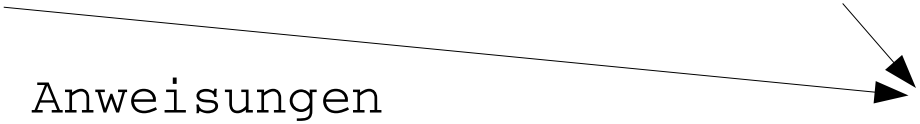
Funktionen

Zum Strukturieren komplexerer Programme werden funktionale Einheiten ausgelagert.

- Syntax

```
returnType FunctionName (ParameterList)
{
    C++ Anweisungen
}
```



falls Rückgabewert
oder Parameter leer

- Funktionen stehen meistens hinter dem „Hauptprogramm“ int main() { } Dabei ist die Reihenfolge nicht von Bedeutung.

- Sie werden vor dem Hauptprogramm deklariert

```
returnType FunctionName (ParameterList);
```

Die Parameterliste muss nur die Typ Deklaration enthalten

- Die Funktionsdeklarationen werden häufig in Files mit der Endung .h ausgelagert. Das Einfügen erfolgt erfolgt mit

```
#include “myHeader.h“
```

Funktionen – Ein Beispiel

- Beispiel

```
int max ( int N , int M )
{
    int result ;
    if ( N > M )
      result = N ;
    else
      result = M ;
    return result ;
}

int max ( int , int);
```

Rückgabe des Maximums 2er Zahlen

Parametertyp deklarieren

Funktionstyp deklarieren

Lokale Variable, nur in der Funktion bekannt

Rückgabewert

steht vor dem main Programm

Parameter Liste enthält nur die Typ Deklaration

Funktionen – Ein Beispiel

- Beispiel

```
int max ( int N , int M )
{
    int result ;
    if ( N > M )
      result = N ;
    else
      result = M ;
    return result ;
}

int max ( int , int);
```

steht vor dem main Programm

Rückgabe des Maximums 2er Zahlen

Parametertyp deklarieren

Funktionstyp deklarieren

Lokale Variable, nur in der Funktion bekannt

Rückgabewert

functionSum.cc

Arbeitsvorschlag:

- Schreiben Sie ein Programm, das in einer Funktion 2 Zahlen summiert.
- Erzeugen Sie eine header Datei mit der Funktionsdeklaration und binden Sie sie ein.
- Lagern Sie die header Datei in ein include dir aus. Mit der Compiler Option -I dir koennen Sie das directory einbinden

Funktionen - Parameterübergabe

Beim Übergeben der Funktionsargumente ist zu beachten, das die Speicherbereiche beim Aufruf der Funktion erzeugt werden und beim Verlassen der Funktion nicht mehr existieren!

- Methoden zur Übergabe von Funktionsparametern
  - Call by value (default)

```
double function (int A, int B) { }
```

A und B werden beim Funktionsaufruf kopiert und Änderungen innerhalb der Funktion haben keinen Effekt auf die Werte A und B im Hauptprogramm

  - Call by reference

```
double function (int &A, int &B) { }
```

Die Adressen von A und B werden beim Funktionsaufruf übergeben und Änderungen der Adressinhalte innerhalb der Funktion beeinflussen die Werte A und B im Hauptprogramm

```
void swap( int &N , int &K ){
    int temp;  temp = N;    N = K;    K = temp;
    return ;
}
```

Alternativ können auch pointer übergeben werden

functionSwap_I.cc

functionSwap_II.cc

Funktionen - Parameterübergabe

Arrays werden häufig an die Funktionen als Zeiger übergegeben. Dabei wird die Arraylänge als konstanter Wert ebenfalls übergegeben.

addArray.cc

```
// test function feature of pointer to array
#include <iostream>
using namespace std;

void sum (const int i , double *a , double *b, double *c) ;

int main()
{
  const int i = 5 ;
  double ValA[i]={2.,3.,4.,5.,8.};
  double ValB[i]={20.,30.,40.,50.,80.};
  double ValC[i];

  sum (i,ValA,ValB,ValC);
  cout << "Sum = " << ValC[2] << endl;

  return 0;
}

// Add 2 double
void sum (const int i , double *a , double *b, double *c) {
  for (int k=0; k < i ; k++)  c[k] = a[k] + b[k]  ;
}
```

Der Inhalt von Array ValC wird in der Funktion sum modifiziert. Die Werte werden by Reference zurückgegeben, Daher sind die Änderungen sichtbar.

