C++

INTRODUCTIO
N AND
PROFESSIONAL
PROGRAMMING

FRED RAY

C++: Introduction and Professional Programming

FRED RAY

Table of Contents

1 The first program 1

1.1 What is a program ?? 1

1.2 The “Hello World” program in C3

1.3 The “Hello World” program in C++4

1.4 Internal details when programming 5

1.5 designations in the lecture 5

1.6 Newer C++ compilers 6

2 Simple data types 7

2.1 variable 7

2.1.1 inflation variables7

2.1.2 Designation of variables 8th

2.2 constants 9

2.2.1 integer constants 9

2.2.2 floating point constants 9

2.2.3 Character constants (character constants) 9

2.2.4 Character string constants (string constants)9

2.2.5 Symbolic Constants (Macros) 10

2.2.6 Constant with variable names 11

3 operators 13

3.1 assignment operator 13

3.2 Arithmetic Operators 14

3.2.1 un ¨are operators14

3.2.2 I¨are operators 14

3.3 comparison operators 15
3.4 Logical Operators 17
3.5 Bit-Oriented Operators 17
3.5.1 un ¨are bit-oriented e operators 17
3.5.2 Am ¨are bit-oriented e operators 18
3.6 Operations with predefined functions 19
3.6.1 Mathematical Functions 19
3.6.2 functions nfu¨ right character strings 21
3.7 Increment and decrement operators 22
3.7.1 Pr ¨afixed notation 22
3.7.2 postfix notation 22
3.8 Compound assignments 22
3.9 W morenow addition e constants 23

4 control structures 25
4.1 Simple instruction 25
4.2 block 25
4.3 branches 27
4.4 derightZ[illegible] 33
4.5 repellent cycle 38
4.6 Non-shedding cycle 38
4.7 Multiway selection (switch statement) 42

4.8 Incededcontrolbresult 43

5 Structured data types 45

5.1 fields 45

5.1.1 One-dimensional fields 45

5.1.2 Multidimensional Fields 51

5.2 structures 52

5.3 union 56

5.4 record choiceyp 57

5.5 General type definitions 58

6 pointer 59

6.1 agreement of pointers 59

6.2 pointer operators 60

6.3 Pointers and Arrays - Pointer Arithmetic61

6.4 Dynamic arrays using pointer variables62

6.5 pointers to structures 67

6.6 reference 68

7 functions 71

7.1 definition and declaration 71

7.2 parameters bresult73

7.3 Ru ¨ckg valuesbefore n functions 74

7.4 fields as parameters 76

7.5 Declarations and header files, libraries79

7.5.1 Example: printvec 80

7.5.2 Example: students 82
7.5.3 A simple library using student as an example83
7.6 The main program 84
7.7 Recursive Functions 86
7.8 egg nbigbiggers Example: bisection 86

8 The data type class 93
8.1 Class declaration data and methods94
8.2 The constructors 94
8.3 The Destroyer 96
8.4 The assignment operator 96
8.5 The print operator 97
8.6 data encapsulation 99

9 File input and output 103
9.1 Copy files 104
9.2 Data input and output via file105
9.3 Switching input/output 105

10 output formatting 107

11 tips and tricks 109
11.1 Pr aprocessor command 109
11.2 timing in the program 110
11.3 profiling 111
11.4 debugging 111

Chapter 1

The first program

1.1 What is a program ??

Actually, everyone already knows programs, but one often understands different contents by them.

- party manifesto ↔ ideas
- theater program ↔ scheduling
- music score ↔ strict sequence of instructions
- Windows program ↔ interactive action with the computer

programnisttheresLo¨senbeforenGave
upinouchfenm computermediums ownSoftware
and includes all four aspects in the above list.

Aettypicalu¨ exerciseebcomplythe followingnSentence:

Ä differentn[edit]sietheres source file[sourceefile]
accordingly enrightGave upposition,andbsubstitute[compile]

What should I do ??

ideaim head or on paper. (What should the computer do?)

program idea

ideafu¨↓rightcomputer
processing
Formulate idea in↓ a programming language.
source file fu¨right the
computerandtranslation Zen

Draft. (How can the computer realize the idea?)

Edit source text/source file. (What expressions may I use?)

Compile [and link] file. (translation into processor language)

Structogram

program code

exportaudibleprogram

↓
program mcode
exportearRemarks:

Test program with different data sets

program test

1. The learning process in programming typically proceeds from the bottom up in the previous overview.
2. softwaree =exportaudiblesProgram + program code + ideas

<u>warningG:</u>Theresprogrammouchfenm computer will I agreetheresexport**to hear**what is described in the

Tmore typicalOnfomtabe*rightI meant that completely differently.*

<u>note:</u>Computers are stupid! Only the (correct and reliable**liquid) software**eusetthe

1.2 The "Hello World" program in C

idea: The simplest program that only writes a message on the screen.

HelloWorld source code (HelloWorld.c):

```
/* HelloWorld.c */

#include <stdio.h>

main()
{
 printf("Hello World
 \n");

/* "\n" - new line */
```

- Start, end of comment
- predefined functions/Variables/ constants
- Beginning of the main program
- simple instruction
-

block statementsource code inputand

compile, execute programear:

0. Turn on computer, log in
   Log in:
   passwd:

1. Tterminalor file managerOopenand change to
   the working directorysel.
   LINUX> cd progs

2. Enter source text in the source file, editor of your own choice.
   LINUX> editHelloWorld.c or
   LINUX> xemacs HelloWorld.c.

3. Compile source file. LINUX> gcc HelloWorld.c

4. programmexportear.
   LINUX>a.out
   or
   LINUX>./a.out
   or
   WIN98> ./a.exe

Remarks:

- LINUX> gcc HelloWorld.c
  generated oneexportaudiblesProgram with the default name a.out .

- casesthere sexportaudibleeProgram, e.g. myprog should be called:
  LINUX> gcc -o myprog
  HelloWorld.c LINUX>
  myprog

- Tueeconcretee command line to the
  compileistuh¨ngiGbeforemvconsiderdeten
  compiler.

1.3 The “Hello World” program in C++

```
// HelloWorld.cc

#include

<iostream.h>

Main()
{
 cout << "Hello World" <<
 endl;
```

HelloWorld. Idea and structogram as in section 1.2

source code (HelloWorld.cc):

- comment to line-in
- predefined classes and methods
- Beginning of the main program
- simple instruction
- block statement

source code inputU.Ni.ecompile, execute programear:

0./1. as in § 1.2.

2. Edit source file.
   LINUX> edit HelloWorld.cc

3. Compile source file.
   LINUX> g++ HelloWorld.cc

4.
   programmexpo
   rtear.
   LINUX>a.out
   or
   LINUX>./a.out
   orWIN
   98> ./a.exe

Remarks:

- ofrightC source code of HelloWorld.c can also be written in C++ andbreplacedwill:
  LINUX> g++ HelloWorld.c
  However, the source text line #include <stdio.h> is then absolutely necessary.
- C instructionsare a subset of the C++ instructions.
- The C comment/* */ can also be used in C++.
- ofrightc++ how // gotnot to the syntax of C andshould therefore not be used in C programs. Otherwise there is a portability problem, ie not every C compiler can compile the source code.

Programming tip:

Esgivet(nearly)alwaysrightmehrightals
oneMonequalsince one ideaim computer
programto realize.
=⇒ Find your own programming style (and improve it).

1.4 Internal details when programming

derighteasy geachangesec a11fto compile
LINUX> g++ -v HelloWorld.cc
generateda layoungereScreen output showing several stages of compilingrens indicates. Here are some tips on how to look at each phase to better understand the process:

a) Preprocessing:
header files(*.hh and *.h) are added to the source fileGt(+ macrodefinitions, conditional compilation)
LINUX> g++ -E HelloWorld.cc > HelloWorld.ii
The addition > HelloWorld.ii directs the screen output to the HelloWorld.ii file. This HelloWorld.ii file can be viewed with an editor and is a long C++ source code file.

b) ü bsubstitutein assembly code:
Here a source text file in the (processor-specific) programminglanguage assembler generated.
LINUX> g++ -S HelloWorld.cc
The resulting HelloWorld.s file can be viewed with the editor.

c) Generate object code:
Now a file is created which contains the direct control commands, ie numbers, for the processor.

LINUX> g++ -c HelloWorld.cc
The file HelloWorld.o can no longer be viewed in the normal text editor, but with
LINUX> hex HelloWorld.o

d) links:
Vbindall object files and necessary libraries for executionMr-ble program a.out .
LINUX> g++ HelloWorld.o

1.5 designations in the lecture

- Commands in a command line under LINUX:
  LINUX> g++ [-o myprog] file name.cc
  The square brackets [ ] mark optional parts in commands, commandsor definitions. Each file name consists of the free waavailableBase name (file name) and the suffix (.cc) which identifies the file type.

- Some file types after the suffix:SuffixFileType

Suffix	FileType
c	C source file
. *H*	C header file (also C++), source file with predefined program modules
.*cc*[.cpp]	C++ source code file
.*hh*[.hpp]	C++ -Header file
. *O*	object file
.*a*	Library file (Library)

- A statementhow < type> means that this placeholder must be replaced by an expression of the appropriate type.

1.6 Newer C++ compilers

Since the first version of this script, newer versions of the header files are available alongside the old header files, such as iostream instead of iostream.h. In some cases compilers like g++ then deliver annoying, multi-line warnings when compiling the source text on page 4. This error message can be removed using LINUX> g++ -Wno-deprecated HelloWorld.cc oppress¨cctwearth.

_ HelloWorld atinright(recommended) use of the new header files achanget himselfourright smallprogram in :

```
//        Include file "iostream" is used insteadof

#include

Main
()
{                     scope operator::is

  std::cout << "Hello World" <<
}
```

§
I wantinnscopee operator, please refer8 do not have to write down every timeeat,henceI prefer the variant:

```
//        Include file "iostream" is used insteadof

#include
//        All methods fromclassstdcan be
using namespace std;

int main()
{
 cout << "Hello World" <<
 endl;
```

Chapter 2

Simple data types

2.1 variable

2.1.1 variable definition

Every meaningful program processes data in some form. This data is saved in variables.

Tueevariable i)is a symbolic
representationsentation(identifier/name)
fu¨righte n nstorage space of data.

ii) is described by type and storage class.

iii) The contents of the variables, ie the data in the memory space, change during program execution.

General form of variable declaration:

[< storage class >] <type> <identifier1> [, identifier2] ;

TypeMemory
Content val
uesto bytes (g++)

\ char	1	Character sign	'H', 'e', 'n'
boolean	1	boolean variable	false, true [C++ only]

–
internal 4-32767.2 ^{31}
short [int]2whole Numbers-32767
–
long [int]4-32767.2 ^{31}

float 4 Floating point numbers 1.1, -1.56e-32
double81.1, -1.56e-32, 5.68e+287

–
unsigned[int] 4 of coursecommone
numbers 32767,32769,23$_{1}$ 1
—–
long long [int]8whole Numbers2 ^{31},
2^{63} 1
longdouble 12 Floating point numbers 5.68e+287, 5.68e+420

Remarks:

- Character data stores exactly one ASCII or special character.

DataTypes.

- The memory requirement of types of the integer group (int) can depend on the compiler and the operating system (16/32 bit). It is therefore advisable to read the relevant compiler instructions or to determine the actual number of bytes required with the sizeof operator using sizeof(<type>) or sizeof(<variable>). See also the following example:

```
/* Demo for sizeof operator */

#include <iostream.h>

Main()
{
 int i;

 cout << " Size (int) = " <<
 sizeof(int) << endl; cout << " Size
 ( i ) = " << sizeof(i ) << endl;
}
```

- Wirightare usually the base type int fu¨rightthe appropriate subrange of integers and unsigned int fu¨rightn / Aof thecommone numbersvuse.theMarking unsigned can also be linkedcan be used with other integer types.

2.1.2 Designation of variables

Guval	awkwa	grou
id		
i		
j		
ra	3	3 is not a letter
re	_ i	* is operator
fi	3*	character
e	a	- is operator

Ex210.c § Variable names start with letters or

underscores, the following characters can also be numbers. The use of spaces and operator characters (3) in names is not permitted, nor are variable names allowed to be keywords of the C++ syntax (see Ref.).

- ___ C/C++ is case sensitive, ie, ToteHosen and toteHosen are different identifiers!
- According to the original C standard, the first 8 characters of a variable identifier are significant, ie a2345678A and a2345678B would no longer be perceived as different identifiers. Compilers now see more characters as significant (C9X standard: 63 characters).

2.2 constants

Most programs, including HelloWorld.cc, use unchangeable values, so-called constants, during the course of the program.

2.2.1 integer constants

Decimal constants (base10): 100 // int; 100
512L// long; 512
128053 // long; 128053
octal constants(Base 8): 020// int; 16
01000L // long; 512
0177// int; 127
Hexadecimal constants (base16): 0x15 // int; 21
0x200// int; 512
0x1ffffl // long; 131071

2.2.2 floating point constants

Floating point constants are always interpreted as double. Some examples infollowing: 17631.0e-78
1E+10 // 1000000000
0
1. // 1
.78 // 0.78
0.78
-.2e-3 // -0.0002
-3.25

2.2.3 character constants(character constants)

The character constant contains the character between the two ' :

```
'a', 'A', '@', '1'//      ASCII
character
'' //                 spaces
'_'//underline/underscore
'\''                      //      prime sign '
'\\'//                            backslash character \
'\n'//                          new line
'\0'//                          Null character NUL
```

2.2.4 string constants(string constants)

Thestringincludesthecharactersbetweenthetwo
" :

```
"Hello World\n"
""//                             empty string
"A"//String             "A"
```

```
/* demo for char / string -constant
                                          */
#include <iostream.h>

Main()
{
 cout << "A" <<"string     : " <<
 sizeof("A") << endl; cout << 'A
```

Ex224.c Each character string is automatically terminated with the (character) character '\0'(*"Hey,hererightH**o**inright thongouchf!"*).thererightist 'A'unequalH "A",wmooseconsists of the characters 'A' and '\0' and thus 2 bytes for storageb One required.

2.2.5 Symbolic Constants (Macros)

If one of the constants used in the previous sections is required more than once, a symbolic name is assigned to this constant, e.g

#define NL '\n'
#defineN5
Ex224.c #defineHELLO"Hello World\n"
or in general

#define<identifier> <constant>

Remarks:

- ofrightPreprocessrightreplaces every occurrence of in the rest of the source code
<identifier> with <constant>, ie, off
cout << HELLO;

will

cout << "Hello World\n";

•

_ ü usuallyironno lower-case letters are used in these identifiers because, for example, MAX AUTO is then immediately recognizable as a symbolic constant.

