Funktionspointer

C gestattet, mit Hilfe von typedef Funktionstypen und Funktionspointertypen zu vereinbaren. Mit diesen Datentypen können dann Variablen, Vektoren, Struktur-/Unionkomponenten oder Funktionsparameter definiert werden. Natürlich können die entsprechenden Definitionen auch direkt ohne mit typedef vereinbarten Datentyp definiert werden.

Funktionstyp

```
typedef void f (void);
```

vereinbart einen Datentyp f, der für Funktionen ohne Parameter und Returnwert steht.

typedef int fcmp (void* p1, void *p2);

vereinbart einen Datentyp f, der für Funktionen mit zwei Pointerparameter steht.

Parameternamen können, müssen aber nicht angegeben werden.

Funktionspointertyp

```
typedef void (*pf) (void);
```

vereinbart einen Datentyp pf, der für einen Pointer auf eine Funktion steht. Funktionstypen werden grundsätzlich in Verbindung mit Funktionspointern verwendet.

Funktionspointer

Vereinbarung von Funktionspointervariablen:

```
f* pF1;   // f ist ein Funktionstyp
pf pF2;// pf ist ein Funktionspointertyp
```

Beide Definitionen vereinbaren eine Pointervariable zur Aufnahme der Adresse einer Funktion ohne Parameter und ohne Returnwert.

Funktionspointerverwendung

Gibt es nun eine Funktion

```
void fxyz(void)
{...}
```

so kann deren Adresse einer Funktionspointervariablen zugewiesen werden.

```
pF1=fxyz;
```

Der Adressoperator würde in diesem Falle, genauso wie bei Arrays ignoriert. Die Funktion kann nun über die Funktionspointervariable aufgerufen werden.

```
pF1();
```

Aufruf einer Funktion über Funktiospointer

- Der Funktionaufruf erfolgt in gleicher Weise, wie bei einer gewöhnlichen Funktion.
- Statt des Funktionsnamens wird der Name der Funktionspointervariablen angegeben.
- Die Parameter müssen entsprechend typverträglich sein.

```c
#include <stdlib.h>
#include <stdio.h>

typedef void(*f)(int i);
f pf;

void fprintdec(int i)
{
  printf("%d\n",i);
}

void fprinthex(int i)
{
  printf("%08x\n",i);
}

int main(int argc, char*argv[])
{
  int i=atoi(argv[1]);
  pf=fprintdec;
  pf(i);
  pf=fprinthex;
  pf(i);
  return 0;
}
```

Beispiel

- Im Beispiel gibt es 2 Funktionen (f-printdec und fprintHex).
- Es gibt weiter eine Funktionspoin-tervariable pf vom Typ f.
- In main wird pf erst fprintdec, dann fprinthex zugewiesen und über pf jeweils ausgeführt.
- Im Ergebnis wird mittels pf(i); die Zahl einmal dezimal und einmal hexadezimal ausgegeben.

Funktionspointerparameter

Programmausschnitt

```
17  void fprintxx(void(*fctp)(int),int i)
18  {
19    fctp(i);
20  }
21  int main(int argc, char*argv[])
22  {
23    int i=atoi(argv[1]);
24    pf=fprintdec;
25    pf(i);
26    pf=fprinthex;
27    pf(i);
28    fprintxx(fprintdec,i);
29    fprintxx(fprinthex,i);
30    return 0;
31  }
```

- Die Funktion printxxx übernimmt als ersten Parameter einen Funktionspointer.
- Beim Aufruf wird der Name der zu übergebenden Funktion angegeben.
- Je nach übergebener Funktion erfolgt die Ausgabe dezimal/hexadezimal.

Funktionspointerreturnwert

Programmausschnitt

```
22  f choice(int i)
23  {
24    if (i) return fprinthex;
25    else   return fprintdec;
26  }
27
28  int main(int argc, char*argv[])
29  {
30    int i=atoi(argv[1]);
31    choice(0)(i);
32    choice(1)(i);
33    return 0;
34  }
```

- In diesem Programmausschnitt gibt es die Funktion choice.
- Die Funktion gibt einen Funktions-pointer auf fprintdec oder fprinthex je nach dem Wert von i zurück.
- Beim Aufruf werden zwei Prame-terklammerpaare angegeben. Die erste Prameterliste wird choice übergeben, die zweite dem Aufruf der zurückgegebenen Printfunkti-on.