Funktionen - Parameterübergabe

Komplexe Datenstrukturen werden aus Funktionen in der Regel als Zeiger zurückgegeben. Im folgenden ist ein Beispiel mit strings implementiert

charpointer.cc

```
#include <iostream>
#include <cstring>
#include <stdio.h>
#define MAXCHAR 255

using namespace std;

char *readinput (char *ptr) ;
int main() {
   char inA [] = "Vorname" ;
   char inB [] = "Nachname" ;
   char *ptr;

   ptr = readinput(inA);
   cout << "Hi " << ptr  << endl;
   ptr = readinput(inB);
   cout << ptr << " ? Ich bleib beim Vornamen! " << endl;
   return 0 ;
}

char *readinput (char *str) {
   char input[MAXCHAR];
   cout << "Bitte " << str << " eingeben: " << endl;
   fgets(input, MAXCHAR, stdin);
   return strtok(input, "\n");
}
```

Die Funktion readinput gibt eine Zeichenkette zurück bzw die Adresse des Anfangs der Zeichenkette

liest von stdin bis newline gefunden

Rückgabewert ist ein char pointer newline ist entfernt

Funktionen – statische Variablen

Die lokalen Variablen von Funktionen existieren nur für die Lebensdauer der Funktion. Zugewiesene Werte existieren nach dem Aufruf nicht mehr.

- Statische Variable

```
static Datentyp Name ;
```

Diese Anweisung definiert eine Variable, deren Inhalt nach dem Schliessen der Funktion erhalten bleibt und bei erneutem Aufruf zur Verfügung steht.

- Globale Variable

```
Datentyp myGlobalVariable ;
…..
void function(){ }
main() {
…..
}
```

Variablen, die ausserhalb von Funktionen und vor dem Hauptprogramm definiert werden, stehen in allen Funktionen zur Verfügung und behalten ebenfalls den Inhalt nach dem Schliessen der Funktionen.

Überladen von Funktionen

Wenn verschiedene Funktionen unter dem gleichen Namen angesprochen werden können, sprechen wir von überladenen Funktionen. Dabei ist nur eine Änderung der Anzahl der Parameter in der Parameterliste oder der Parametertypen zulässig.

```
double multiply (int a , int b){
return (double) (a*b) ; }

double multiply (double a , double b){
return a*b ; }

double multiply (int a , int b , int c) {
return (double) (a*b*c) ; }

double multiply (double a , int b , int c) {
return a * (double)(b*c) ; }
```

Rekursive Funktionen

Der Selbstaufruf von Funktionen wird als Rekursion bezeichnet. Eine Abbruchbedingung sorgt dabei für eine endliche Anzahl von Aufrufen.

```
unsigned int fakultaet(unsigned int zahl) {
    // Die Fakultät von 0 und 1 ist als 1 definiert.
    if (zahl <= 1) return 1;
    return fakultaet (zahl - 1) * zahl;
}
```

- Bei einer rekursiven Lösung wird ein Problem immer wieder schrittweise in ein einfacheres Problem verwandelt.
- Rekursive Programme können fast immer auch iterativ implementiert werden. Meist führt das jedoch zu komplizierterem Code. Rekursive Lösungen sind in der Regel rechenzeit- und speicherintensiver.

