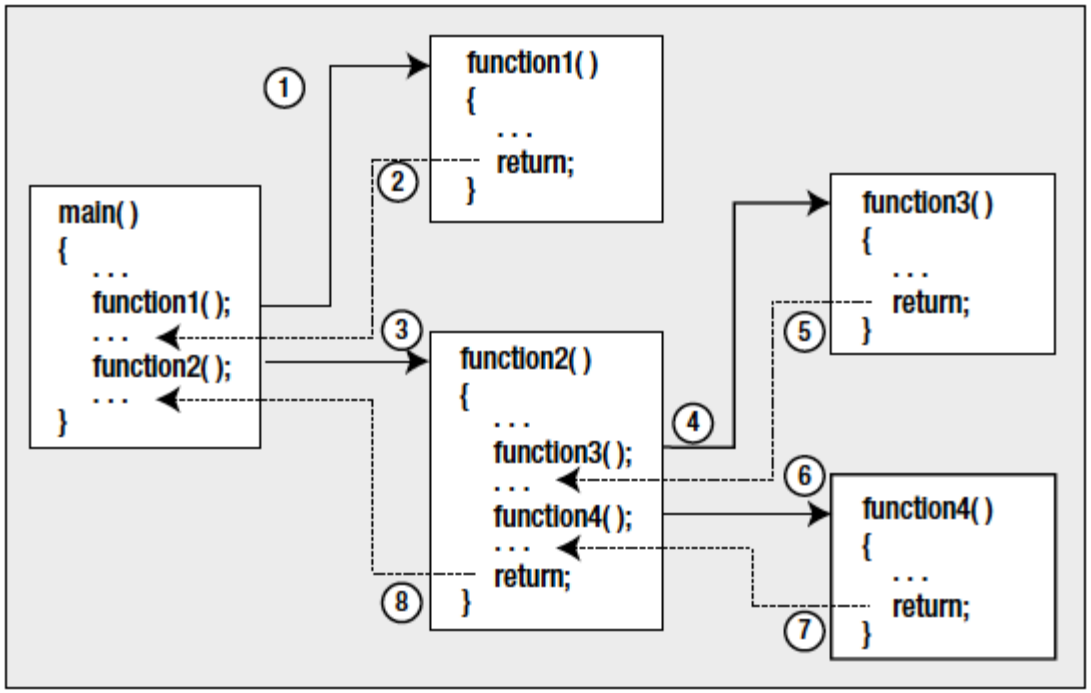


Programm Struktur

Programm Struktur

- Variable Scope und Lifetime
- Variable Sope und Functions
- Functions Parameter
- Pointer als Parameter und Return Types

Variable Scope und Lifetime

```
{
int a = 0; // ...
// Reference to a is ? here
// Reference to b is ? here
{
int b = 10; // ...
// Reference to a and b is ? here
} // ...
// Reference to b is ? here
// Reference to a is ? here
}
```

- Variablen innerhalb von Blocks werden auch als “automatische” Variablen bezeichnet. Da automatisch angelegt und zerstört.

Variable Scope und Lifetime (2)

```c
int main(void)
{
    int count = 0; // Declared in outer block
    do
    {
        int count = 0; // This is another variable called count
        ++count; // this applies to inner count
        printf("count = %d\n", count);
    }
    while( ++count <= 5); // This works with outer count

    printf("count = %d\n", count); // Inner count is dead, this is outer count
    return 0;
}
```

- Was wird ausgegeben?
- Begründe deine Antwort

```
count = 1
count = 1
count = 1
count = 1
count = 1
count = 1
count = 6
```

Variable Scope und Functions

- Der Body von jeder Function ist ein Block
- Die automatischen Variablen sind local und nur innerhalb der Function gültig.
- Die Variable Deklaration einer Function ist unabhängig von anderen Variablen in anderen Functions

Functions

- Bestehend aus
  - Function Header
  - Function Body

```
Return_type Function_name ( Parameters –
separated by commas)
{
  // Statements…
}
```

Functions **Parameters**

Die Namen der Parameter sind lokal nur innerhalb der Funktion sichtbar.