2.2.6 Constant with variable names

Ex226.c wei.ea variable declaration with the ending¨sselwortconst geis marked, this variable can only be initialized in the declaration part and never again afterwards, ie it acts as a constant.

```
// Constants and
variables


Main()
{
 constinternal      // The only initialization of
      internal    i, j  =5;// First
           initialization of variables

 cout << "Hello

 World\n"; i = j + N;

 cout << endl << i << " << j << " " << N <<
```

difference:

#define N 5 Eswill no disk space fu¨rightN benorequired,since N in ge
all source code is replaced by 5.

const int N =5; Variable N is saved, the program works with it
her.

Chapter 3

Expressione,Operators and mathematical functions

- expression¨ckeconsist of operands and operators.
- arevariables, constantsorrightexpression again¨cke.
- operators fuearnActions with operands.

3.1 assignment operator

The assignment operator <operand_A> = <operand_B> assigns the value of the right operand to the left operand, which must be a variable.

For example, in the result of the statement sequence

```
{
 int x,y;

 x = 0;
 y = x +
 4;
```

the value of x is 0 and the value of y is 4. where x, y, 0,x+4 operands, where the latter is also an expression consisting of the operands x, 4 and the operator +. Both x = 0 and y = x + 4 are expressions. First the trailing semicolon ; converts these expressions into statements to be executed!

Ex310.c Es kann also multiple assignments occur. The following three assignments are aequivalent.

```
int a,b,c;

a=b=c=123;          // 1st

a = (b = (c =       // 2nd

c =                 // 3rd option (default)
123;
b =
}
```

3.2 arithmeticoperators

3.2.1 U.Na

atwellren operatorskickstonly one operand.

operator	Description	Example
-	negation	-a

3.2.2 Iareoperators

atama operatorstwo operands occur. The result type of the operation ha¨ngtbeforenenn operatorsaway.

operator	description	example
+	addition	b + a
-	subtraction	b - a
*	multiplication	b * a
/	Division (! with integer values !)	b/a
%	remainder in integer division	b % a

```
{
 internal i,j ;
 float ij_mod, ij_div,
 float_ij_div;

 i = 8;

 ij_div = i /                    // Attention: result
 j;                              is 2


                                 // now: result is
 float_ij_div =                  2.666666
 i/(float)j;                     // explicit or
}
```

Ex320.c Dividing integers calculates the integer part of the division, ie 8 / 3 returns 2 as the result. However, if the result is 2.666666, at least one of the operators must be converted to a floating-point number, as can be seen in the example.

```
{
 int k;
 double x =
 2.1;
                    // k stores

 k =                // k stores 0, Integer

 k =                // k stores 3,

 k = -              // k stores -3 or -4, compiler

 k =                //

 x =                // x stores

 x =                // x stores

 x = 1 +            // x stores

 x = 0.5 +          // x stores
 1/2;
```

Ex320.c Concerning.inrightpriority rulenwas¨right operators may beouchfyoue literaturevprovedyoueoldThe rule “point calculations before dash calculations” also applies in C/C++. Analogueto thebecome school expr¨ckein round brackets (<expression>) firstcalculated.

3.3 comparison operators

Vcomparison operatorsare bina¨reoperators. The result value is always oneInteger value, where FALSE returns 0 and TRUE returns non-zero.

operator	description	example
>	bigsweeter	b > a
>=	bigsweetrightorrightsame	b >= 3.14
<	smaller	a <b/3 Ex330.cc
<=	Smaller or equal	b*a <= c
==	equal (! with floating point numbers!)	a == b
!=	not equal (! with floating point numbers!)	a != 3.14

```
boolbi,bj;
internal i;

bi = ( 3 <=
4 ); bj = (
3>4        );
                  TRUE=        " << bi
c out << " 3      << endl;

//           if - statement will be
i = 3;
if (i
<= 4)
   cout << "\ni less or equal 4
  }
}
```

A typical error occurs when testing for equality by using instead of theequalsoperators == the assignment operator = is written. The compiler accepts both source texts, possibly (depending on the compiler) a warning is issued if the code is incorrect.

```
{
//            Incorrect
 int

 i = 2;
 if ( i =                  // Assignment i=3 is always
 3 )
    cout << " BB: i=" <<i<<endl;//              i
    is 3
    i = 0;
   }
}
```

in theincorrectn codekickstthe unknownwisheNebeffecton,thereßenrightWero f the variable i in the test geat will, wowearì.e following,correct code nonehas side effects.

```
{
//            Correct
 int

 i = 2;
 if ( i == 3 )                       // Correct
   {
     cout << " BB: i=" << i          // i always
     << endl; i = 0;                 remains 2
   }
 cout << " CC: i = " << i <<
}
```

3.4 Logical Operators

Esgivetnowright aandn / Arenlogicaln Operator:

operator description		example		
!	logical negation	! (3>4)	//	TRUE
U.Ni.etwo bina¨relogicale Operators:				
operator	description	example		
&&	logicalAND	(3>4)&&	(3<=4)	// FALSE
||	logical OR	(3>4) ||	(3<=4)	// TRUE

Ex340.c Tueetruth tables fu¨rightthe logical AND and the logical OR areknown from algebra (otherwise, see literature).

```
{
 const int Ne                 // one
      = 5; int

 cout << " i =
 " ;                          // input

 if ( i <=           )        // other limit
 No&& i >= 0
    cout << "i between 0 and 5"
  }
}
```

3.5 bit-orientedoperators

eggn bitis the smallest information unit with exactly two mosamen conditionfind:

(

wouldticëscht bit set

(0

≡

(0
≡

≡
false true

*L*1A byte consists of 8 bits, so a short int number is 16 bits long. WhenOperators in bit operations usually occur in integer expressions¨ckeon.

3.5.1 U.Näbit-oriented operators

operator	Description	Example
~	binaovercomplement,bitwironnegationnof the operand	~k

3.5.2 Amäbit-oriented operators

operator	Description	Example
&	bitwise AND theoperands	k & l
|	bitwiseOR	k | l
^	bitwise exclusiveOR	k^l
<<	Left shift bits from <op1> by <op2> digits	
>>	Right shift of bits from <op1> by <op2> digits	

k<<2// = k*4

k >>2// = k/4

Truth table:x	and	x &	and	x |	and	x^y
0	0	0		0		0
0	L	0		L		L
L	0	0		L		L
L	L	L		L		0

```
// bitwise operators

#include

<iostream.h>

main()
{




 n1 =~k;         // complement           L.. LLL00L = -7 = -6
 n2 = k &l;    // bit                    - 1
 AND n3 = k | l;    //                   0..000L00 = 4
 bit OR n4 = k ^l;                       0..000LLL = 7
                 // bit                  0..0000LL = 3
 XOR                                     0..0LL000 = 24 = 6 *
 n5 = k<<2; // shift left by 2           2^2
}
```

These operators are demonstrated in the following examples:

```
l = 5;      //                    0..000L0L = 5
k = 6;      //                    0..000LL0 = 6
```

Ex350.c

TueeBit operations are nuplusHwhen testing whether an even or odd in-terger number is available. The least significant bit can be used with integer numbers to differentiate between even and odd numbers (see also the bit representation of the numbers 5 and 6 in the above code). Therefore, if this bit is ORed with a set bit, the least significant bit remains unchanged for odd numbers. This is exploited in the following code.

```
// mask for odd
numbers


Main()
{
   const   mask =                          0..0000
      int   1;

 count
 <<"Number:                    //read  number

 count <<"" << i << " is a ";

// Check for odd number:
//Load bit remains unchanged for odd
numbers

 if ((i | mask) == i)
  {
    cout << "odd";
  }
 else
  {
    cout << "even";
  }
 cout << "number." << endl << endl;
```

Ex351.c

3.6 Operations with predefinedfunctions

3.6.1 Mathematicalfunctions

The header file math.h contains, among other things, the definitions of the mathematical functions and constants summarized in Table 3.1:

Rounding a real number x can be achieved with ceil(x+0.5) (ignoring the rounding rules in eg, 4.5).

Function/Constant	Description
square(x)	square rootbeforenx: $\sqrt{2}$ x (x ≥ 0)
exp(x)	*ex*
log(x)	of courselocalright logarithmbeforenx:loge x (x > 0)
pow(x,y)	Exponentiation (x > 0 if y is not an integer)
fabs(x)	absolute value of x: | x |
fmod(x,y)	real remainder of x/y(y = 0)
cell(x)	n / Anexteall e number ≥ *x*
floor(x)	n / Anexteall e number ≤ *x*
sin(x), cos(x), tan(x)	trigonometric functions
asin(x),acos(x)	trig inverse functions (*x* ∈ [1, 1])
assigning(x)	trig inverse function
M_E	Euler's number e
M_PI	*p*

Table 3.1: Mathematical functions

Ex361.c foot¨rightyoue Permitted:sinceThe programmer is responsible for the operations, ie the domain of the arguments. Otherwise program aborťwhichor produce nonsensical results.

```
// Math.functions
#include <iostream.h>
#include <math.h>

Main()
{
 double x,y,z;

 x =-1;//x                    < 0 !!
 and =square(x);    //     Square root with wrong

 argument cout << "x = " << x << ", y= " << and <<

 endl;

                          //Absolutely value
 z = fabs(x)

                          //     Power function
 y = 3.0;                 //try   2.0 , 3.0 and
 z =
 cout << "(x,y) = "<< x <<", "<< y
                            << " , x^y = " << of
}
```

The functions from math.h are stored in a special mathematical library, so the compiling and linking command must take this library libm.a into account, ie
LINUX> g++ Ex361.cc [-lm]

3.6.2 functionsnfu¨rightcharacter strings

Ex362.c The header file string.h contains, among other things, the definitions of the following functions for strings:

Function	Description
strcat(s1,s2)	Attachmentlengthnbeforen s2an s1
strcmp(s1,s2)	Lexicographical comparison of strings s1 and s2
strcpy(s1,s2)	Copies s2 to s1
strlen(s)	Number of characters in string s (= sizeof(s1)-1)
strchr(s,c)	Finds character c in string s

tabell3.2: Classic functions fu¨rightthongs

```
// String
functions
#include                        //
<iostream.h>
#include
//Definition       and initialization of string
variables
//--> sec. 5.1

 char s[30], s1[30] = "Hello", s2[] =
 "World";internal        i;

 cout << "s1 = " << s1 <<

 i =                              // lex.

 cout << "cmp : " << i << endl;

 strcpy(s,s1);/                  / copy s1 on

 s cout <<"p                     : " << s

 strcat(s,s                      // Appends s2

 count <<"p       : " << s << endl;

 i =strlen(s);// length of string s cout

 << "Length of s : " << i << endl;
}
```

detailsandbhethese functions (and others) ko¨nenby means of
LINUX> man 3
string LINUX> man

strcmp can be
obtained.

3.7 Increment and decrement operators

3.7.1 P[illegible]

```
++<lval                    // <lvalue> = <lvalue>
ue>                        + 1
```

```
//      Example: prefix
{
 int i=3, j;

 ++i;//          i =

 j =             // i = 5, j = 5
                 // above prefix notation is
 i = i
 + 1;
}
```

3.7.2 postfix notation

```
<lvalue                    // <lvalue> = <lvalue>
>++                        + 1
```

```
//      Example: postfix
{
 int i=3, j;

 i++;//          i =

 j =             // i = 5, j = 4
                 // above postfix notation is
 j = i;
 i = i
}
```

Pre¨-U.Ni.epostfixed notation should economicalvusedwearth,mostlytbusedone thisefu¨rightan index variable in cycles (§ 4).

3.8 Compound assignments

value assignmentsthe form

<lvalue> = <lvalue> <operator> <expression>

knockout¨nen to

verkurtwill.

<lvalue> <operator>= <expression>

Here <operator> ∈ {+,-,*,/,%,&,|,^,<<,>>} from § 3.2 and § 3.5 .

```
{
 inti,j,w;
 float
 x,y;
                 i = i+j
 i += j          w = in >> 1 (= w/2)
 in >>= 1;       x = x*y
 x *=y;
```

3.9 Continueenow¨additionalconstants

foot¨right system dependentpole number ranges,Exactlyopportunitiesetc. istyoueselectionlenrightfol-ing constants quite helpful.

FunctionDescription
FLT_DIG numberlgoodvalidrightdeci
mals fu¨rightfloatFLT_MIN Smallest
representablepositive number
FLT_MAX
bigate,representableepositive
numberFLT_EPSILON Smallest positive
number with 1.0 + ε = 1.0

(Job advertisementfake)
DBL_ howifinfu¨rightdouble
LDBL_ howifinfu¨rightlong doubles

Table 3.3: A few constants from float.h

FunctionDescription
INT_MIN Smallest representable
integer number
INT_MAX bigate,representableep
ositive integer SHRT_
howifinfu¨rightshort int

LONG_ howi finfu¨rightlon
g int
LLONG_ howi finfu¨righ
tlong long int

Table 3.4: A few constants from limits.h

Wmoreconstants knockout¨nenunder the gabusynLinux distributions di-directly in the files /usr/lib/gcc-lib/i686-pc-linux-gnu/3.2.3/include/float.h and */usr/include/limits.h*to be checked. The corresponding header files can also be created with the command

```
LINUX> find /usr -name float.h -print
```

be searched.

Chapter 4

control structures

4.1 Simple instruction

A simple statement is made up of an expression and the semicolon at the end of a statement:

<expression> ;

Examples: cout << "Hello World" << endl; i = 1 ;

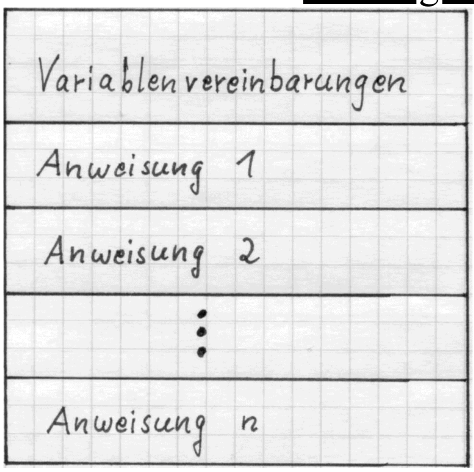
4.2 block

The block (also compound statement) is a summary of agreements and statements using curly brackets:

```
{
<statement_1>
...
<statement_n>
```

```
//            Example block
{                       // beginning of
block
  internalin;              // Agreement
                           //
  i  =  O;                 Instruction
}                          //
```
}

Structogram:



- In C, the declaration part must immediately follow the beginning of the block. In C++ you canseveraleVparts of the agreementim block exist, shehimeatnnowrightbefore enrightrespective first use of the variable names. From gru¨nden derightHowever, this should not be exploited for the sake of clarity in the program.