Inline Funktionen

Der Aufruf von Funktionen wird durch das Schlüsselwort inline verändert. Der Programmcode wird durch den Compiler direkt in das Programm eingefügt.

```
inline returnType myFunction( ParameterList ) {
   C++ code ;
}
```

Diese Anweisung ist für den Compiler nicht bindend. Typischerweise werden nur kurze Funktionen direkt eingefügt.

Funktionen – Eingabe Parameter

Der Funktion main() können auch Parameter übergeben werden. Die vollständige Definition ist `int main(int argc , char *argv[])`

Dabei gibt argc die Anzahl der Argumente an; im vector argv[] stehen Pointer auf die übergebenen Parameter. In argv[0] steht dabei immer der Programmname. Hilfsfunktionen zum entpacken der Parameter stehen über die Standard Bibliotheken zur Verfügung.

readOptions.cc

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   for(int i=0; i < argc; i++)
      {
         printf("argv[%d] = %s ",i,argv[i]);
         printf("\n");
      }
   return 0;
}
```

testOptions.cc

Komplexere Beispiele finden Sie hier

adderOptions.cc

Funktionen - Bibliotheken

Funktionen können in Bibliotheken als Objektfiles sowohl statisch (.a) als auch dynamisch (.so) gespeichert werden.

- Erzeugen einer statischen Bibliothek

  Der Quellcode ist die function myFunc.cc und die header Datei myFunc.h

  - compilieren der Quelldateien und erzeugen des Objektfiles

    ```
    gcc -c myFunc.cc -o myFunc.o
    ```

  - verwenden der archiver ar zur Erzeugung der statischen Bibliothek

    ```
    ar rcs libmyFunc.a myFunc.o
    ```

    Der Bibliotheksname muss die ersten 3 Buchstaben lib enthalten und mit .a enden.
    Im link Schritt von gcc muss -static hinzugefuegt werden.

- Erzeugen einer dynamisch gelinkten Bibliothek

  - compilieren der Quelldateien und erzeugen des Objektfiles

    ```
    gcc  -fPIC -c myFunc.cc -o myFunc.o
    ```

    Die Option -fPIC erzeugt positionsunabhaengigen code

  - verwenden von gcc zur Erzeugung der dynamischen Bibliothek

    ```
    gcc -shared -Wl,-soname,libmyFunc.so -o libmyFunc.so myFunc.o
    ```

  - beim link Schritt in gcc zum Einbinden der dynamischen Bibliothek reicht die Endung .so der Bibliothek

    ```
    g++ myProg.cc -L. -lmyFunc
    ```

Funktionen – Werkzeug Bibliotheken

Es existiert eine C++ Bibliothek, die standardisierte Werkzeugprogramme enthält (C++ Standard Library). Die C Bibliothek ist darin enthalten.

- Standard Funktionen aus den Bereichen

  Mathematische Funktionen, Input / Output, Zeichenketten, Zeit / Datum, Speichermanipulation, …….

```
#include <cmath>
   …..
   double t, x;
   x = sin(t);
   x = log(t);
   x = exp(t);
   x = sqrt(t);
   …..
```

Erzeugung von Zufallszahlen:

