1 Advanced

1.1 Structs

Structs ermöglichen es, verschiedene Datentypen in einem Datenobjekt zusammenzufassen. Dies ist bei Hierarchiestrukturen nützlich und trägt auch generell zu einer guten Code Organisation bei. Das zugreifen auf Elemente ist mittels des *Punktoperators* möglich. Structs werden außerhalb von Funktionen definiert.

```
struct Person {
    char name[64];
    int alter;
};

int main() {
    struct Person Max;
    strcpy(max.name, "Max");
    max.alter = 17;
}
```

1.2 Unions

Unions ermöglichen es, verschiedene Datentype im gleichen Speicherbereich abzulegen. Allerdings kann nur ein Wert auf einmal belegt werden. Dies ermöglicht Speichereffizientes programmieren.

```
union overlay {
    long longWert;
    float floatWert;
}
```

Hier wird ein *long* und ein *float* in den gleichen Speicher gelegt. Zugriff erflogt wie bei Structs.

1.3 Bitfields

Mit Bitfields ist es möglich, die genaue Anzahl an Bits, die ein Wert belegen soll, zu bestimmen. Hat man beispielsweise mehrere *boolean* Werte, wäre es Verschwendung dafür 1, 2, oder sogar 4 Bytes pro Wert zu belegen. Im folgenden Beispiel wird gezeigt wie man dieses Problem löst.

```
struct {
    unsigned char bool0: 1;
    unsigned char bool1: 1;
    unsigned char bool2: 1;
    unsigned char bool3: 1;
    unsigned char bool4: 1;
    unsigned char bool5: 1;
    unsigned char bool6: 1;
    unsigned char bool7: 1;
};
```

Diese Struktur beinhaltet nun 8 Werte, die alle ein bit groß sind und daher genauso viel Speicher benötigen wie ein regulärer *char*.

1.4 typedef

Das *typedef* keyword wird verwendet, um Typen neue Namen zu geben. So kann man beispielsweise das Erstellen einer struct vereinfachen. (vergleiche oben)

```
typedef struct person_t {
    (...)
} Person;

int main() {
    Person p;
}
```

1.5 SDCC - Small Device C Compiler

Siehe: http://sdcc.sourceforge.net/doc/sdccman.pdf