Sprachstandards von C

Agenda 1

1 Einführung
2 Der Weg zum Sprachstandard C99
3 Der Sprachstandard C99
    1 Variablen und Konstanten
    2 Typumwandlung
    3 Ein- und Ausgabe
    4 Operatoren
    5 Kontrollstrukturen
    6 Funktionen

Agenda 2

1 Einführung

- Imperative Programmiersprache
- 1971-1973 von David Ritchie & Bell Labs
- Direkter Zugriff auf die Hardware
- Hohe Verbreitung
- Standardbibliothek mit rudimentären Funktionen
  - Compilerhersteller können weitere Bibliotheken einbinden

2 Der Weg zum Sprachstandard C99

- K&R-C
  - C-Variante der Entwickler
- C89/C90
  - Einheitliche Eigenschaften festgelegt
  - Umfang der Standardbibliothek festgelegt
  - Neue Sprachelemente hinzugefügt
- C95
  - Fehlerkorrekturen und Änderungen am Sprachumfang

3 Der Sprachstandard C99

- C99 inkl. 3 Korrekturen → C0X
- Basis für neue C-Standards (vorher C90)
- Präprozessor-Test
  - C89/C90: ---
  - C95: 199409L
  - C99: 199901L

```
#if defined(__STDC_Version__) && __STDC_Version__ >= XX
Quelltext
#endif
```

3 Der Sprachstandard C99

- C++ Erweiterungen flossen ein
  - „//“ Zeilenkommentar
  - Schlüsselwort inline
  - Variablendeklaration innerhalb for-Anweisung
- Neuerungen von C99
  - Unterstützung komplexer Zahlen
  - Erweiterung ganzzahliger Datentypen
  - Lokale Felder variabler Größe

3 Der Sprachstandard C99

3.1 Variablen und Konstanten

- Ganzzahlen
  - char -128 – +127
  - Integer:
    - int „natürliche Größe“
    - short int -32767 – +32767
    - long int 2^32 Zeichen
    - long long int 2^64 Zeichen
    - int_least8_t vorg. Mindestbreite
    - int32_t vorg. exakte Breite
  - signed und unsigned

3.1 Variablen und Konstanten

- wide character
  - Zeichen > 1 Byte
  - wchar_t
  - Muss bei der Initialisierung angegeben werden
  - Bsp.:

```
char c1 = 'D';
wchar_t c2 = L'D';
```

  - Zusätzliche Bibliotheksfunktionen, die auf wchar_t anwendbar sind (printf → wprintf)

3.1 Variablen und Konstanten

- Fließkommazahlen
  - float — einfache Genauigkeit = 32bit
  - double — doppelte Genauigkeit = 64bit
  - long double — 80bit
  - Immer „signed“

3.1 Variablen und Konstanten

- Konstanten...
  - ...mit #define

```
#define zahl1 20
```

    - Nachteil:
      - Typ der Variablen nicht erkennbar
  - ...mit const

```
const int zahl2 = 21
```

    - Vorteil:
      - Typ und Gültigkeitsbereich werden überprüft

3.1 Variablen und Konstanten

- Variablenverhalten
  - static
    - Variable wurde auf fester Speicheradresse gespeichert
  - volatile
    - Variable ändert sich außerhalb des Programmkontextes
  - register
    - Variable wird in das Register gelegt

3.2 Typumwandlung

- Implizite Typumwandlung
  - Umwandlung wird vom Compiler ausgeführt
  - Erweiternde Typumwandlung

```
int i = 42;
float f = i;
```

- Explizite Typumwandlung
  - Typumwandlung wird im Code angegeben
  - Einschränkende Typumwandlung

```
float f = 42.123;
int i = (int) f;
```

3.3 Ein- und Ausgabe

- Printf
  - Keine Standard-Funktion
  - Zeichenkette auf der Standardausgabe
  - Formatelemente: %d, %o, %x, %a, %c, ...
  - Flags können zusätzlich genutzt werden (+, -, 0, #)
  - Feldbreite kann angegeben werden
  - Nachkommastellen können festgelegt werden

```
printf("Zahl 20: %o", 20);
printf("\nZahl 20: %+i", 20);
printf("\nZahl 20: %05d", 20);
printf("\nZahl 20.543586: %10.3f", 20.543586);
```