```
#include <cstdlib>
#include <ctime>
   …..
   // initialize with time
   srand(time(0));
   // random integer
   int r = rand();
   …..
```

- Objektorientierte Klassenbibliothek

  Klassendefinitionen für Input/Output, Strings, Standard Template, Container, Iteratoren, Fehlerbehandlung, …….

Die C++ Standard Library wird im g++ Link Schritt automatisch hinzugefügt.

Funktionen – Werkzeug Bibliotheken (2)

Die C++ Standard Library wird bei der Verwendung des g++ Compilers im Link Schritt automatisch hinzugefügt.
Neben der C++ Standard Library gibt es viele kommerzielle und public domain Bibliotheken mit spezieller Funktionalität. Vor dem Programmieren einer speziellen Softwarelösung sollten existierende Bibliotheken durchgesehen werden.

- Bibliotheken - Beispiele

  boost freie C++ Bibliothek (mathematische Lösungen, …..)
  OpenMP Paralleles Computing
  cuda Programmieren auf Nvidia Graphikkarten

- Verwendung von Bibliotheken
  - Hinzufügen des header files im Programmcode

    ```
    #include <boost/random.hpp>
    ```

  - Aufrufen der Funktion im Hauptprogramm

    ```
    gen()
    ```

  - Hinzufügen der Bibliothek im Link Schritt des Compilers

    ```
    g++ myCode.cc -L/usr/lib64/boost/ -lboost_random
    ```

    (/usr/lib64/boost/libboost_random.so wird hinzugefügt)

Arbeitsvorschlag:

- Schreiben Sie ein Programm, das die 3. Wurzel aus einer Zahl mit der Newton Methode bestimmt.

- Wie sieht der Algorithmus aus?

- Der Wert aus dem die Wurzel gezogen werden soll und die Genauigkeit sind im Programm festgelegte Parameter.

KubikWurzelNewton_0.cc

- Verwenden Sie nun command line Eingaben um den Wert aus dem die Wurzel gezogen werden soll und die Genauigkeit zu übergeben.

KubikWurzelNewton_I.cc

- Der Wert und die Genauigkeit sollen in beliebiger Reihenfolge eingegeben werden können.

KubikWurzelNewton_II.cc

-Benutzen Sie Funktionen und schreiben sie diese in die Bibliothek libmyFunc.a in MeinArbeitsDirectory/lib

KubikWurzelNewton_III.cc

TestLibrary_KubikWurzelNewton.txt

Rezept zur Verwendung von Bibliotheken am Beispiel von KubikWurzelNewton

Funktions-Templates

C++ beinhaltet Möglichkeiten zur generischen Programmierung. Ein Funktions-Template verhält sich wie eine Funktion, die Argumente verschiedener Typ Definitionen bzw Rückgabe Wert Typen akzeptiert. Die Funktionen enthalten dazu Platzhalter Typ Definitionen.

```
// Beispiel einer Template Funktion
#include <iostream>
using namespace std;

template <typename Type>
Type maximum ( Type a , Type b ) {
  return ( a > b ? a : b ) ;
}
int main() {
  int x = 5 , y = 7 ;
  int m = maximum <int> (x,y) ;
  cout << "maximum of " << x << " and " << y << " is " << m << endl;

  char a = 'a' , b = 'b' ;
  char r = maximum <char> (a,b) ;
  cout << "maximum of " << a << " and " << b << " is " << r << endl;

  return 0 ;
}
```

Platzhalter Typ Definition

Angabe des verwendeten Typs

funcTemplate.cc

Funktionen – Werkzeug Bibliotheken

Standard C++ library ist eine Programmierbibliothek, die auch die C Bibliotheken enthält.

- Beispiel Container – vector class

  vector ist eine container class, die dynamische arrays implementiert

```
#include <vector>
    …..
    vector <int> myVector;
    vector <int> p;
    vector <double> yourVector(10,5.0)
    int iElem = 10 ;
    myVector.push_back(iElem);
    myVector.push_back(74);
    myVector.pop_back();
    int myVectorSize = myVector.size();
    for(int i=0;i<yourVector.size();i++)
     cout << "Elem "<<i<<" is " << yourVector[i]<< endl;
    …..
```

Intialisieren von vector

Hinzufügen eines Elements am Ende von myVector

Löschen des letzten Elements von myVector

Grösse von myVector

Schleife über alle Elemente von yourVector

Beispiel Vector Class

readFile_0.cc