- The closing bracket at the end of the block “}” is not followed by a semicolon.

- A block can always be used in place of a statement.

- Blöckekönnen any into each other right be nested.

Ex420.c • The variables declared in a block are only visible there, ie the variable does not exist outside the block (locality). Conversely, variables of the superordinate block can be accessed.

```
// blocks
#include
<iostream.h>
main()
{                                  // outer

 i = j
 = 1;                              // Begin inner
 {                                 block
  int

  i = k = 3;
  count <<"inside     i = "
  << i << endl; cout
  << "i_outer j = " <<
  j << endl;
 }// End inner block

j =                                // k
i+k;
```

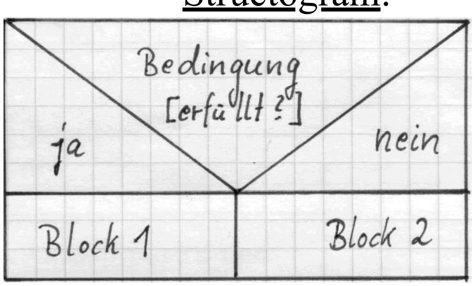
4.3 branches

The general form of branching (also alternative) is

if (<logical expression>)
<statement
_A> else
<statement_B>

U.Ni.e e.gdtin turns in turnalsapplice. deright else -Branchcannwignoredbecome (simple alternative).

Structogram:



As is so often the case, a concrete problem can be programmed in various ways.

(

Example: We consider the calculation of the Heaviside function

$$y(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Ex431.c and present four variants of implementation.

```
//                    supporting
#include <iostream.h>
Main()
{
 doublex,y;

 cout << endl << " Inputargument  :
 "; cin>>                            x;
//Version                       a

//                              version

//                              version

//                              version

}
```

<u>Option A</u>: simple alternative

```
//              version
{
 y =
 if ( x >=
 0. )                    // exactly one statement in the

 cout << " Result of version a) : " << y
}
```

<u>variant b</u>: dual alternative

```
//              version
{
 if ( x >= 0.0 )
     y = 1.0 ;
 else
     y = 0.0 ;

 cout << " Result of version b) : " << y
}
```

variante c:double alternative with Blo¨ckin

```
//              version
{
 if ( x >= 0.0 )
   {
     y = 1.0 ;
   }
 else
   {
     y = 0.0 ;
   }

 cout << " Result of version c) : " << y
 << endl;
```

variant d: decision operator.

Stepping in a double alternativein each branch only one value assignment to the same variable (as in versions b) and c)), then the decision operator

<log. expression> ? <expression A> : <expression B>

```
//              version
{
 y = (x > = 0) ? 1.0 : 0.0 ;

 cout << " Result of version d) : " << y
 << endl;
```

be used.

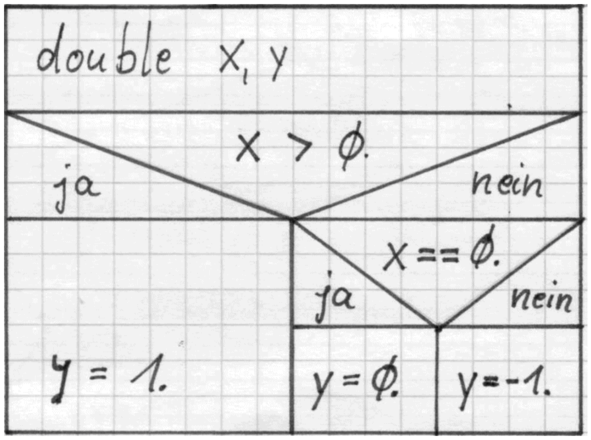
Example: Another example is the calculation of the signum function (sign function)

y(x) =

1 $x > 0$

0 $x = 0$

-1x < 0



Ex432.c and we present several variants of implementation.

Structogram:

We consider two implementation variants, the framework program is identical to the framework program on page 28.

```
//                    version
{
 if ( x > O.O )
   {
    y = 1.0 ;
   }
 else
   {
    if ( x == O.O )
      {
       y = 0.0 ;
      }
    else
      {
       y = -1.0 ;
      }
   }
 cout << " Result of version a) : " << y
}
```

Option A: nesting of alternatives

varianteb:casesenright else branchnowrightouchs onewfester if-else statementbe-stands, variant a can be slightly modified.

```
//              version
{
 if ( x > 0.0 )
   {
    y = 1.0 ;
   }
 else if ( x == 0.0 )
   {
    y = 0.0 ;
   }
 else
   {
    y = -1.0 ;
   }

 cout << " Result of version b) : " << y
}
```

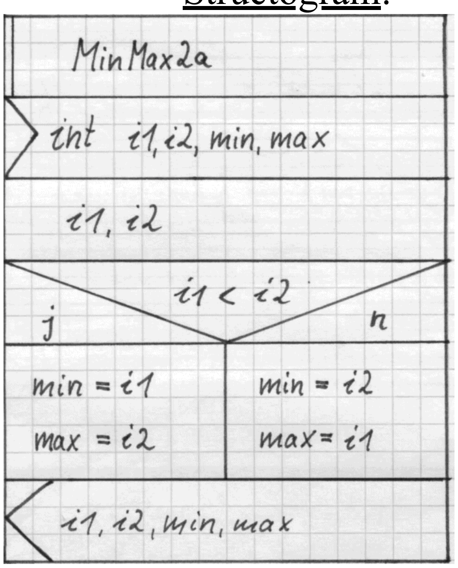
In general, such a multipath decision can be

if (<logical expression_1>)
<statement_1>
else if (<logical expression_2>)
<instruction_2>
...
else if (<logical expression_(n-1)>)
<statement_(
n-1)> else
<statement_n>

written, with the else branch being optional.

Ex433.c Example: Determining the minimum and maximum of two numbers to be entered.

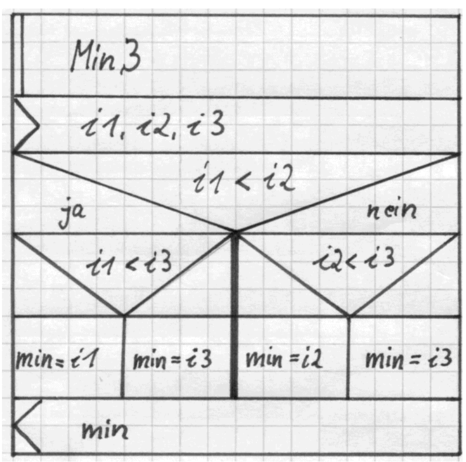
<u>Structogram</u>:



```
// Example: Maximum and Minimum of two

numbers #include <iostream.h>

Main()
{
 int i1,i2,min,max;

 cout << endl << " Inputarguments
      i1 i2 : ";cin>> i1 >> i2 ;

 if ( i1 < i2 )
   {
     min = i1
     ; max =
     i2 ;
   }
 else
   {
     min = i2
     ; max =
     i1 ;
   }

 count << "Min,Max(a)          : " << min << ",
```

Ex434.c Example: Determining the minimum of three numbers to be entered.

Structogram:



```
// Example: Minimum of three
numbers

#include

<iostream.h>


 cout << endl << " Input            i1 i2     :
 Arguments

 if ( i1  < i2 )
   {
     if ( i1  < i3 )
       {
         min = i1;
       }
     else
       {
         min = i3;
       }
   }
 else
   {
     if ( i2  < i3 )
       {
         min = i2;
       }
     else
       {
         min = i3;
       }
   }
 count << "minutes(a)       :
 "<<minutes<< endl;
```

4.4 derightZoijs(for loop)

At theZacooling cyclesstands for the number of cyclesufea priori, thefracture test is done before running a cycle. The general form is

for (<expression_1>; <expression_2>; <expression_3>)
  <statement>

Ambest be the zacooling cyclesan one example allowedfeeds.

Ex440.c Example:Esistthe sum of the first 5 natulocaln numbers tocalculate.

```
// Example : sum of natural
numbers

Main()
{
 int                             // loop index, sum, last

 n =                             // initialize last

 isum = 0;                       // initialize sum
 for ( i = 1, i <= n,
 i=i+1)
   {
     isum = isum + i;
 cout << endl << "Sum of first " << n
      << " natural numbers = " << isum
}
```

in theabove program example, i is the running variable of Zaoil cycle,wmooseinitialized with i = 1 (<expression_1>), continued with i = i+1 (<expression_3>). does noti.ein i<= n (<expression_2>) regarding the upper limit of the loop capacity¨ufeIs tested. Inside the loop sum = sum + i; (instruction) theactual calculation steps of the cycle. The summation variable sum must be initialized before entering the cycle.

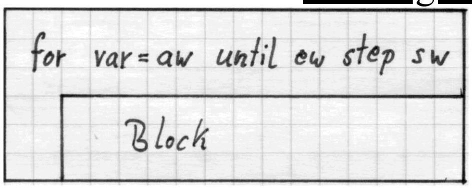
A compact version of this summation loop (correct but very difficult to read) would be:
for (isum = 0, i = 1; i <= n; isum += i, i++)
A distinction is made between the end of an instruction ";" and the separator "," in a list of expressions. These lists are processed from left to right.

§
The <expression_2> is always a logical expression (3.3-3.4) and <expression_3>
is an arithmetic expression for manipulating the run variables, e.g

i++
j = d-
2 d
+= 2
x =x+h// float type
k =2*k// doubling

l =l/4// Quartering - be careful with integers

Structogram:



The control variable can be a simple variableout 2.1 be eg, int or

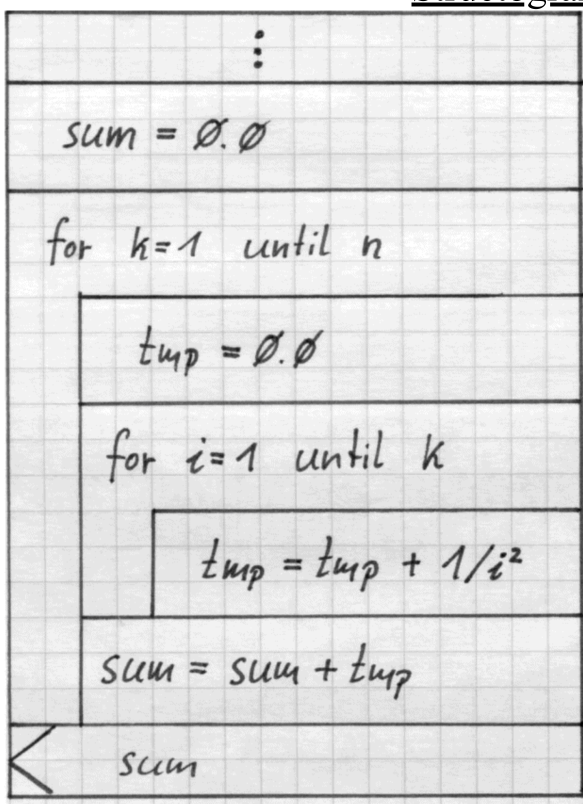
•

double .

Loop

• Be careful when using floating point numbers (float, double) asvariable. The correct abort test may not be easy to implement there due to the internal number representation.

Structogram:



Example: It is the double sum

S S
S
S

nkn

sum= 1

i^2

`
x ˛ ¸

k=1 i=1

t_k

$$= \quad tk$$
$$k=1$$

Ex443.c

Ex442.c

was¨rightn to be entered to be calculatedto.

```
// Example: double
sum
#include
<iostream.h>
main()                                      // loop index, sum, last
{                                           index

 count <<"Inputs n : ";cin>>        // read

 sum_k =                            // initialize outer

 for ( k = 1; k <= n; k++)
  {
    sum_i = 0.0;                    // initialize inner sum
    for ( i = 1, in <= k,           // last index depends
    i++)
     {
       sum_i = sum_i +
    count <<"      Sum(" << k << ") = " << sum_i <<
    endl;
    sum_k = sum_k +sum_i;// sum_k grows unbounded
  }
 count <<"Double Sum   (" << n << ") = " << sum_k
```

Wmoresimple examples calculate the sum of the first even natulocalnNumbers and the Za¨hlen onecountdowns.

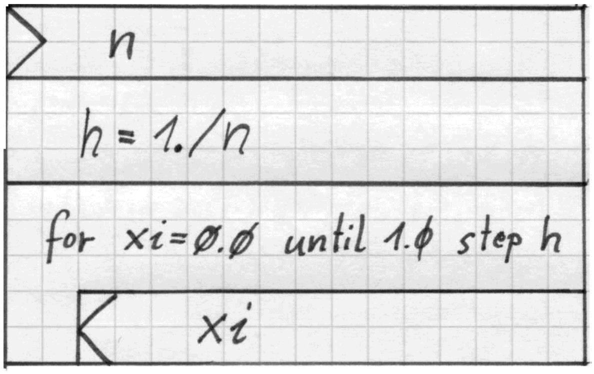
The following examples illustrate the problem of the limited accuracy of floating-point numbers in connection with cycles and some tips on how to work around them.

Loop

Example: Output of the discrete nodes xi of the interval [0, 1], which is in nequal subintervals, i.e.,

$$x_i = iH \quad i = 0, \dots, n \qquad \text{With } H = \frac{1-0}{n}$$

Structogram:




```
Main()
{
 float
 far,car,xi,h;i
 nternal;

 cin>> n;// # subintervals
 xa =0.0e0;// # startinterval
 car =1.0e0;// # endinterval
 h= (xe-xa)/n;// length subinterval


 for (xi = far; xi <= car; xi += h)
   {
    cout << xi << endl;
   }
```

Dafloating point numbernnowright onelimitedenumberlgoodvalidright digitsbsit,canit (mostly) happens that the last node xn is not output. Only for = 2k , k cannin our example a correct processing of the Zacooling cyclesguaranteetwearth. selectionee are

1. ∈ ÄchangeGinsabort testsinxi <= xe + h/2.0 , but xn is still in error.

```
for (xi = far; xi < = car + h/2.0; xi += h)
    {
   cout << xi << endl;
    }
```

2. Cycle with int control variable

```
for (i = 0; i <= n;
i++)
  {
   xi = xa + i*h;
   cout << xi << endl;
  }
```

!