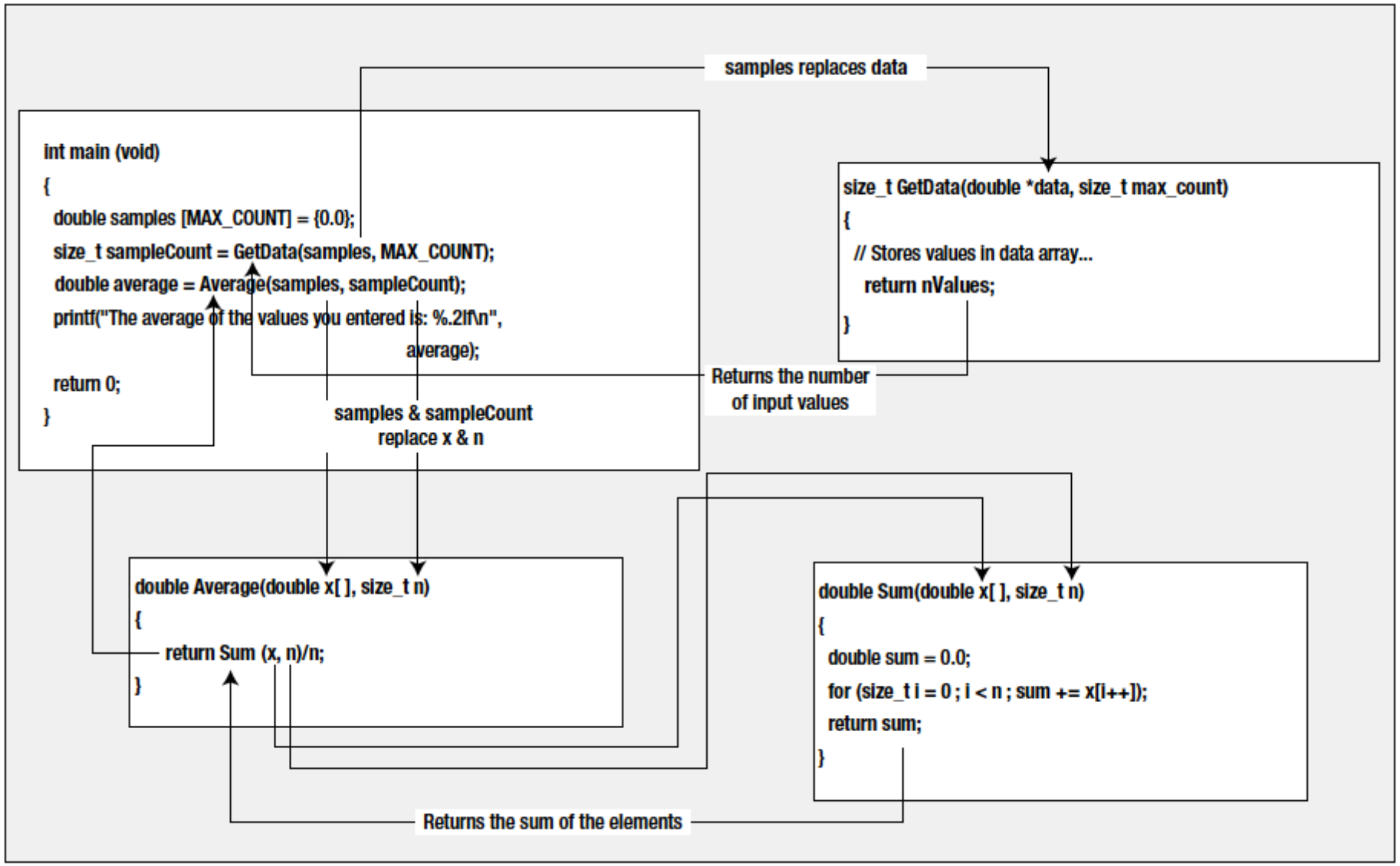
Beispiele:

```
bool SendMessage(char *text)

void PrintData(double *data, int count)

int SumIt(int x[], size_t n)

char* GetMessage(void)
```

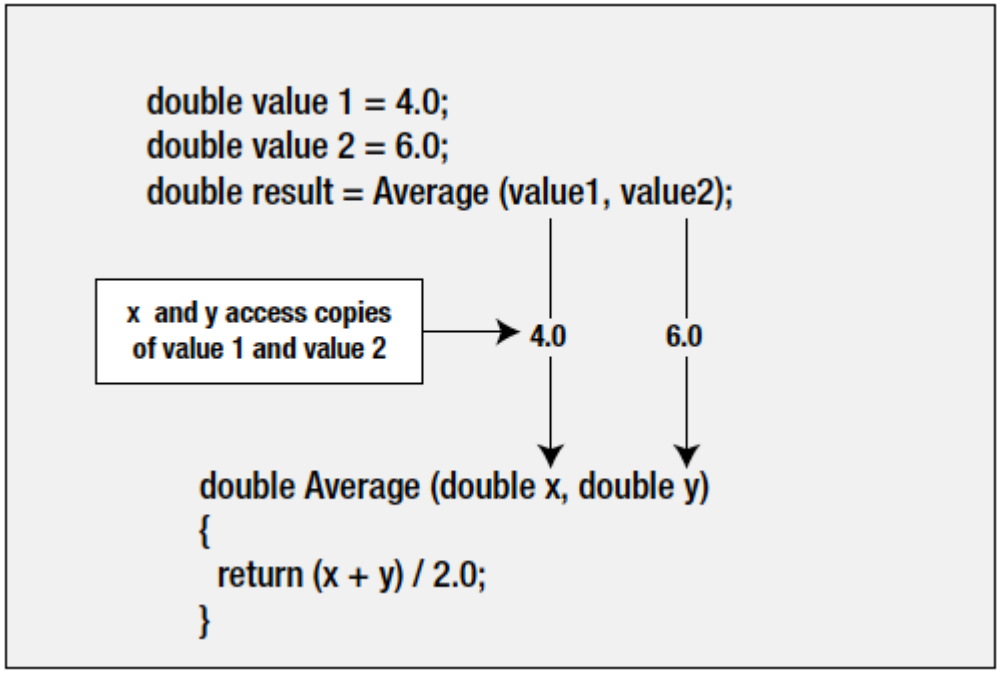
Beispiel - Programm Struktur

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#define MAX_COUNT 50

// Function to calculate the sum of array elements
// n is the number of elements in array x

double Sum(double x[], size_t n)
{
    double sum = 0.0;
    for(size_t i = 0 ; i < n ; ++i)
    sum += x[i];
    return sum;
}