```
// read text File with 4 columns
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main()
{
  char InputFileName[256] = "myDataFile.txt" ;
  // open a file in read mode.
  ifstream inFile ( InputFileName );

  double colA , colB , colC , colD ;
  vector<double> vColA , vColB , vColC , vColD ; // instanciate
  int nRow = 0 ;
  if (inFile.is_open()) {
      while ( inFile >> colA >> colB  >> colC >> colD ) { // read one row
        vColA.push_back(colA);
        vColB.push_back(colB);
        vColC.push_back(colC);
        vColD.push_back(colD);
      }
      nRow = vColA.size() ; // number of elements in the vector
      inFile.close();
  }
  else {
    cerr << "Can not open file " << InputFileName << endl;
  }

  return 0;
}
```

Filename des Text Files

Öffnen des Files

Schleife über alle Einträge des Files

Schliessen des Files

Beispiel Vector Class

```
// read text File with 4 columns
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main()
{
  char InputFileName[256] = "myDataFile.txt" ;
  // open a file in read mode.
  ifstream inFile ( InputFileName );

  double colA , colB , colC , colD ;
  vector<double> vColA , vColB , vColC , vColD ; // instanciate
  int nRow = 0 ;
  if (inFile.is_open()) {
      while ( inFile >> colA >> colB  >> colC >> colD ) { // read one row
        vColA.push_back(colA);
        vColB.push_back(colB);
        vColC.push_back(colC);
        vColD.push_back(colD);
      }
      nRow = vColA.size() ; // number of e
      inFile.close();
  }
  else {
    cerr << "Can not open file " << InputF
  }

  return 0;
}
```

readFile_0.cc

Filename des Text Files

Öffnen des Files

Schleife über alle Einträge des Files

readFile_I.cc

Arbeitsvorschlag:

- Schreiben Sie ein Programm, das die Werte aus dem Textfile mit 4 Spalten liest und unter Verwendung von vector die Mittelwerte und Standardabweichungen in einer Funktion bestimmt und die Werte in ein File schreibt.

Strings

Statt Zeichenketten in ein `char array` zu schreiben lässt sich auch die Standard Library Klasse `string` verwenden. Eine explizite Angabe der Länge der Zeichenkette ist hier nicht notwendig.

- Syntax und einige wichtige Funktionen

```
#include <string>
…...
string myStrA = “Warum“   ,  myString ;
string myStrB = “ auch das noch“
myString = myStrA + myStrB;
cout << myString << endl;    →  Warum auch das noch
…...
int nLength = myString.length(); → nLength = 19
pos = myStrA.find(“um“); → pos = 3
myStrA.insert(5,“ nicht“); →  Warum nicht
myStrB.erase(5,9); →  auch
myStrA.c_str();
```

Deklaration der `string` Variablen

Strings lassen sich durch addieren aneinander hängen

Länge eines strings

Finden eines sub strings

Einfügen / löschen eines sub strings

Gibt eine nicht-modifizierbare Standard C Zeichenarray Version der Zeichenfolge zurück

Viele Verbesserungen in C++11

Sortieren von Arrays

Es ist eine beliebige Folgen von ganzen Zahlen gegeben, die durch Algorithmen aufsteigend sortiert werden soll. Die Algorithmen sollen dabei möglichst effizient vorgehen, d.h. die Zahl der durchzuführenden Operationen pro Element soll klein sein.

- Sortieren durch Einfügen

  Gehe wie beim Sortieren von Spielkarten vor. Es gibt 3 Gruppen
  - I. links stecken die sortierten Karten
  - II. eine einzusortierende Karte
  - III. rechts ein unsortierter Bereich mit Karten

  Algorithmus:
  Nimm die erste Karte und betrachte sie als sortiert (I.)
  Nimm die nächste Karte (II.) und suche die Position für die Karte im linken sortierten Teil (I.).
  Nimm die nächste Karte im rechten unsortierten Teil (III.)

  Beispiel: 111 | 120 | 555 90 100
  0 1 2 3 4

  Element 0 ist sortiert
  Schleife über Element 1 bis 4

Sortieren von Arrays

Es ist eine beliebige Folgen von ganzen Zahlen gegeben, die durch Algorithmen aufsteigend sortiert werden soll. Die Algorithmen sollen dabei möglichst effizient vorgehen, d.h. die Zahl der durchzuführenden Operationen pro Element soll klein sein.

- Sortieren durch Einfügen

  Beispiel: 111 | 110 | 555 90 100
  0 1 2 3 4

  Element 0 ist sortiert

  Schleife über Element K=1 bis 4

  K=1: 110 ist kleiner als 111, füge 110 vor 111 ein

  110 111 555 90 100
  0 1 2 3 4

  K=2: 555 ist grösser als 111 (letzte sortierte Element), 555 bleibt am Platz

  110 111 555 90 100
  0 1 2 3 4

  K=3: 90 wird an den Anfang geschoben

  90 110 111 555 100
  0 1 2 3 4

  K=5: 100 rückt an die 2. Stelle und alle anderen rücken 1 Position vor

  90 100 110 111 555
  0 1 2 3 4

Sortieren von Arrays

Es ist eine beliebige Folgen von ganzen Zahlen gegeben, die durch Algorithmen aufsteigend sortiert werden soll. Die Algorithmen sollen dabei möglichst effizient vorgehen, d.h. die Zahl der durchzuführenden Operationen pro Element soll klein sein.

- Sortieren durch Einfügen