3.3 Ein- und Ausgabe

- Scanf
  - Liest Wert ein und speichert ihn in der angegebenen Variablen ab
  - Erwartet Adresse der Variablen
  - Adressoperator &
  - Formatelemente: %o, %u, %d

```
scanf("%d", &x);
```

3.4 Operatoren

- Prä- und Postfix Operatoren
  - Inkrement- (++) und Dekrement- (--) Operatoren
  - Präfix ist zu bevorzugen

```
int x = 20;
int zahl1 = x++;
int zahl2 = ++x;
```

- Shift-Operator
  - << und >>
  - Bitweise Verschiebung des Inhalts der Variablen
  - << entspricht *2
  - >> entspricht /2

3.4 Operatoren

- Logik-Operatoren
  - <, >, >=, <=, !=, ==
  - &&, ||, ! → and, or, not
  - Boolsche Werte
    - false == 0
    - true != 0
  - Rückgabetyp der Vergleichsoperatoren: int
  - Datentyp _Bool über <stdbool.h>

3.5 Kontrollstrukturen

- if-Anweisung

```
if (exp.)
{
    int i = 5;
    tu dies;
}
else if (exp.)
{
    wenn nicht, dann das;
}
else
{
    und wenn das auch nicht, dann das;
}
```

- switch-Anweisung

```
switch(exp.)
{
    case(exp.):
        statement;
    break;

    case(exp.2):
        statement;
    break;
}
```

3.5 Kontrollstrukturen

- for-Schleife

```
for(i = 0; i < 10; ++i)
{
    statement;
}
```

- while-Schleife

```
int i = 10;
while (i > 1)
{
    statement;
    --i;
}
```

- do-while-Schleife

```
int i = 10;
do
{
    statement;
    --i;
}
while(i > 1);
```

3.6 Funktionen

- Definition
  - Funktionsdefinition

```
Rueckgabetyp Name(Parameter)
{
    statement;
    return-statement;
}
```

- Prototypen
  - Funktionsdeklaration

```
Rueckgabetyp Name(Parameter);
```

3.6 Funktionen

- Inline-Funktionen
  - Schnellstmöglicher Funktionsaufruf
  - Compiler kopiert Funktionsinhalt an die Stelle des Funktionsaufrufs
    - Sprung zur Funktion und Rücksprung entfällt
  - Compiler entscheidet selbst, ob eine Funktion als „inline“ ausgeführt werden soll

```
inline Rueckgabetyp Name(Parameter)
{
    statement;
    return-statement;
}
```

3.7 Header

- Form: *headername*.h
- Einbindung über #include
- Nötig um das Programm zu modularisieren
- Funktionen oder Konstanten werden an einer anderen Stelle im Programm benötigt
- Bsp.: <ctype.h>
  - int isdigit(int c)
  - int islower(int c)
  - ...

3.8 Arrays

- Definition

```
int myArray[x];
int myArray2[x][y];
```

- Initialisierung

```
int myArray[3] = {1, 2, 3};

int myArray2[2][3] = {
                        {1,2,3}
                        {4,5,6}
                    };

int myArray3[2][3] = {1,2,3,4,5,6};
```

3.8 Arrays

- Arrays variabler Größe
  - Deklaration wie üblich
  - Arraygröße ist variabel

```
int i = 5;
int myArray[i];
```

3.9 Strings

- Kein Datentyp „String“
- Strings sind Arrays vom Datentyp char
- „Stringende-Zeichen“ \0 wird benötigt

```
char string1[] = { 't', 'e', 's', 't', '\0'};
char string2[] = {"test"};
```

- Bibliotheksfunktionen für Strings
  - int strcmp (char* string1, char* string2)
  - char* strcat (char* string1, const char* string2)
  - char* strcpy (char* dest, const char* source)

4 Ausblick

- C1X
  - Multithreading
  - char16_t und char32_t zur Unicode-Unterstützung
  - Prüfung von Array-Grenzen zur Laufzeit
  - Unterstützung der dezimalen Darstellung von Fließkommazahlen

5 Literatur

- http://openbook.galileocomputing.de/c_von_a_bis_z/
- http://de.wikibooks.org/wiki/C

Vielen Dank für Eure Aufmerksamkeit

Bei Fragen:

9weiser@inf...