Tueecommon summation of minor and majorouterrightNumbers can also

inaccuracysidesfuear. In the example, the sum

$s1$:=

n

/ $n._{i=1}$

1/i

2with the

S(theoretically identical) sum

compare

2:= S1 1

$i=$

[2]for big

(65,000, 650,000)

row.cc

.

```
#include
<iostream.h>
#include
<math.h>
#include
<float.h>

Main()
{
 float
 s1,s2;
 int i,n ;

 cout << "The first sum will be ratherprecise until = "

     <<ceil(sqrt(1./FLT_EPSILON)) << endl;
 cin >> n;

 s1 = 0.0;
 for (i=1; i<=n; i++)
   {
     s1 += 1.0/i/i;
   }
 cout << s1 << endl;

 s2 = 0.0;
 for (i=n, i>=1, i--))
   {
     s2 += 1.0/i/i;
//s2 +=1.0/(i*i);results in info
//since i*i is longer than int supports
```

The numerical result in s2 is more accurate because all small numbers are there firstare added, which at s1 because of the restrictionsgoodnnumberlgoodvalidrightDigits no longer contribute to the summation kocan. At the same time is closedbnote,that the computation of 1.0/(i*i) ends in an overflow, since i*i can no longer be represented in int numbers. On the other hand, the calculation of1.0/i/i complete¨ndiGimRange of floating point numbers.

4.5 Repelling cycle (while loop)

At therejecting cycle is the number of passagesufenot fixed a priorithe abort test is performed before running a cycle.

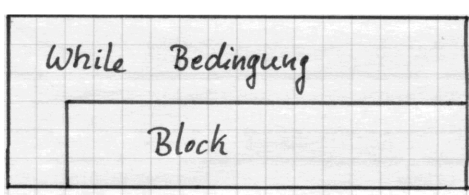
The general form is

while (<logical expression>)
<statement>

```
//          Example : Binary log. of a
#include <iostream.h>
Main()
{
 double
 x,xsave;
 intcnt;

 cout << endl << "Input
 x:" ;cin>> x;

 cnt = 0;                              // Initialize
 while ( x >
 1.0 )
     x= x/2.0 ;
     cnt = cnt + 1;
   }
 cout << endl << "Binary log. of " << xsave
                              << " = " << cnt <
}
```

Ex450.c Example: determineinnrounded upn buildinglogarithms(Bases2) one a-reading number.

Structogram:



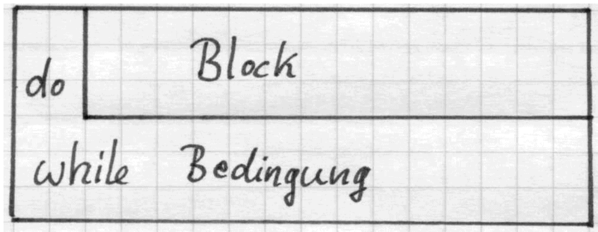
Comment:If the very first test in the rejecting cycle is FALSE, then the statement block inside the cycle is never executed (the cycle is rejected).

4.6 Non-rejecting cycle (do-while loop)

At thenon-rejecting cycle is the number of passagesufenot fixed a-priori, the termination test takes place after a cycle has been run through. Thus passairthe non-rejecting cycle uses the instructions inside the cycle at least once.

The general form is

of the
<statement>
while (<logical expression>) ;



Structogram:

Ex460.c Example: A character is read from the keyboard until an x
is entered.

```
// Example : Input of a character until 'x'
#include
<iostream.h>
main()
{
 char ch;

 of the
   {
     cout << endl << "Input command (x =
     exit,...)
          ";cin

 cout << endl << "         Exit
      << endl <<
      endl;
```

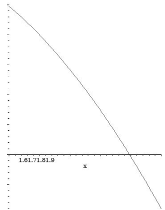
Consider weathera somewhat more demanding example, namely the Losolutioncan be determined from sin(x) = x/2 with x ∈ (0, π). For this one considers

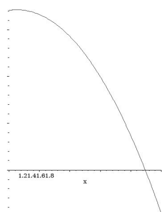
youeEquivalent zero problem: Determine the zero x0 ∈ (0, π) of the

–
Function f(x) := sin(x)*x*/2 = 0 .

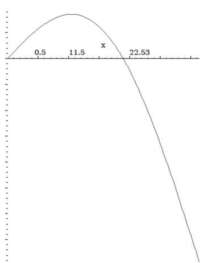
Analytically:No practical Losolutionegpresent.

graphicsch:The function f(x) is graphed and the LoResolution interval reduced manually (halved). This process is continued untiluntil x0 can be determined accurately enough, ie, to a predetermined number of digits.





f(x) =sin(x)-x/20.25



0.2

0

–0.2

0.3

0.2

0.2

0.15

–0.4

0.1

–0.6

–0.8

0.1

0.05

-1

0 2

–1.2 0 2

–1.4

–0.05

–1.6

Numeric: The above graphical procedure can be applied to a purely numerical

rics VExperienced im computer abendure we arth (enright *MAPLE* -

Ex462.m callffsolve(sin(x)=x/2,x=0.1..3

returns as Naapproximation result $x_0 = 1.895494267$). We are developing a program to determine the Zero of f (x) := sin(x) x/2 in the interval [a, b] by bisecting the interval, where it is assumed for simplification that f (a) > 0 and f (b) < 0. The midpoint of the interval is denoted by c := (a + b)/2. then

knockout¨nenWe ubhethe Lo¨sunGstate the following:

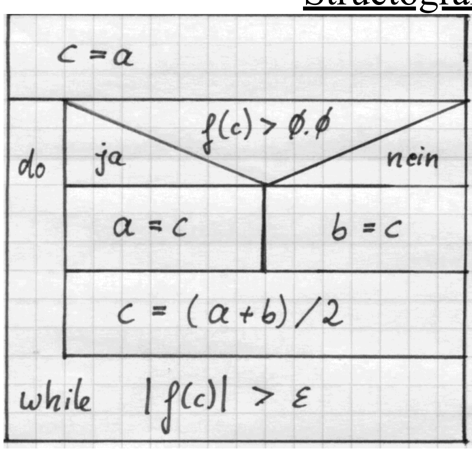
$x_0 \in [c, b]$ if f (c) > 0 .

$x_0 := c \quad \text{cases} \; f(c) = 0$

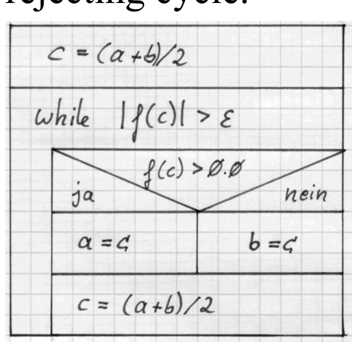
$x_0 \in [a, c]$ if $f(c) < 0$

Ex462.c By redefining the interval limits a and b, the zero search can be reduced to the smaller (halved) interval. We demonstrate the implementation using a non-repellent cycle.

Structogram:



The above bisection can also be realized by means of a rejecting cycle.



```
//zero calculation by bisection in [a,b]
#include
<iostream.h>
#include <math.h>

Main()
{
 const double Eps =
 1e-6; double
 a,b,c,fc;

//        Check that f(a) >0,f(b) < 0
 --
 .              Do-While loop
//    =              // since f(a) >
  fc = sin(c) - c/2;
  of the
   {
    if ( fc > 0.0 )
     {
       a = c;
     }
    else
     {
       b = c;
     }
    c=
    (a+b)/2.0; fc
    = sin(c) -
    c/2;
   }
  while ( fabs(fc) > eps);
//while ( fabs(fc) !=0.0);// endless!! Why?
```

Since floating-point numbers only work with limited accuracy, an abort test f (c) = 0 usually results in an endless program. That's a break test like |f(c)| < ε with a given accuracy 0 < ε 1 is preferable.

§ Likent:Behindcooling cyclesn(for) which executes at least one cycleto hearknockoutnen as well asbyHdifferentiron end(while)to thesalso by non-rejecting cycles (do while)quitexpressed¨cctwearth.thiseA¨equivalentE. g.cannbeggVuseinrightappliToningin4.8 are lost. If in a zacooling cyclesinrightAway-

Loops.c brokent alwaysFALSEE resultsie.inrightloopcoïrphe willNoeexecutedtouches,then istinright appropriatedifferentiron end cycleafterH howbeforerightaequivalent. Howeveristthe not deviron endcycle no longer aequivalent,there the loop köŕpheis also processed once in this case. See the example file Loops.cc.

4.7 Multiway selection (switch statement)

Tueeadditionalegselection ermpdt oneindividualsreactnouchf specificWgeta variable.

```
switch (<expression>)
   {
     case <const_expression_1>:
               <statement_
  1> [break;]
     ...
     case <const_expression_n> :
               <statement_
  n> [break;]
     default:
          <statement_default>
   }
```

```
// Demonstration of Switch statement (break
!!)
#include
<iostream.h>
main()
{
 int

 number;


 switch(numbe
 r)
  {
    case 1:          One = << number <<
        cout
        << "
        break;       Two= "<< number << endl;
    case 2:
        cout                    // Comment this
        << "
        break;       line Three ="<<
    case 3:
        cout         number << endl;
        << "
             << "not in interval" <<
             endl;
        break;// not necessary
  }
}
```

Ex470.c Example:outputeinrightnumber¨rtrightwas¨rightthe integer inputs {1, 2, 3}.

Above switch statementkcdtalsoHwedt onemultiple vbranching(Sincee31)be implemented, however, in the switch statement, the individual branchesexplicit andbheleave the break; statement. Without break; will togetherplusHenrightgo to the next branch¨rige blockprocessedruns.

4.8 Unconditional Tax Instructions rungsudownhille

breakThe na is immediately aborted¨chstaexteriornswitch whiledo-while, for statement.

Ex480.c continueCancellation of the current and start of the nanextncycle of a while,do-while, for loop.

goto <brand> Continuation of the program at the with <brand> : <statement> marked spot.

Comment :Except for break in the switch statement, the above statements should be used very sparingly (better not at all), since they run counter to structured programming and produce the dreaded spaghetti code.
in theInternship are above instructions for Lo¨sunGof exercises etc. not allowed.

Chapter 5

Structured data types

Wirightin this chapter new Mpdsidesof data storageear.

- array:
  Grouping of elements of the same type.

- Structure (struct):
  Grouping of components of different types.

- union (union):
  ü bstorageseveral components of different types in the same memory space.

- recordpnType(number)
  basic dataypw e d tfree wapalpablemW crop area.

5.1 fields (arrays)

5.1.1 One-dimensional fields

Data (elements) of the same type are combined in a field. The general convention of a static field is

<type> <identifier>[dimension];

where the square brackets “[” and “]” are an essential part of the agreement. A one-dimensional array is mathematically equivalent to a vector.

Ex510.c

```
//          Examplearr
{
 const int N=5;
 double x[N],y[10];// Declaration

 x[0] = 1.0;                  //
 x[1] = -2;
 x[2] = -
 x[1];
 x[3] =

// access to x[5] , ie, x[N] is not
permitted
```

The square brackets are used in the declaration part of the dimension declarationx[N] and in the instruction part access to individual array elements x[3] . The field can already be initialized during the declaration:

double x[N] = {9,7,6,5,7}

–

<u>Attention :</u>The numbering of the array elements begins with 0. Therefore, onlyon array elements xi, i = 0, . . . , N 1 can be accessed. Otherwise, mysterious program behavior, inexplicable miscalculations and sudden program crashes are to be expected, the cause of which is not obvious because they may only appear in remote program parts.

Typical mistake

//Typical error

{

```
const int N = 123;
      int ij[N] , i;
...
for (i = 1; i <= N;i++)// !!    WRONG!!
  {
    cout << ij[i] << endl;
  }
}
```

The array elements ij1, ij2, ij3, ij4 and the meaningless value of ij5 become
spent, howevernot the very first array element ij0 .

The dimension of a static field must be known at compile time, so only constants or expressions consisting of constants can appear as a dimension.

```
{
const intN=5, M=1;
        int size;
 floatx[5];//Correct
 short i[N];//Correct
 charc[N-M+1];//Correct
 intij[size];//                    !! WRONG!!
}
```

```
//                    String
#include <iostream.h>
#include
<string.h>
main()
{
 const intL=11;            //
 10+1char word[L];

 strcpy(word,"math");      //

 cout << endl << word <<
 endl;

 for (i = 0; i < L;
 i++)
  {
   cout << word[i] <<
   " ;
```

Ex511.c Example: An interesting special case of the field is the character string (string). We initialize the string with the word "math" and print it out in normal type and character by character.

Tueestring

ha¨ttealsoHWithchar

word[L] = "math"; or

char word[] = "math";

initializetwill ka,where in the latter case the La¨ngeof field word ouchsthe la¨ngeenrightCharacter string constant is determined.

Example: Calculation of

L_2-norm of a vector, i.e.,

$$\|\underline{x}\|_{L2} := \sum_{i=0}^{N-1} x_i^2$$

Ex512.c

```
// Array: L_2 n
#include
<iostream.h
#include
<math.h>
main()
{
 const int                    Initialize
 N=10; double
 x[N], norm;
//
 for (i = 0; i < N ;
 i++)                         L_2 standard
   {
     x[i] =
     sqrt(i+1.0);
   }
//
 norm = 0.0;
 for (i = 0; i < N

 count <<'L2 norm        : ' <<
}
```

Als small exampleserveeU.NsyoueFibonaci sequence of numbers,wmooseandbheyoue two-staged recursion

$$f(n) := f(n-1) + f(n-2) \quad n = 2, \ldots$$

Fibo1.c is defined with the initial conditions f (0) = 0, f (1) = 1. To check, we can use Binet's or de Moivre1's formula.