Funktionspointerarray

Programmausschnitt

```
22  f choice(int i)
23  {
24    if (i) return fprinthex;
25    else   return fprintdec;
26  }
27
28  f arr[]={fprintdec, fprinthex};
29
30  int main(int argc, char*argv[])
31  {
32    int i=atoi(argv[1]);
33    arr[0](i);
34    arr[1](i);
35    return 0;
36  }
```

- In diesem Programmausschnitt wird ein initialisiertes Array von Funktionspointern angelegt (Zeile 28).
- In den Zeilen 33/34 erfolgt der Funktionsaufruf über das indizierte Array.

Komplettes Beispiel

```c
#include <stdlib.h>
#include <stdio.h>

typedef void(*f)(int i);
f pF;

void fprintdec(int i)
{
  printf("%d\n",i);
}

void fprinthex(int i)
{
  printf("%08x\n",i);
}

void fprintxx(f fctp,int i)
{
  fctp(i);
}

f choice(int i)
{
  if (i) return fprinthex;
  else   return fprintdec;
}
```

```c

f arr[]={fprintdec, fprinthex};

void main()
{
  int I=9999;
  pF=fprintdec;
  pF(I);
  pF=fprinthex;
  pF(I);
  fprintxx(fprintdec,I);
  fprintxx(fprinthex,I);
  choice(0)(I);
  choice(1)(I);
  arr[0](I);
  arr[1](I);
}

```

Anwendungen

Funktionspointer eignen sich zur Übergabe der Vergleichsfunktion an eine Sortierfunktion. Damit kann die Sortierung nach verschiedenen Kriterien erfolgen. Im Beispiel unten gibt es 3 Funktionsdeklarationen für Vergleichsfunktionen im Kontext des Währungsbeispiels (Prakt 8) und eine Sortierfunktion sort, die eine der drei Vergleichsfunktionen als Parameter übergeben bekommt.

```
int cmpLand(tWrg*,tWrg*);
int cmpLkz (tWrg*,tWrg*);
int cmpwKZ (tWrg*,tWrg*);
void sort(tWrg* pdata, int len, int (*cmp)(tWrg*,tWrg*));

sort (vWrg, sizeof vWrg/sizeof(tWrg), cmpLand); // Aufruf
```

Anwendungen

- Im Foliensatz zu dynamischem Speicher wird eine doppelt verkettete Ringliste vorgestellt. Die Funktion `addItemToList` soll darin ein neues Datenelemet so einfügen, dass es zwischen das kleinere und größere Element eingekettet wird. So kann eine Liste sortiert erzeugt werden.
- Da die Listenimplementation universell sein soll und keine Kenntnis über die von ihr verwalteten Daten hat. Wird der Vergleich der Daten in eine zu übergebende Funktion ausgelagert.

Funktionspointer als Funktionsparameter

Aufruf der übergebenen Funktion

```
void* addItemToList  (tList* pList,
                      void * pItem,
                      int(*fcmp)(void*pItList,void*pItNew))
{
   void* pItemBh;

   for (pItemBh =GetFirst(pList);
        pItemBh!=NULL && fcmp(pItemBh,pItem)<=0;
        pItemBh =GetNext(pList));
   if (pItemBh!=NULL) {if (InsertBefore(pList,pItem)==FAIL)return NULL;}
   else               {if (InsertTail  (pList,pItem)==FAIL)return NULL;}
   return pItem;
}
```

Anwendungen

Bei der Programmierung grafischer Benutzeroberflächen mit c dienen Funktionspointer dem Eventhandling. So wird ein Bedienelement erzeugt, in dem eine Struktur, die es beschreibt, erzeugt und in geeigneter Weise verwaltet wird. Diese Struktur enthält in der Regel auch Funktionspointer, die die Funktionalität des Bedienelements definieren (Was passiert, wenn der Butten betätigt wird). Man nennt diese Funktionen call-back-functions.

Das Erscheinungsbild und Verhalten des Bedienelements ist in Bibliotheken vorprogrammiert und oft in weiten Grenzen parametrisierbar, die Verbindung zur Anwendung wird über call-back-functions hergestellt.