for (divisor = number-1; divisor>1; divisor--)
            if (number % divisor == 0) {ist_primzahl = 0; break;};
      return ist_primzahl;
}


void main()
{
      printf('Ist 13 eine Primzahl (1 - Ja, 0 – Nein) ? %d', isprime(13));
      printf('Ist 15 eine Primzahl (1 - Ja, 0 – Nein) ? %d', isprime(15));
}
```