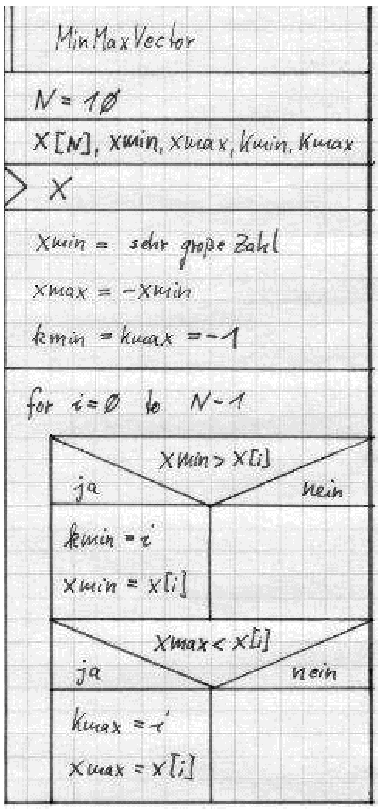
$$f(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

```
//                          Demo of Fibonacci
#include
<iostream.h>
#include
<math.h> main()
{
  const int N =
        20; int i;          //!!      N+    !
        int
     x[N+1];                Calculate Fibonacci
     double fib;
''
     for ( i = 2; i <= N;
                        i++ )
//                          output
  - -
  -                         Check last Fibonacci
  fib = ( pow(0.5*(1.0+sqrt(5.0)),N)
         -pow(0.5*(1.0-sqrt(5.0)),N)
         )/sqrt(5.0);

}
```

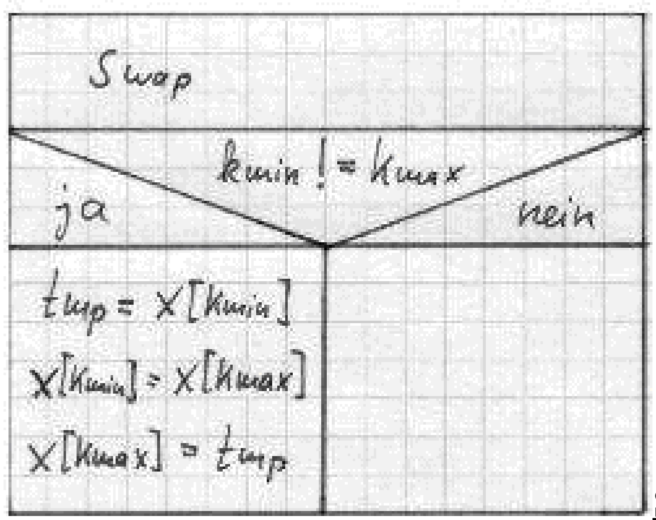
[1] http://www.ee.surrey.ac.uk/Personal/R.Knott/Fib onacci/fibFormula.html

Ex513.c As a further example, the minimum and maximum of a vector are to be determined and the corresponding vector elements are to be swapped with one another (similar to pivoting). This includes the two subtasks:

a) Determine minimum and maximum (and mark the positions). Structogram:



b) VexchangeMin/Max Entryge At Vectorlage0 or at identicalNo swapping is necessary for vector elements.



<u>Structogram</u>:

At theVexchange
futouches
the obvious first
idea x[kmin] =
x[kmax]
x[kmax] =
x[kmin]
not to success. Why?

```
// Pivot for a
vector #include
<iostream.h>
#include
<float.h> main()
{
 const int N=10;
 double x[N], xmin,
 xmax, tmp; intkmin,
 kmax, i;
//Initialize x
 for (i = 0; i < N ; i++)
    {
      cin >> x[i];
    }
//Initialize min/max
 xmin=DBL_MAX; // in
 floats.h xmax = -DBL_MAX;

//Initialize
 indices kmin =
 kmax = -1;
//   Determine
 min/max for (i = 0; i
 < N; i++)
    {
      if ( xmin > x[i] )
        {
         xmin =
         x[i]; kmin
         = i;
        }
      if ( xmax < x[i] )
        {
         xmax =
         x[i]; kmax
         = i;
        }
    }
// Swap pivot elements
// Do nothing for N=0 or constant
 vector if ( kmax != kmin )
   {
    tmp=    x[kmin];
```

5.1.2 Multidimensional Fields

Tueeentryäeenrightso farrightbconsidered1D-Folder areimSpoakbehind each otherstored (linear memory model), for example, is the row vector as

x_0 x_1 x_2 x_3 x_4

double x[5];

agreed and saved as

x_0	x_1	x_2	x_3	x_4

where each cell is 8 bytes long.

A two-dimensional (static) array, for example, a matrix A with N = 4 rows and M = 3 columns

:=

A_{00} A_{01}
$A_{02}A_{10}$
A_{11}
A_{12}
$AN\times$ MA20A21A22
A30A31A32

can also only be stored linearly in memory, i.e.,

A_{00}	A_{01}	A_{02}	A_{10}	A_{11}	A_{12}	A_{20}	A_{21}	A_{22}	A_{30}	A_{31}	A_{32}

- This results in two options for the 2D field declaration: Variant 1: As a 2D array.

```
doubleA[N][M];// Declaration
A[3][1] =5.0;// Initialize A(3,1)
```

- Variant 2 : As a 1D array.

```
doubleA[N*M];          //
Declaration A[3*M+1]
=5.0;//        Initialize A(3,1)
```

Example: As an example we consider the multiplication of the matrix $A_{N\times M}$ consisting of N = 4 rows and M = 3 columns with a row vectoru_M the Lange M . The result is a row vector f N of La¨nge N ,ie, Ex514.c $f_N := A_{N\times M}\, u_M$ .The components of $f = [f_0, f_1, \ldots, f_{N-1}]^T$ bcalculate

to

$$f_i := \sum_{j=0}^{M-1} a_{i,j} \cdot u_j \qquad \forall i = 0, \ldots, N-1.$$

hoherdimensionalefolderkönndeclared and used analogous to version 1will. In variant 2 double B[L,N,M]; be accessed using B[i*M*N+j*M+k].

5.2 structures

The structure defines a new data type which combines components of different types. The type declaration

```
struct <struct_identifier>
 {
   <data declaration>
 };
```

allows the declaration of variables of this type

<struct_identifier> <var_identifier>;

```
// Structure
 {
// new
 structure
 struct
 Student
  {
   long long int register;
               int skz;
               char name[30],
               firstname[20];
  };
// Variable of type
 Student Student
 arni,robbi;
// Data input
 cout << endl << " firstname : ";

 cin >> arni.firstname;
```

Ex520.c Example:We declare a data type to store the persosimilardata of a student.

The assignmentrobbi = arni; copies the complete dataset from avariables to the other. The component firstname of the variable arni (of the type Student) is accessed via
on it.firstname
The data is saved in the form

matriculation	sketch	Surname	First name

awayHa¨ngiGbeforen compiler settingsor. -optionsknockout¨nen smallerunused-teSpEicherlu¨ckin betweenenn componentsimSpoakperformn(Dateaalignme ntst fu¨rightfaster data access).

Ex520b.c Tueestructure student can easily fu¨rightStudents who have several majorsprove, to be expanded.

Ex523.c

```
{
 const int MAX_SKZ=5;

 struct Student_Mult
  {
    long long int register;
                 int skz[MAX_SKZ];
       int nskz; // number of studies
                 char name[30],
                 firstname[20];
  };
// Variable of type
 Student Student
 arni,robbi;
// Data input
 cout << endl << " firstname : ";

 cin >> arni.firstname;
...
 robbi =arni;          // complete
 copy cout << robbi.firstname <<
```

```
// Array of
structures
{                                        // new
 structure student
  {
    ...
  };
 const int N =
      20; int i;                        //
                                         Array
 for (i = 0; i < N; i++)
  {
   cin >>
   group[i].firstname;
   ...
  }
}
```

Ex522.c Tueestructure student incl̇talready have fields as components. On the other hand kȯ̈nenthiseData types can in turn be arranged into fields.

structuresnknockout¨nen in turnat thee structureddatatypesals componentscontain.

```
// Structures within structures
{
 structurePoint3D// simple structure
  {
    double x,y,z;
  };

 struct Line3D              // structure uses
  {
    Point3D
    p1,p2;

 Line3D                     // Declare
                            variables

 cout << "Start point: ";
 cin>> line.p1.x >> line.p1.y >>
 line.p1.z; count <<"End point: ";

 cin>> line.p2.x >> line.p2.y >>
 line.p2.z;
```

In the example above, line.p2 is a variable of the Point3D type, whose data can be accessed using the . Operators can be accessed.

5.3 union

AlleseUnion components are on the same storage area ubhe-shown overlapping. The type declaration

```
union <union_identifier>
  {
 <data declaration>
  };
```

allows the declaration of variables of this type

```
[union] <union_identifier><var_identifier>;
```

Components of the union are accessed like a structure.

Ex530.c

```
// Union
#include
<iostream.h>
main()
{                                   //newunion
 union operand
  {
    internal
          i;                        // longest data
    float
          f                         // declare

 cout << endl << "Size
 (operand) : "

 ui =                              // init as
 cout << endl << ui << "                   << uf <<""

 uf =                              // Init as
 123;

 out =                             // Init as
 123;
}
```

operand The memory requirement of aunion

i

f

i.e

judgetafter the bigatenComponet (here sizeof(double) = 8). The union is used to store spacesaving should be reserved for experienced programmers because of the possibility of errors (ie, no

use in internship).

5.4 recordmt

```
// enum
#include
<iostream.h>
main()
{                                    new
//
 total day
    Monday Tuesday Wednesday
    Thursday,
    Friday Saturday Sunday



 if ( weekday ==
 monday )
  {
  cout << "bad mood" <<
  endl;
```

ofrightrecordcbypist onebasicypwedtfriiwowpalpablemWcrop areathes may beillustrated by the days of the week.

Ex540.c

```
day weekday;            // variable of
                           enum
weekday = monday;       // data init
```

C++ has a predefined type bool, which takes the values false and true

acceptncan. InCla¨ßt himselfthesby definition

enum bool {false,true}

inanalogrightWiron reach,wobegg falsebyH 0U.Ni.e truebyH 1representativepresentedwhich conforms to § 2.1.1 and § 3.3.

5.5 General type definitions

The general type definition

typedef <type_definition> <type_identifier>

is the consequent advancement to freely definable types.

Ex550.c

```
// general type
definitions
Main()
{
//                    new
 typedef char    Text[10
 typedef
 struct
  {
   double        Point3

//                    newvariabl
 Boolean
 text entry;
 Point3D pts[10], p = {1, 2,
 3.45};
}
```

The program example below illustrates the definition of the three new types Boolean, Text and Point3D.

Interestingly, a variable of type Text is now always a character string variable with a (max.) length of 100. Note also the initialization of the variable p. Even a constant of the type Point3d can be declared and initialized with it.

Chapter 6

pointer

Up until now, we've always accessed variables directly, meaning it didn't care where the data was stored in memory. A new type of variable, the pointer, stores addresses considering the type of data stored there.

6.1 agreement of pointers

If the pointer to an object of type int is denoted by p, then

int *p;

whose declaration, or general is made by

[storage class] <type> *<identifier>;

defines a pointer to the data type <type>.

```
//          Pointer
{
 structure
 student
  {
   ...
```

SOknockoutfnenyoueefollowing pointer variables are defined

```
char          *cp;          // pointer on char
internal   x, *px;          // int variable, pointer on int
float         *fp[20];      // array of 20 pointers on float
float         *(fap[10]);   // pointer array of 10 on floats
University student *ps;     // pointer structure on Student
```

Ex610.c

6.2 pointer operators

derightwell¨re reference operator(address operator)

&

<variable>determines the address

of the variable in the operand.

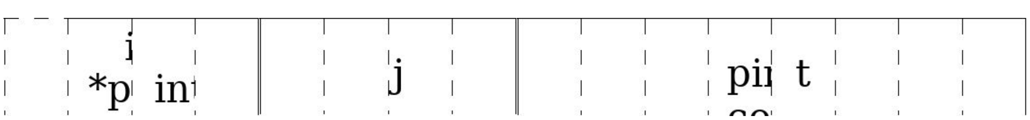
derightwell¨redereference operator(access operator)

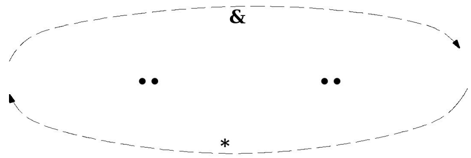
*<pointers>

Ex620.c

```
// Pointer
operators
#include
<iostream.h>
main()

  i    =                       //i   = 10
  pint = &i;         // pointer
  j    =             initialization

  *pint =                      //i   =

  *pin +=                      // i +=
}
```

allows (indirect) access to the data pointed to by the pointer. The data can be manipulated like a variable. In the example aboveacts *pint as an int variable and accordingly all operations defined for it can be performed with it.





Attention : In the program fragment

```
{
  double *px;
  *px =3.1;        // WRONG!
}
```

willalthough storage space fu¨rightreserves the pointer (8 bytes), but the value of px is still undefined and so the value 3.1 is converted to a dafu¨rightnotdesignated memory area written ⇒mysterio¨seprogram stepreU.Ni.e-error.

There is a special pointer constant 0 (NULL in C) which refers to the (hexadecimal) memory address 0x0 (= nil) and which can be tested for as a pointer variable.

6.3 Pointers and Arrays - Pointer Arithmetic

Arrays use the linear memory model, ie an element that follows in the index is also physically stored in the immediately following memory area. This fact allows pointer variables to be interpreted as field identifiers and vice versa.

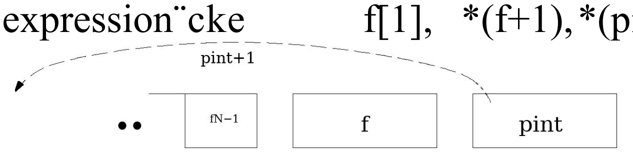
```
{
  const int N = 10;
        int f[N],*pint;// array and pointer

  pint =&f[0];// init pointer
}
```

Field identifiers are always treated as pointers, hence the program line

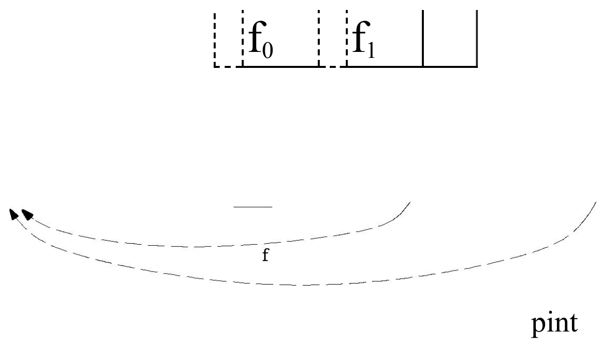
identical with

```
pint = &f[0];

   pint = f;
```

Ex630.c consequentialGtherefore set the expression¨cke f[1], *(f+1),*(pint+1),



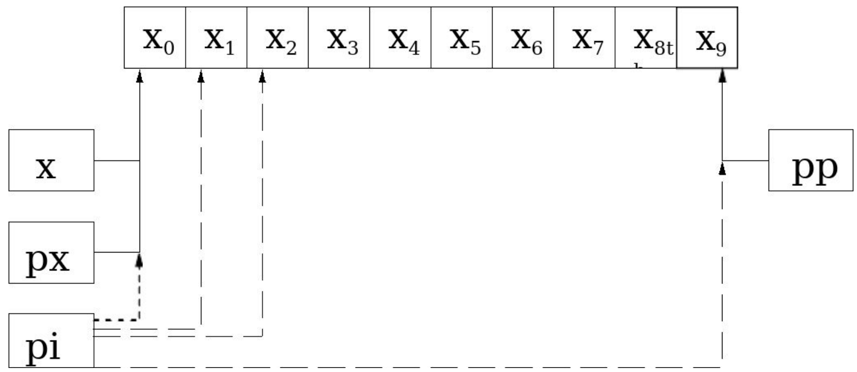
pint[1] represents the identical access to array element f1.



