

# Globale Variablen

## Typische Speicherübersicht eines C-Programms:

Globale Variablen ermöglichen

- einfachere Programmierung
- schnellen Zugriff zur Laufzeit (keine Stack-Kopie nötig)

ABER:

- verleiten zu „spezifischen“ Modulen – z.B.:

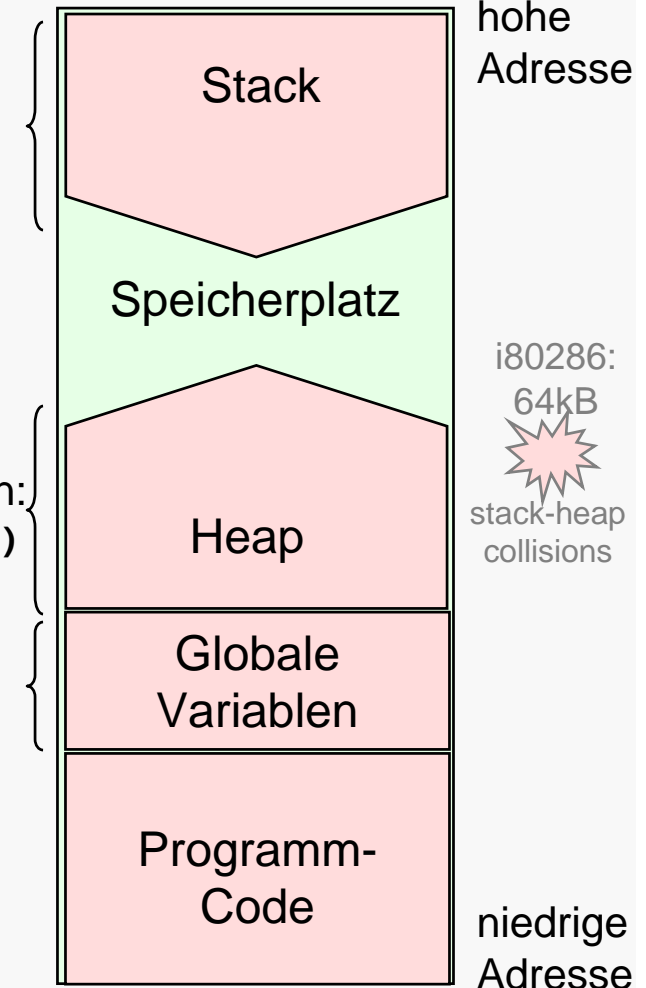
```
int x;  
int squareL(int x)  
{ return (x*x);  
}
```

```
int squareG(void)  
{ return (x*x);  
}
```

Parameterliste,  
lokale Variablen,  
Rückgabewerte,  
CPU-Status

dynamische Daten:  
`malloc()`, `new()`

statische Daten  
(vgl.: `static`)



Weitere Nachteile des Einsatzes globaler Variablen:

- Keine Interrupt-Sperrung während d.Parameter-Übergabe
    - ↳ Gefahr der Daten-Inkonsistenz (durch andere Threads o. unübersichtliche Folge von Aufrufen)
  - Kein Überblick über benötigte Variablen für eine Funktion
    - ↳ Erschwerung der Wiederverwendung von Code
  - Bezeichner-Reservierung über das ganze Programm
    - ↳ Fehlerquelle: lokale Variablen setzen globale außer Kraft
  - Verfügbarkeit von Daten über die gesamte Laufzeit
    - ↳ Erhöhung des Speicherbedarfs
  - Ansprechbarkeit von Variablen über das ganze Programm
    - ↳ Aufhebung aller Kapselung, Gefahr der Datenmanipulation
- ➔ Eliminierung globaler Variablen (fast) immer zu empfehlen, manchmal aber mit Arbeit verbunden.

z.B.: Libraries!

## Beispiel:

Zeichen eingeben und ausgeben:

```
int main(void)
{ char z=' ';
  while (EWIG)
  { printf ("\nGib Zeichen: ");
    z = _getch();
    if (z!=ENDE) printf("\nDas war ein: %c\n",z);
    else {printf("\nDanke!\n");_getch();exit(0);}
  }
  return 0;
}
```

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

#define ENDE      27
#define EWIG      1
```

(Globals\main\~\Test.exe)

## Forderung: Modularisierung / Trennung von Ein- u. Ausgabe

```
char z=' ';\nint Get(void)\n{\n    printf (\"\\nGib Zeichen: \");\n    z = _getch();\n    return 0;\n}\n\nint Put(void)\n{\n    if (z!=ENDE) printf(\"\\nDas war ein: %c\\n\", z);\n    else { printf (\"\\nDanke!\\n\"); _getch(); exit (0); }\n    return 0;\n}\n\nint main(void)\n{\n    while (EWIG)\n    {\n        Get(); /* Zeichen eingeben */\n        Put(); /* Zeichen ausgeben */\n    }\n    return 0;\n}
```