```
int Array[n];
int K, J ,EinZuOrdnen;
for (K=1, K < N, K++){
   EinZuOrdnen = Array[K];
   J = K-1;
   while ( J >= 0 && Array[J] > EinZuOrdnen) {
      Array[J+1] = Array[J];
      J--;
   }
   Array[J+1] = EinZuOrdnen;
}
```

zu sortierendes Array der Grösse n

Schleife vom 2. Element bis Ende

Zeiger auf das letzte sortierte Element

Verschiebe die Elemente von Array[0..K-1] eins weiter nach vorn, wenn sie grösser als das einzuordnende Element sind. Setze J an die einzuordnende Stelle.

Setze einzusortierende Zahl ins Array

Sortieren von Arrays

Es ist eine beliebige Folgen von ganzen Zahlen gegeben, die durch Algorithmen aufsteigend sortiert werden soll. Die Algorithmen sollen dabei möglichst effizient vorgehen, d.h. die Zahl der durchzuführenden Operationen pro Element soll klein sein.

- Sortieren durch Einfügen

```
int Array[n];
int K, J ,EinZuOrdnen;
for (K=1, K < N, K++){
   EinZuOrdnen = Array[K];
   J = K-1;
   while ( J >= 0 && Array[J] > EinZuOrdnen) {
      Array[J+1] = Ar
      J--;
   }
   Array[J+1] = EinZu
}
```

zu sortierendes Array der Grösse n

Schleife vom 2. Element bis Ende

Zeiger auf das letzte sortierte Element

Setze einzusortierende Zahl ins Arr

wuerfeln_I.cc

Arbeitsvorschlag:

- Schreiben Sie ein Programm:
  Würfeln mit j Würfeln, der i Flächen hat und es soll k mal gewürfelt werden. i, j, k werden dem ausführbaren Programm gegeben. Das Würfelergebnis soll einmal unsortiert und einmal sortiert ausgegeben werden.

Sortieren von Arrays

- Quicksort Algorithmus

  Sortiere beliebige Folge von ganzen Zahlen, A[0,n-1]

  I. Wähle ein Element aus der Folge, Pivotelement A[p] so das A in einen linken A[L] und einen rechten A[R] Teil geteilt wird.

  II. Ordne A[L] so das alle Elemente groesser als das Pivotelement sind, A[L] >= A[p]
  Ordne A[R] so das alle Elemente kleiner als das Pivotelement sind, A[R] <= A[p]
  Das Ordnen erfolgt durch Vertauschen der linken und rechten Elemente

  III. Wiederhole I. und II. solange fuer die linken und rechten Felder bis jeweils nur 2 Elemente uebrig sind.

- Anschauliche Darstellung verschiedener Sortier-Algorithmen

  http://www.bluffton.edu/~nesterd/java/SortingDemo.html

gcc preprocessor Anweisungen

Preprocessor Anweisungen stehen vor dem eigentlichen C/C++ Quelltext und sind durch # gekennzeichnet. Sie werden vor dem Compile Schritt bearbeitet, d.h. alle Anweisungen werden in den Qelltext integriert. Die preprocessor Anweisung endet sobald \n (newline) gefunden wurde, “;“ ist daher nicht notwendig. Anweisungen über mehrere Zeilen werden mit “\“ fortgesetzt.

- Eigenschaften
  - Ersetzen von Macros, die mit `#define` definiert wurden
  - Ersetzen von Kommentaren `/* */` durch Leerzeichen(`//` ist nicht erlaubt)
  - Rekursives Abarbeiten und Einfügen von `#include`
  - Entfernen der `\` Zeilenumbrüche
  - Bearbeiten und Auflösen der `#if , #ifdef , #ifndef , #elif , #endif` Anweisungen
  - `#if` kann mit einfachen Ausdrücken umgehen
- Wertvolle und wichtige Funktionalität
  - Erzeugen von umgebungsabhängigem Quellcode (debug mode, rechnerplattformspezifisch, compilerspezifisch, ..
  - Definition von Macros

Details: https://gcc.gnu.org/onlinedocs/cpp/

gcc preprocessor Anweisungen

- Macros werden mit der directive `#define` und `#undef` definiert und vom preprocessor im laufenden Quelltext gesucht und durch den Macrotext ersetzt.
  - Parameter Definition