The address represented by (pint+1).will result in (address in pint) + sizeof(int) . In this case, int designates the data type on which thePointer pin points. Access to other array elements fi, i = 0 . . . N 1 is analogous.

The following operators are applicable to pointers:

- Comparison operators: == , != , < , > , <= , >=
- Addition + and subtraction -
- increase ++ ,decrement -- and compound operators += , -=




```
// pointers and arrays
#include
<iostream.h>
main()
{
  const int N=10;
  double   x[N], *px, *pp,

//                          initialize
  px =
  for (i = 0; i < N; i++ )
   {
     *(px+i) =(i+1)*(i+1);          x[i] =
     ...
   }
//                  output x;
// pointer pi as loop variable pp =x+N-1;/
  pointer at last element of x for ( pi = x;
  pi <= pp; pi++)
   {
     count <<""    << *pi << endl;
   }
```

Ex630.c

To demonstrate, we consider an example in which an array is first defined and initialized in a conventional way and then output using pointer operations.

6.4 Dynamic arrays using pointer variables

Ex641.c Until nowreferred a pointer to already provided (allocated) memory fu¨ronesimple variable, structure, field. foot¨rightHowever, a pointer can also be added to the typegrightSpoak areadynamicHallocatetwearth.Mr Jerezandbone usesthe new conclusion¨sselwortnew . The memory allocated in this way can be changed usingdelete to be released again.

```
// Dynamic variables and
// Dynamic array 1D
#include
<iostream.h> main()
{
  intn,i;
  double*px, *pvar;

  count <<"Input n :     ";    cin

  px = new                        //Allocate

//initialize array
  for (i = 0; i < n; i++ )
   {
     px[i] = (i+1)*(i+1);
   }

// output x;
  for ( i = 0; i < n; i++)
   {
     count <<"" << px[i] << endl;
   }

  delete []px;//                  Deallocate

   pvar = new                     // Allocate
   double;                        variable
  *pvar = 3.5 * n;
}
```

The statement px = new double[n]; allocates n*sizeof(double) bytes forenn pointer px. danachcanntheresdynamicefeld px how one staticfeldbe treated. However, dynamic fields make better use of the existing storage space, since this can be released with the delete command and used again for other purposes.

Ex640. danger:The above dynamic field declaration is only fu¨rightC++ goodvalid.In Cmueatnother commands are used - here are the differences.

C++C

#include <malloc.h>

px = newdouble[n];px = (double*)

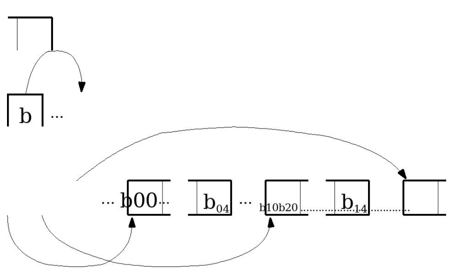
malloc(n*sizeof(double)); delete []px; free(px);

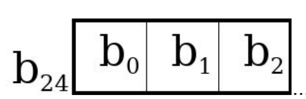
Ex642.c§ eggnbetweenone-dimensionaldynamic field

lt himself on the one handbyH one one-dimensional dynamic field (analogous to variant 2 in 5.1.2) as well as by a pointer to a field of pointers. This looks like this for a matrix with n rows and m columns.

```
// Dynamic array 2D
#include
<iostream.h>
main()
{
  intn,m,i,j;
  double**b;// pointer at pointers at double


  b = new(double*[n]);      // Allocate row
  for (i = 0; i < n;
  i++)
   {                         // Allocate
     b[i] = new

  for (i = 0; i < n; i++)// b
   {
     for (j = 0; j < m; j++)
       {
         b[i][j] = (i+1)*(j+1);
       }
   }
...
  for (i = n-1, in >=
  0, i---)
   {                            //deallocate
     delete[]bi];
   }                            //deallocate row
  delete[] b;
```

First the pointer must be allocated to the line pointer, only then can the memory for the individual lines be requested. When deallocating memory, all rows must also be freed again. For the case n = 3 and m = 4, the figure shows how the data is stored in memory.





Attention: There is no guarantee that the individual rows of the matrix are arranged consecutively in memory. Thus, the storage of the dynamic 2D array differs from the storage of the static 2D array, although the syntax of the element access b[i][j] is identical. On the other hand, this matrix storage is more flexible, since the rows can also have different lengths (sparse matrices or matrices with a profile).

Demonstrationofwrongcode

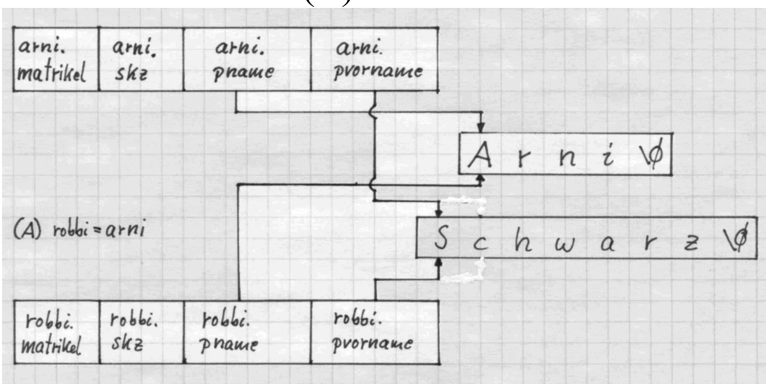
```
// wrt. copying a structure with pointers
#include <iostream.h>
#include <strings.h>// strcpy, strlen

struct Student2
  {
    long long int register;
              int skz;
              char *pname, *pfirstname; // Pointers in
              structure
  };

Main()
{
 Student2 arni, robbi;

 cin>>                              // read
// Allocate memory for arni.pfirstname
 arni.pfirstname = new char[strlen(tmp)+1];// Don't
 forget "+1 strcpy(arni.pfirstname,tmp); // and copy input
 on it

// the same with the remaining data on arni
  ...
// roughother
              wrongcopying
 robbi = arni;
// points (A)
  ...
 delete [] arni.pfirstname; // deallocate
// points (B)
  ...
//Let us allocate some tiny dynamical array char
 *tiny;
 tiny = new
 char[5];
 strcpy(tiny,"tin
 y");
```

Ex643-

showrightknockoutnenturn appear in structures or general types. Here, however, graspleCare should be taken when using the dynamic fields, since fu¨rightstatic variables otherwise uncritical operations

plomoreHto miraculoüsprogram
inreceivewasearnknockoutcan.

What does the data store look like at times (A), (B) and (C)?



- robbi does not have its own dynamic fields.

- delete [] arni.pfirstname;
  thus also releases the memory area addressed by robbi.pfirstname and thus robbi.pfirstname points to a memory area that is no longer reserved, which the program may use as it sees fit.

- tiny = new char[5];
  taket himselfennfriibecomenSpoak placeU.Ni.eandbhe writesheyn spa¨ter.

- Under LINUX-gcc, at time (C) robbi.pfirstname points to the same address as

the pointer tiny, so that the data from robbi.pfirstname can be output using
cout << robbi.pfirstname << endl;
gives the output tiny.

- way out:

Ex643- Eshimeatnwas¨rightrobbi own dynamic fields are allocated and the contents of the dynamic fields of arni mueatnbe copied to this. (see assignment operators and copy constructors fu¨rightclassesalso§ 11).

6.5 pointers to structures

```
// Pointer at structure
{
  structure student
  {
   ...
  };
  student peter, *pg;

//       init peter
  ...

  pg =&Peter;// pointer at peter

  cout << (*pg).firstname; //

  conventional access cout<<pg-

  >firstname; // better access
```

§ We consider the structure Student (5.2) and define a pointer to it.

Tuee accesses(*pg).firstname and pg->firstname are vo¨lliGaequivalent. All-thing improves the latterclearly the readability of a program, especially if the pointer represents a dynamic array of type Student. This is particularly evident when accessing array elements of firstname (ie, single characters). Access to the 0th character is via

or	pg->firstname[0] (*pg).firstname[0]	or or	*pg->firstname *(*pg).firstname

and accessing the 3rd character using			
or	pg->firstname[3] (*pg).firstname[3]	or or	*(pg->firstname+3) * ((*pg).firstname+3)

Ex650.c Note that pg->firstname represents a pointer to type char and the dereferencing operator * is performed before the + addition. Conjecture and test what you get when you use *pg->firstname+3.

6.6 reference

```
// Reference
// i, ri, *pi are different names for one
variable #include <iostream.h>
Main()
{
 internali; // i
 int ? = i; // declaration reference
 on i int *pi;

 pi = ? // declaration pointer

 on i; i = 7;
 cout << i << ri << *pi;

 ri++;
 cout << i << ri << *pi;

 (*pi)++;
 cout << i << ri << *pi;
}
```

AeReference is an alias (pseudoname) fu¨righta variable and can do the samehow these are used. References (unlike pointers) do not represent an object of their own, ie no additional memory is required for them.

§ referencesnbecome hodfigureto the parameterbresulta nfunctions used, they-he 7.2. Another useful application is the reference to an array element, structure element or inner data of a complicated data structure as shown below, derived from the example on page 54.

```
// Reference and dynamic array of type
student
#include
<iostream.h>
main()
{
 structure student
   {
     long long int register;
                int skz;

 student                          // pointer at
 group[4];
// data input;

 i = 3;
 {
// reference on comp. of structure
    int&rskz = group[i].skz;
// reference on structure
    Student&rg= group[i];
// reference on comp. of referenced
    structure long long int&rm=
    rg.matriculation;

    cout << "Student no. " << i << ";
    cout << rg.firstname << " " << rg.name
    << ", "; cout << rm << ", " << rskz <<
    endl;
```

Ex662.c

Chapter 7

functions

7.1 definition and declaration

Purpose of a function:

- ofsO earliern will oneprogram partlinothernprogram sectionsn againin[illegible]To the program andbmore evidentand more manageable toodesign, this part of the program is programmed once as a function and called up in the rest of the program with its function name.

- Alreadycompleted functions kön̈enwas¨rightother programs of other programmers available¨gunGare provided, analogous to the use of pow(x,y) and strcmp(s1,s2) in§ 3.6.

In the general form of the function definition with

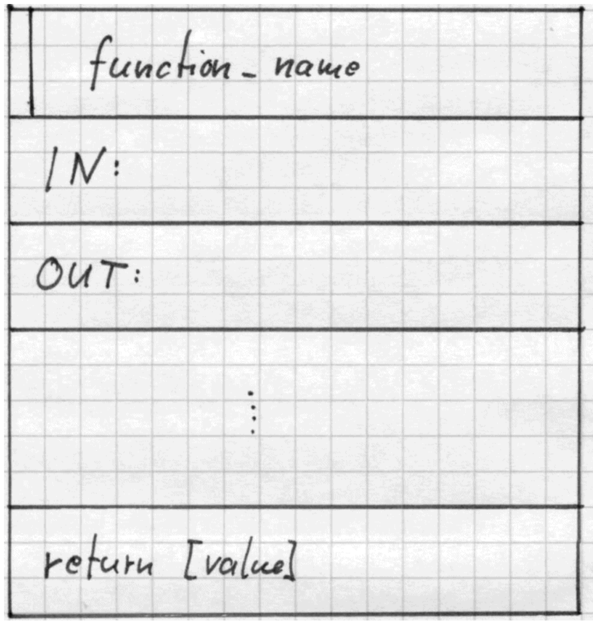
```
<storage class><type> <function_name> (parameter_list)
{
 <agreements>
 <instructions>
}
```

putVagreementU.Ni.eapplice partennfunctional co¨rphethererightU.Ni.e <type> laysthe type of Ru¨cvaluefixed. The combination <function_name>and (parameter_list) uniquely identifies a function and is therefore used ascalled signature of a function. The function definition becomes fu¨righteach function exactly once benorequired.

The difference is the function declaration

```
<storage class><type><function_name> (parameter_list) ;
```

ineach source file nGwmooseyoueFunction <function_name> calls.



Structogram:

Ex710.c Example: We write the calculation of sgn(x) from page 29 as a function.

```
// Demonstration of function declaration and
definition

#include <iostream.h>

double sgn(doublex);// declare function sgn
- - -

Main()
{
 double a,b;
 - - -

 b=                    // function

 count << "sgn(" << a << ") = " << b <<
 endl;
}
 ..

double                 // definition of function
sgn(double x)
{

 y = (x > 0 ? 1st : 0th) + (x < 0 ? -

 return                // return value of
}
```

Remarks: The sgn() function is uniquely described by its signature.
thesHatfu¨rightdeclarationnU.Ni.edefinitelynb
eforenfunctionsnyoue Consequences:

(i) Some more (or even more) identical function declarations
double sgn(double x);
are allowed in the example above.

(ii) togetheradditioneFunction declarations with other parameter lists arebelieves, e.g.:
double sgn(double*
x); double sgn(int
x);
since the arguments differ from the initial definition. However, we have not yet defined these new functions.

(iii) Ae togetheradditionedeclarationn(see § 7.2)
double sgn(double& x);
is not allowed because the signature is as under (i). Therefore, the compiler cannot figure out whether the function under (iii) or the function under (i) is in the statement
y = sgn(x);
meantis.

(iv) Different functions with the same name are identified by their different parameter lists, see item (iii).

(v) derightRu¨cvalueof a function cannot be used to identify itare tightened, the declarations
double sgn(int
x); int
sgn(int x);
knockout¨nennottdifferencesnwearth(samee signature)U.Ni.ethereforerightleantthecom piler from this source text.

7.2 parametersdownhille

When designing a program, we distinguish three types of parameters of a function:

INPUTParameter data is used in the function but not verachanges,that is, they are constant within the function.

INOUTParameter data is used in the function and verachanges.

OUTPUT parameter data are initialized in the function and, if applicableverachanges.

Ex721.c Programmatically we will not distinguish between INOUT and OUTPUT parametersdifferentiate. There are generally three Moequalsidesthe programtechnicalnPassing parameters

1. ü bresulte n rightData of a variable (by value).

2. ü bresultthe address of a variable (by address)

3. ü bresultenrightreferencese.gouchf onevariablee(engl. :by reference), whereby an address is transferred in a hidden manner.

Comment:
Wifuse a variable in the function as a constant¨tztwill, then shouldthey are also treated as such, ie pure INPUT parameters should always be marked as const in the parameter list. This increases security against unintentional data manipulation.