```
#include <conio.h>\n#include <stdio.h>\n#include <stdlib.h>\n\n#define ENDE    27\n#define EWIG    1
```

(Globals\\Glob\\~\\Test.\*)

Forderung: Modularisierung / Trennung von Ein- u. Ausgabe  
... ohne globale Variablen:

```
int common (char rw, char *zeichen)
{ static char __zeichen=' ';/*ohne static: wirkungslos!*/
  if (rw == READ) *zeichen = __zeichen;
  else             __zeichen = *zeichen;
  return 0;
}
```

```
int main(void)
{ while (EWIG)
  { Get(); /* Zeichen eingeben */
    Put(); /* Zeichen ausgeben */
  }
  return 0;
}
```

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

#define ENDE      27
#define EWIG      1
enum {READ, WRITE};
```

# Globale Variablen

```
int common (char rw, char *zeichen)
{ static char _zeichen=' '; /*ohne static: wirkungslos!*/
  if (rw == READ) *zeichen = _zeichen;
  else             _zeichen = *zeichen;  return 0;
}

int Get(void)
{ char zeichen=' '; /* static moeglich, nicht noetig */
  printf ("\nGib Zeichen: ");  zeichen = _getch();
  common (WRITE, &zeichen);    return 0;
}

int Put(void)
{ char zeichen=' ';
  common (READ, &zeichen);
  if (zeichen!=ENDE) printf ("\nDas war ein: %c \n", zeichen);
  else { printf ("\nDanke, das war's!\n");_getch(); exit (0);}
  return 0;
}

int main(void)
{ while (EWIG)  { Get(); /* Zeichen eingeben  */
                 Put(); /* Zeichen ausgeben  */
                 } return 0;
}
(Globals\nnoGlob\~\Test.*)
```

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#define ENDE    27
#define EWIG    1
```

```
int common (int (**pp2f)(void), char *zeichen);
int Get(void);
int Put(void);
```

```
int main(void)
{ while (EWIG)
  { Get(); /* Zeichen eingeben u. weiterschauen */
  }
  return (0);
}
```

# Globale Variablen - und Callbacks

```
int Get(void)
{ char zeichen=' '; static int (*p2f)(void); // Zeiger-Var.
  printf ("\nGib Zeichen: ");    zeichen = _getch();
  common (&p2f, &zeichen);      (*p2f)();      // Aufruf
  return (0);                  // p2f ();      // auch zulaessig
}

int common (int (**pp2f)(void), char *zeichen)
{ static char _erst=1, _zeichen=' ', _write=1;
  if (_erst) { *pp2f=Put; _erst=0; } /*Nur fuer Get()!*/
  if (_write) _zeichen=*zeichen; else *zeichen=_zeichen;
  _write = 1 - _write;
  if(*zeichen==ENDE){printf("\nDanke!\n");_getch();exit(0);}
  return (0);
}

int Put(void)
{ char zeichen=' '; int (*dummyP2f)();
  common (&dummyP2f, &zeichen);
  printf ("\nDas war ein: %c \n", zeichen);
  return (0);
}

(Globals\noGlobCall\~\Test.*)
```



# Globale Variablen - und Callbacks

Einführung von: `int (*common_(char **zeichen))(void);`

Die Funktion: `common_(char **zeichen)`

habe als Rückgabewert R :

... die Adresse einer anderen Funktion... `(*R)()`

...mit eigener leerer Parameter-Liste... `(*R)(void)`

... und eigenem Rückgabewert `int`: `int (*R)(void)`

- dann heißt der Prototyp von `common_()`:

`int (*common_(char **zeichen))(void)`

# Globale Variablen - und Callbacks

```

int (*common_ (char **zeichen))(void);

int Get(void)
{ static char *zeichen=NULL;          //NULL statt _erst
  if(!zeichen) common_(&zeichen);    //Speicherg in common_()
  printf("\nGib Zeichen: "); *zeichen = _getch();
  (*common_(&zeichen))();             //Aufruf ueber Rueckgabe-Pnt.
  return (0);
}

int (*common_ (char **zeichen))(void)
{ static char _zeichen=' ';           //Auch als Feld ausbaubar
  if(!*zeichen) { *zeichen=&_zeichen; } //Initialisierg
  if(**zeichen==ENDE){printf("\nDanke!\n");_getch();exit(0);}
  return(Put);
}

int Put(void)
{ char *zeichen=NULL;
  if (!zeichen) common_ (&zeichen);
  printf ("\nDas war ein: %c \n", *zeichen);
  return (0);
}

```