```
#define TEXT_ARRAY_SIZE 128
int char[TEXT_ARRAY_SIZE];  →  int char[128];
#undef TEXT_ARRAY_SIZE
#define TEXT_ARRAY_SIZE 1024
int char[TEXT_ARRAY_SIZE];  →  int char[1024];
```

Gültigkeit bis zum #undef

  - Funktion Definition

```
#define myMin(x,y) ((x)<(y))?(x):(y))
….
double p=7,q=9;
double min = myMin(q,p);
```

  - Der # Operator mit einen Variablen Namen wird durch einen string mit Inhalt von str( ) ersetzt.

```
#define str(x) #x
….
cout << str(myText);   →  cout << “myText“ ;
```

  - Der ## Operator verbindet 2 Argumente

```
#define verbinde(x,y) a##b
….
verbinde(c,out)<<“Nützlich?“; →  cout << “Nützlich?“;
```

gcc preprocessor Anweisungen

- Bedingtes Einsetzen von Macrotext in den laufenden Quelltext durch die preprocessor Anweisungen `#if , #ifdef , #ifndef , #elif , #endif`

```
#ifndef DEBUG
#define DEBUG 1
#endif
```

Parameter DEBUG wird definiert falls noch nicht vorhanden. DEBUG wird hier true oder false gesetzt.

```
#if DEBUG
#define DPRINT(x) (std::cout << (x) << endl)
#else
#define DPRINT(x)
#endif
```

Definition einer Funktion die das Argument ausgibt falls DEBUG gesetzt ist

Sonst ist die Funktion leer

```
…..
DPRINT(keys[jKeys]); → cout << keys[jKeys] << endl;
```

Im Quelltext erscheint nur mit DEBUG 1 output !

Für x können längere Ausdrücke gesetzt werden

Fortsetzung in der nächsten Zeile

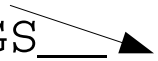
```
#define D(x) do {std::cerr << x ;} \
        while(0); std::cerr << endl

D(key << " " << myVal);  → cout<<key<<" "<<myVal<<endl;
```

gcc preprocessor Anweisungen

- Es gibt intern gesetzte Parameter auf die man zugreifen kann

```
__DATE__     __TIME__     __cplusplus     __FUNCTION__
__LINE__     __FILE__     __VA_ARGS__
```



ISO Standard des compilers

C++ Programme mit vielen Macros sind schwer lesbar!