7.3 Rureturn valueeof functions

Each function has a function result of data type <type> . As types are allowedbe used:

- simple data types (§ 2.1.1),
- structures (§ 5.2), classes,
- pointer (§ 6.1),

- References (§ 6.6),

jedocHno fields and functions - therefor¨rightbut pointer to a field or aFunction and references to fields.

derightRu¨cvalue(function result)will with

return <result> ;

antherescalling program andbresult. A special case are functions of the kind

void f(<parameter_list>)

fu¨rightwhich no Ru¨cvalue(voii.e =empty)is expected, so with

return ;

inthe calling program back¨cswept will.

Wirightlook at the mplidesenrightparametersbresultam exampleenrightsgn function with variable double a .

ü bresultart	parameter list	call	effect of x++	effect of (*x)++	use	recommendation
by value	double x	sgn(a)	internal	—-	INPUT	[c]
	const double x		not allowed	—-	INPUT	C [simple data types]
by address	double* x	sgn(&a)	internal	internal/external	INOUT	C
	const double* x		internal	not allowed	INPUT	C [complex data types]

	double* const x		not allowed	internal/externa l	INOU T	[c]
by reference	double& x	sgn(a)	internal/externa l	—-	INOU T	C++
	const double& x		not allowed	—-	INPUT	C++

tabell7.1 :
Mpbsidesen rightparametersbres
ult

The ”by-reference” variant double &const x is rejected by the compiler and the ”by-address” variant const double* const x , ie,Pointers and data you¨fenlocally not verachangetwearth,i s tpractically meaningless.

Ex731.c

```
// demonstration of
void void fun(const
int);

Main()
{
 ...
 fun(13);
 ...
}

voidspass(const int i)
{
 cout << "But now it's time" << i
 << endl; return;
}
```

examplesfu¨rightFunction results:

floatf1(...) float
number [struct] studentf2(...)
structure student int*f3(...)
Pointer to int number
[struct] Students*f4(...) Pointer to Structure
Student
internal(*f5(...)) [] Pointer to array of int
numbers
internal(*f6(...)) () Pointer to a function that has the
result type int

Remarks:

Aefunctionnrepresentfseveraleru¨cdelivery instructions return[<result>]; besit,eg, one in each branch of an alternative. However, this is no longer clean structured programming.

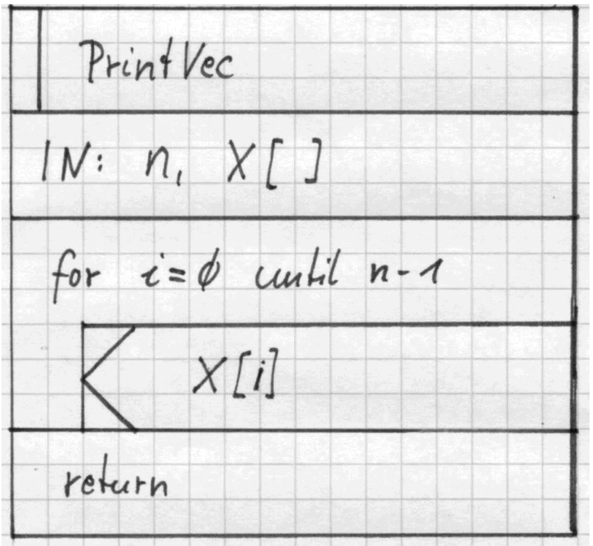
⇒
= Each function should have exactly one return statement at the end of the function bodybsit(Defaulti.ewas¨righttheresInternship).

7.4 fields as parameters

Staticefields ko¨nenanalogous to their declaration as function parameters ubresultwearth. HoweversmueatnAllesedimensions,except the honextndimension, to be known at compile time.

Ex740.c As a first example, we consider the output of a (static or dynamic) 1D field, ie, a vector x of length n.

Structogram:




```
#include <iostream.h>
//
//Print elements of a vector (length n)
//
void PrintVec(const int n, const double x[])
{
 int i;

 cout << endl;
 for (i=0; i<n, i++)
   {
    count <<"" << x[i];
   }
 cout << endl <<
 endl; return;
}
//Main()
{
     const int
          N=4;
      int n,i;
 double f[N] =
 {1,2,3,4}; double *df;

 cin >> n;
 df = newdouble[n];//     Allocate dynamic array
//Initialize df
```

```
//initialize df
...
PrintVec(N,f);// Print static
arrayPrintVec(n,df);                    //
Print dynamic array
}
```

AlsnmsLet's consider the output of a 2D static array, ie, aMatrix with MCOL columns and NROW rows. The number of columns must be defined as a global constant here, otherwise the following function cannot be compiled.

```
#include
<iostream.h>
                                 // global
//.................................................................
//        Print elements of a matrix
//           (nrow rows and fixed number MCOL of
//
//                 doesn't
//void PrintMat_fix(const int nrow, const double
a[][])
//doesn't help to fix that
//void PrintMat_fix(const int nrow, const int ncol,
//const double a[][])
//
void PrintMat_fix(const int nrow, const double
a[][MCOL])
{
 int i,j;

 cout << endl;
 for (i=0, i<nrow, i++)
  {
    cout << "row " << i
    << ":"; for (j=0;
    j<MCOL; j++)
     {
      count <<"" << a[i][j];
     }
    cout << endl;
  }
 cout << endl <<
 endl; return;
}
//Main()..........................................................
{
 const intNROW=4;          // local

 PrintMat fix(NRO          // print static
}
```

Ex740.c Unfortunately kön weatheryouefunctionn PrintMat_fixnowrightwas¨right static2D Folder(Matrices)apply, and then only fu¨rightthose with NCOL=3 columns - yesa matrix double aa[7][9] can no longer be output with this function. However, we can interpret the 2D field as a 1D field of length NROW*MCOL and thus generalize the function in such a way that any static 2D fields and dynamic 1D fields that can be interpreted as 2D fields (as in version 2 on page 52) can be handed over.

```
#include <iostream.h>
//..............................................................................
// Print elements of a matrix
// (nrow rows and ncol columns)
//
void PrintMat(const int nrow, const int ncol, const
double a[])
{
 int i,j;

 cout << endl;
 for (i=0, i<nrow, i++)
  {
   cout << "Row" << i
   << ":"; for (j=0;
   j<ncol; j++)
    {
     count <<'" << a[i*ncol+j] ;
    }
   cout << endl;
  }
 cout << endl <<
 endl; return;
}
//Main()..........................................................................
{
 const int NROW=7,MCOL=9;  // local
 constants double a[NROW][MCOL]
 = { ...};// static           matrix

 cin >> nrow; cin >>          // read dimensions
 ncol;                        of b
 b = new double
 [NROW*MCOL];
// initialize matrix b
 ...                          // Pointer on first
// output matrices
 PrintMat(NROW,MCO
```

Since the PrintMat function expects a 1D array (i.e. a pointer), from the static2D Feld a one pointerouchfyoue first rowenrightmotherxandbresultwearth. Therefore, a[0] appears in the corresponding call line.

7.5 Declarations and header files, libraries

Normally, the source code of a computer program is composedconsists of (substantially) more than one source text file. With it functions, data structures (and global constants, variables) and macros from other source text files (*name.cc*)usedtwearthkanbusedmomnheader files(name.hh, name.h) which the declarations fu¨rightcontain the source text file name.cc.

7.5.1 Example: printvec

printvec. § We want to use the PrintVec and PrintMat functions programmed in 7.4 in another code (ie, main program). First we copy the definitions of the two functions (and everything else that is needed for compiling) into the new file printvec.cc.

```
//      printvec.
#include <iostream.h>
void PrintVec(const int n, const double x[])
{
 ...
}
void PrintMat(const int nrow, const int ncol, const
double a[])
{
 ...
```

The file printvec.cc is now compiled (without linking it!)

LINUX> g++ -c printvec.cc

Ex751- whereby the object file printvec.o is created. The main program in

```
//        Ex751-old.cc
//        declarations of functions from
void PrintVec(const int n, const double x[]);
void PrintMat(const int nrow, const int ncol, const
double a[]); Main()
{
 const intN=4, M=3;                    // local constant
                                       // static matrix a
 double a[N][M] = {4,-1,-0.5, -1,4,-1, -0.5,-1,4,
 3,0,-1 },

 PrintMat(N, M,                        // print
 a[0]);                                matrix
}
```

Ex751-old.cc bOnebusytwe11rightthe declarations of the two functions. Compiling the main file

LINUX> g++ -c Ex751-old.cc

creates the object file Ex751-old.o which has to be linked with the other object file to the finished program a.out

LINUX> g++ Ex751-old.o printvec.o

satofficialscompile and link l[illegible]can also be expressed in a command line¨ckin

```
LINUX> g++ Ex751-old.cc printvec.cc
```

whereby some compilers expect the main() main program in the first source text file (here Ex751-old.cc).

printvec.h Tueedeclarationsnimmain programmwas¨rightyouefunctionsnouchsprintvec.cc scream-We use the header file printvec.hh

```
//        printvec.hh
//        declarations of functions from

void PrintVec(const int n, const double x[]);
void PrintMat(const int nrow, const int ncol, const
```

U.Ni.e weather substituteenndeclaration partlimmain programmbyHyouebeforenPrePer-processor instruction

Ex751.c #include "printvec.hh"

```
//        Ex751.
#include
"printvec.hh"
Main()
{                                              // local
                                               constant
 double a[N][M] = {4,-1,-0.5, -1,4,-1, -0.5,-1,4,
 3,0,-1 },

 PrintMat(N, M,                                // print
 a[0]);                                        matrix
 PrintVec(N,u);
```

which automatically inserts the content of printvec.hh before compiling Ex751.cc.

Tueeanfuapproxchen" " around the file name indicates that the hea-derfile printvec.hh in the same directoryhow to find the source file Ex751.cc.

The command

LINUX> g++ Ex751.cc printvec.cc

in turn generates the program a.out.

student.

7.5.2 Example: students

student.h§ §

Wirightknockoutnenalso self-defined data structures, eg the data structuresSave Student, Student Mult from 5.2 and Student2 from 6.4 and global constants in a header file student.hh.

```
//          student.
const int MAX_SKZ = 5;

structure student
 { ... };
struct Student_Mult
 { ... };
struct Student2
 { ... };

void Copy_Student2(Student2& lhs, const
```

The new function Copy Student2 is defined in student.cc, where the function corpheouchsEx643-correct.cc copyt would.

```
//          student.
#include <strings.h>
#include "student.hh"

void Copy_Student2(Student2& lhs, const
Student2& rhs)
{
 lhs = rhs;
//Allocate memory and copy data
 lhs.pname       = new
 char[strlen(rhs.pname)+1];
 strcpy(lhs.pname,rhs.pname);
 lhs.pfirstname= new
 char[strlen(rhs.pfirstname)+1];
 strcpy(lhs.pfirstname,rhs.pfirstname);
```

Ex752.c Since the Student2 structure is used, the header file student.hh must also be included in student.cc. The new Copy Student2 function can now be used in the main program Ex752.cc to copy a structure. Of course, the main program needs the header file student.hh for this.

The command

LINUX> g++ Ex752.cc student.cc

finally creates the program a.out.

7.5.3 A simple library using student as an example

umthe repeated compiling togetheradditionalrightSource files and the ones with themLibraries are used to avoid possibly long lists of object files when linking. At the same time, libraries have the advantage that you can make your compiled functions (along with the header files) available to others in a compact form without having to reveal your programming secrets (intellectual property). This is demonstrated using the (very simple) example from §7.5.2.

- Generate the object file student.o (compile)
  LINUX> g++ -c student.cc

- Generating/updating the library libstud.a (archiving) from/with the object file student.o. The library identifier stud is freely selectable.
  LINUX> ar r libstud.a student.o
  TueeArchiving options (here, only r) ko¨nenwith the usedcompilers vary.

- Ex752.c Compile the main program and link with the library from the current directory
  LINUX> g++ Ex752.cc -L. -lstud

The following steps are necessary to compile and link the program without using a library.

student. g++ -c student.cc

g++ -c Ex752.cc

student.o

a.o g++ Ex752.o student.o

$Ex752.cc \longrightarrow Ex752.o$

awayku¨rzeni.eis also modaily:

$\longrightarrow$ a.out

Ex752.cc, student.cc g++

Ex752.cc student.cc

When using the libstud.a library, the process is as follows

student.cc g++ -c

student.cc

g++ -c

student.o aright right left bSt and i.e. a

*lib*stud.a

a.o g++ Ex752.o

student

Ex752.cc

Ex75 2c c

Ex752.o

□ -L. -lstud

wowsb eggalready existing library in turn abkurtw earthc a n :

L. -lstud

*Ex752.cc, lib*stud.a g++ Ex752.cc -

a.out

7.6 The main program

Tueesyntax used so far fu¨rightthe main program

```
Main()
{
 ...
}
```

is always used by the compileras

```
int main()
{
 ...
 return 0;
}
```

understood, since for functions without type specification theType int used as default willwedtthe standard ru¨cvalue 0A Ru¨cvalue 0bedeu-tes that the main program was processed without errors.

The program processing can be stopped at any time, also in functions, with the instruction exit(<int_value>); be aborted. The value <int_value> is then the return value of the program and can be used for error diagnosis.

```
#include
<iostream.h>              // needed to declare

void fun(const            // declaration of

int main()
{
 int n;

 cin>> n;
 if(n<0) exit(-           // choose an error

 fun(s);                  // call fun()

 return                   // default return value
}

void fun(const intn)// Definition of fun()
```

TheresThe program above breaks off when n < 0hrunGimmediately from andreturns error code 10. The exit statement can also be used in fun()will.

As with other functions, the main program can also be called with parameters, but in

int main(int argc, char* argv[])

the parameter list (more precisely, the types of parameters) prescribed, where

- argv[0] the program name and
- argv[1] . . . argv[argc-1] the arguments when calling the program as character stringsandbresult.

- Esgilt always argc≥1,i.eathe program name always ubresultwill.

```
//        for a real C++ solution, see Stroustrup, p.126
#include <iostream.h>
#include<stdlib.h>// needed for atoi, exit

void fun(const                   // declaration of

int main(int argc, char* argv[])
{
 int n;

 cout << "This is code " << argv[0]

 << endl; if (argc > 1) // at least one

 argument
  {
   n = atoi(argv[1]); // atoi : ACSII to
   Integer
  }
 else // standard input
  {
   cout << " input n :
   ";cin>> n;
   cout << endl;
  }

 fun(s); // call fun()


void fun(const int n)
{
 if (n<0) exit(-10); // choose
 an error code

 cout << "But now it's time" << n <<
 endl; return;
```

Ex760.c

Tueefunctionn atoi(char*) (= ASCII to int) converts the ubdevoted Sign-concatenate to an integer and is declared in stdlib.h. By means of the analogfunctionn atod(char*) lt himself one float numberas parameters andbresult. After compiling and linking, the program can a.out using

LINUX> a.out

or.