// Function to calculate the average of array elements
double Average(double x[], size_t n)
{
    return Sum(x, n)/n;
}

// Function to read in data items and store in data array
// The function returns the number of items stored
size_t GetData(double *data, size_t max_coun t)
{
    size_t nValues = 0;

    printf("How many values do you want to enter (Maximum %zd)? ", max_count);

    scanf_s("%zd", &nValues);
    if(nValues > max_count)
    {
        printf("Maximum count exceeded. %zd items will be read.", max_count);
        nValues = max_count;
    }

    for( size_t i = 0 ; i < nValues ; ++i)
        scanf_s("%lf", &data[i]);

    return nValues;
}

// main program - execution always starts here
int main(void)
{
    double samples[MAX_COUNT] = {0.0};
    size_t sampleCount = GetData(samples, MAX_COUNT);
    double average = Average(samples, sampleCount);
    printf("The average of the values you entered is: %.2lf\n", average);

    return 0;
}
```

Beispiel - Programm Struktur



Pass-By-Value versus Pass-By-Reference




```
double Sum(double x[], size_t n)
{
double sum = 0.0;
for(size_t i = 0 ; i < n ; ++i)
sum += *(x++);
return sum;
}
```

Functions Prototypes

```
// #include & #define directives...

// Function prototypes
double Average(double data_values[], size_t count);
double Sum(double *x, size_t n);
size_t GetData(double*, size_t);

int main(void)
{
    // Code in main() ...
}

double Average(double x[], size_t n)
{
    return Sum(x, n)/n;
}

double Sum(double x[], size_t n)
{
    // Statements...
}

size_t GetData(double *data, size_t max_count)
{
    // Statements...
}
```

Pointer als Parameter und Return Types

- const Parameter

```
bool SendMessage(const char *text)
// text kann nicht verändert werden!
```

- const Parameter ist nur sinnvoll bei Pointer Parameter
  - Call by Value
- Vorteil
  - Compiler Prüfung!
    - Error wenn const -Daten verändert werden
    - Der Übergebene Pointer muss auf const zeigen
  - Impliziert, dass die Funktion keine Änderungen vornimmt

Pointer als Parameter und Return Types (2)

- Probiere es aus:
  - Analysiere das Programm `sentencesorter.c`
    - Kommentiere die Funktionsheader und die definierten Parametertypen
    - Implementiere die fehlenden Funktionsblöcke
    - Kommentiere das Programm vollständig

Gefahren bei Pointer Rückgabe (Returning Pointer)

- Ein numerischen Wert wird als Kopie zurückgeben
- Mit Hilfe von Pointer können mehrere Werte zurückgegeben werden (wobei auch der Pointer als Kopie zurückgeben wird)
  - z.B. String
- Aber, es gibt Gefahren beim Zurückgeben von Pointer

Gefahren bei Pointer Rückgabe (Returning Pointer) (2)

```c
#include <stdio.h>

long *IncomePlus(long* pPay); // Prototype for increase pay function

int main(void)
{
    long your_pay = 30000L; // Starting salary
    long *pold_pay = &your_pay; // Pointer to pay value
    long *pnew_pay = NULL; // Pointer to hold return value

    pnew_pay = IncomePlus(pold_pay);

    printf("Old pay = $%ld\n", *pold_pay);
    printf(" New pay = $%ld\n", *pnew_pay);

    return 0;
}

// Definition of function to increment pay
long* IncomePlus(long *pPay)
{
    *pPay += 10000L; // Increment the value for pay

    return pPay; // Return the address
}
```

Was ist der Output?

```
Old pay = $40000
  New pay = $40000
```

Gefahren bei Pointer Rückgabe (Returning Pointer) (2)

```c
#include <stdio.h>

long *IncomePlus(long* pPay); // Prototype for increase pay function

int main(void)
{
    long your_pay = 30000L; // Starting salary
    long *pold_pay = &your_pay; // Pointer to pay value
    long *pnew_pay = NULL; // Pointer to hold return value

    pnew_pay = IncomePlus(pold_pay);

    printf("Old pay = $%ld\n", *pold_pay);
    printf(" New pay = $%ld\n", *pnew_pay);

    return 0;
}

// Definition of function to increment pay
long *IncomePlus(long *pPay)
{
    long pay = 0; // Local variable for the result
    pay = *pPay + 10000; // Increment the value for pay

    return &pay; // Return the address of the new pay
}
```

Was ist der Output?

```
Old pay = $30000
  New pay = $27467656
```