LINUX> a.out 5

be started. In the formercase the value of n is read from the keyboard,imsecond case, the value 5 from the command line ubtaken andn assigned. An elegant, and real C++-Ls, regarding the handover of command line parameters can be found in [Str00, pp.126].

7.7 Recursive Functions

☾

ˣfunctionsnknockoutfneninC/C++ can be called recursively.

```
// definition of function power
double power(const double x,
const int k)
{

 if ( k == 0)
  {
   y = 1.0;              // Stops
  }
 else
  {
   y = x * power(x,k-    // recursive
   1);

 return                  // return value of function
}
```

Ex770.c Example: The power to be realized.

x^k with x ∈

, k∈

can also as

= $x \cdot x^{k-1}$ $k > 0$ 1 $k = 0$

7.8 eggnbigoutersExample: Bisection

–

The example on page 39 was about determining the zero of f (x) := sin(x) x/2 in the interval (a,b), with a = 0 and b = 1. Provided that f (a) > 0 > f (b) this problem can be solved (for continuous functions) using bisection. The bisection algorithm essentially consists of the steps for each interval [a, b].

(i) c := (a + b)/2
(ii) is|f(c)| close enough to 0 ?
(iii) Inwhich interval haelevenemußi.eHsearch further ?

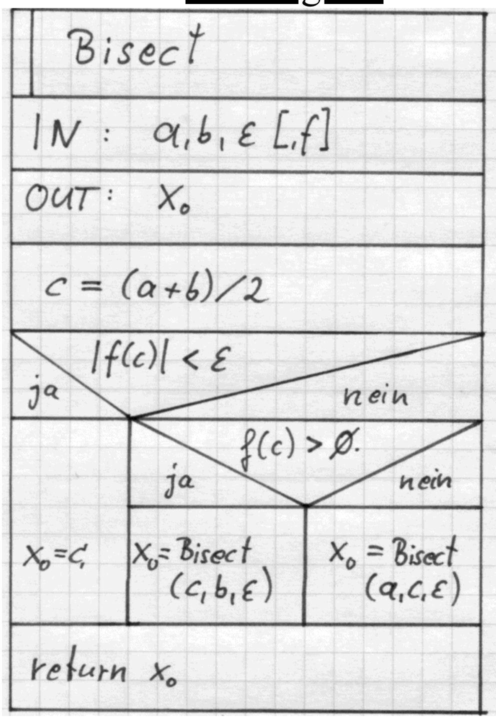
thesist one classicrecursion,wobeggpunkt(iii)youen / Anexterecursionn initiatesU.Ni.ePoint (ii) is intended to guarantee the cancellation of the recursion. Formally knockoutcanweatherthesput it like that¨cken:

x_0 := Bisect(a, b, ε) :=

c :=(a + b)/2if |f (c)| < ε
Bisect(c, b, ε)else if f (c) > 0

Bisect(b, c,
e) otherwise,casesf(c) < 0

Structogram:



thesgives the function definition fu¨rightBisect() which with
Bisect1.c x0 = Bisect(a,b,1e-6);
calledn willU.Ni.eto version 1 of the bisection program
futouches

```
double Bisect1(const double a, const double b, const
double eps)
{

 fc = sin(c) -
 0.5*c;
 if ( fabs(fc) <
 eps )                      // end of
  {
   x0 = c;
  }
 else if(fc > 0.0 )          // search in right
                             interval
  {
   x0 =                      // ie, fc < 0.0
   Bisect1(c,b,eps);
  }

 return                      // return the
}
```

To make the program a bit more flexible, we will fix the in Bisect1()

programmed function f(x) by the global function

```
double f(const                // declare and
  {returnsin(x)                - 0.5*x ;}   //
```

substitute. at the same timeiGknockoutcouldnwe replace the function parameter eps with a globalReplace constant EPS with bale, resulting in version 2.

Bisect2.c Tueeflexibilityäthe bisection function läßtcontinue to riseheynby entering the function f(x) to be evaluated as a variable in the parameter list ubresult.A functionnas a parameter/argument is always used as a pointer ubresult,ie, a functionas an argument, like the declaration fu¨rightf6 built on page 76being. Specifically, this means:

double (*func)(double) is a pointer to a function func with a

double variablesalsargumentsnU.Ni.edouble

as the type of the Ru¨creciprocal. This allows us the function declaration and definition of

Bisect3()

```
                // declaration of Bisect3
double Bisect3(double (*func)(double), const
                double a, const double b, const
                double eps=1e-6);
...
Main()
{...}
              // definition of Bisect3
   double Bisect3(double (*func)(double),
                const double a,
                      const double b, const

 fc =            // calculate value of parameter
 if ( fabs(fc) < eps )
  {
  x0 = c;                     // end of
  }
 else if(fc > 0.0 )
  {
  x0 =                        // search in right
  Bisect3(func,c,b,eps);      interval
  }
 else                         // ie, fc < 0.0
  {
  x0 =

 return                       // return the
 x0;
```

The fourth argument (eps) in the Bisect3() parameter list is optionalArgument which is not used when calling the functionbresultwearthgot to. In this case, the default value specified in the function declaration is automatically assigned to this optional argument. In our case, the call would be in the main program

x0 = Bisect3(f,a,b,1e-12)

youeRecursion at |f (c)| < ε := 10−12 cancel, waduring

x0 = Bisect3(f,a,b)

```
                                  // declare and
double g(const double x)
                                  //definition of functions
                                  g(x)
```

Bisect3.c

beautifulbegg|f(c)| < ε :=
10−6 stops. We could now add
another function

declare and define, and call the bisection algorithm in version 3.
with it:

Bisect3.c x0 = Bisect3(g,a,b,1e-12)

Comment:Da weatheroureal sargumentsnin Bisect3andbdevotedfunctionn funcis a pure INPUT parameter, we should mark it with const. However, the correct labeling of the first argument is in Bisect3

```
double Bisect3(double (* const func)(double), const
                    double a, const double b, const
                    double eps=1e-6);
```

a bit confusing at the beginning.

– — ∈

ourrightProgram works satisfactorily fu¨rightf (x) = sin(x) x/2 and yieldsfootrightthe input parameters a = 1 and b = 2 the correct solution x0 = 1.89549, the same for a = 0 and b = 2, however, the (trivial) solution x0 = 0 is not found here because a = 0 is entered would. With the inputs a = 0, b = 1 or a = 1, b = 0.1 (x0 := 0 [a, b]) the program aborts after a while with a segmentation fault, since the recursion does not abort and at some point the fu The memory (stack) reserved for function calls is no longer sufficient.

–

knockoutnenwe secure our program in such a way that, for example, the existing zerox_0 = 0 as well asin[0, 1]to thesin [1, 0.1] is found? Which cases can occur with regard to the function values f (a) and f (b) (preliminary assumption: a < b)?

(i) f (a)>0 > f(b) (ie, f(a) > 0 and f(b) < 0), e.g., a = 1, b = 2

=⇒ Standard case in Bisect3().

(ii) f(a) > 0 and f(b) > 0, e.g., a = 0.5, b = 1.5, respectively

–

$f(a) < 0$ and $f(b) < 0$, eg, a=1, $b = 0.5$

⇒

possibly

nonezero =cancel.

(Eskönn There may be zeros in the interval, which we do not find with the bisection method ko¨nen!)

as well as (iii) $f(a) = 0$ or $f(b) = 0$, better $|f(a)| < \varepsilon$ etc.

or =⇒

= a or b are the zero,

a and b are a zero.

(iv) $f(a) < 0 < f(b)$, e.g. $a = -1$, $b = 0.1$

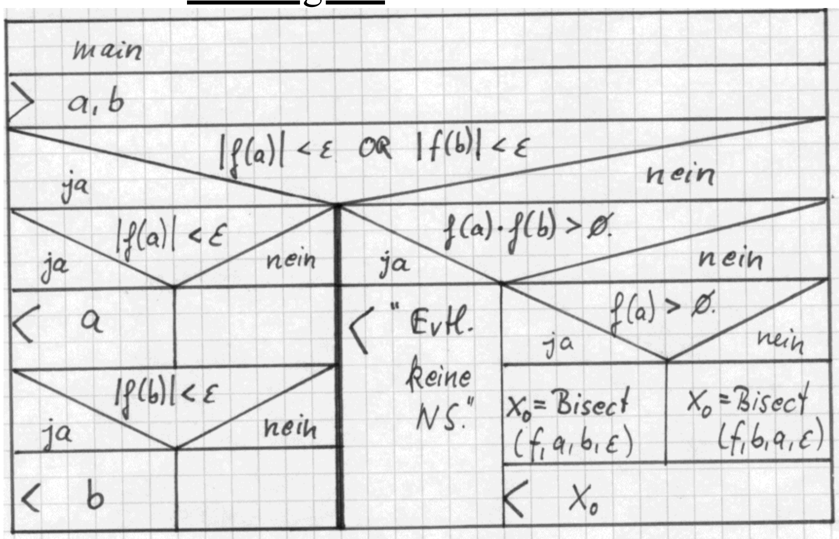
Swap a and b =⇒ Case (i).

(v) $a = b$ =⇒ included in (ii) and (iii).

$b < a$ =⇒fuMrton (i) or (iv).

Bisect4.c thisecase distinction fuMrtus to the following structural diagram and *version 4*.

Structogram:



Bisect5.c —— To thescroendnFinally we define further functions in the program h(x) = 3 ex, t(x) = 1 x2, ask the user which math. function fu¨rthe root search is to be used and calculate the root(s) in the given interval. This selection can easily be implemented with a switch statement and leads to version 5 of the program.

Bisect6.c Comment:The three functions Bisect[1-3]() differ in their parameter lists. Therefore all three functions can be used under the name Bisect(), since their signatures differ and thus the compiler knows exactly which Bisect() function should be used.

Chapter 8

The data type class (class)

§§ ofrightdataayp student2ouchs6.4 incl.dynamicedata structuresnthrough whichInitialization and copying of the corresponding variables must be specially implemented each time (see also 7.5.2). A great advantage, among many others, of the class concept in C++ is that data structures with dynamic components can also be handled in the main program like simple data types. Of course, this requires some preliminary work.

≈ Ae class(class) is a data type with associatedothernmethodn (functions) and willsdHcreated in a structure and can be used analogously.The methods of a class always have access to the data of this class.

students.c students.hh Starting from the structure Student2 we derive a class Students. We save all declarations of the class in the header file studenten.hh and all definitions in studenten.cc. A modified variant with regard to the pointer initialization can be found in the studs.hh and studs.cc files.

studs.h

studs.c

8.1 Class declaration data and methods

```
//          students.h
class students
{
// Data in
students public:
  long int
       register;
       int skz;

//                           Methods in
publi

//                                   Default constructor (no
 Students(

//                                     Constructor with 4
 Students(const char firstname[], const char name[],
        const longinternal          mat_nr =0,

//                                    Copy
 students(const students&

//                                    destruct
 ~Students(

//                                    Assignment
 students& operator=(const students &

//                                    Further
 - -
 -
```

Here is the declaration of the Students class with the absolutely necessary methods.

TueelistedhonorednmethodnLet's look at them in the order given.

8.2 The constructors

Constructors are used for the initial initialization of data in a class. In general, a class should have at least the following constructors.

- Definition of the default constructor (no arguments)

```
//          default
Students ::
Students()
{
 matriculation = skz
 = 0; pname =
```

In the construction student :: denotes the scope operator
::the acces¨rigksinceenrightMethod Students() on class Students
and is part of the signature of this method
(function). The default constructor is used in
the main program with students robbi;
called up automatically, so that all data from robbi is
initialized with 0 or the zero pointer.

•

```
//           parameter
Students :: Students
        (const char firstname[], const char
         name[], const longintmat_nr,const
         intskz_nr)
{
 matriculation
 = mat_nr;skz=
            skz_nr;

 pname = new char[strlen(name)+1];
 strcpy(pname,name);
 pfirstname= new char[strlen(firstname)+1];
```

Definition of a parameter constructor

With the parameter constructor, a variable of type Students
declared, defined and initialized at the same
time. Students
arni("Arni","Black",812,7981579); It would
also be possible:

```
    char tmp1[20],
tmp2[20]; long int tm;
    int ts;

cin >> tmp1 >> tmp2 >>

tm >> ts; Students
```

parameter constructorsnknockout¨nencontain optional parameters whoseStandard values are already specified in the declaration (Page 94). Thus, a variable definition of the students type would also be about students arni("Arni","Schwarz"); goodvalid.

- Definition of the copy constructor

```
//          copy
Students :: Students(const students&
orig)
{
 matriculation =
 orig.matriculation;s
 kz= original.skz;


 if (orig.pname !=0 )
  {
    pname = new char[strlen(orig.pname)+1];
    strcpy(pname,orig.pname);
  }
 else
  {
    pname = 0;
  }
 if (orig.pfirstname != 0)
  {
   pfirstname= new
   char[strlen(orig.pfirstname)+1];
   strcpy(pfirstname,orig.pfirstname);
  }
 else
```

The copy constructor allows definition in terms of another variable of the same class, as in

students mike(arni);

8.3 The Destroyer

```
//                          destruct
Students :: ~Students()
{
 if (pfirstname != 0)
 delete [] pfirstname; if(
       pname != 0) delete []
```

Every classbsitsexactlyand adestructor,which when leaving the Guvalid-range of a variable (end of block, end of function) is called automatically. The main task of the destructor is mostly to free dynamic memory of the class.

8.4 The assignment operator

Constructors always access uninitialized data. To assign the data of arni to the already initialized (with default values) variable robbi, ie, robbi = arni; , an assignment operator has to be defined, wmooseessentially from the functional co¨rpseriouslycomposed of destructor and copy constructor. However, here is before uberpuovens,whether the right side of the assignment (ubdevotedvariable)nottidenticalH to thelinkinpageof assignment (this) is.

```
//                          assignment
students& students :: operator=(const students
& orig)
 if ( &orig != this)                // test
  {
    if                 delete []
    (pfirstnam         pfirstname;

    matriculation = orig.matriculation;
    skz= original.skz;

    if (orig.pname !=0 )
      {
        pname = new char[strlen(orig.pname)+1];
        strcpy(pname,orig.pname);
      }
    else
      {
        pname = 0;
      }
    if (orig.pfirstname != 0)
      {
       pfirstname = new
       char[strlen(orig.pfirstname)+1];
       strcpy(pfirstname,orig.pfirstname);
      }
    else
      {
       pfirstname = 0;
      }
  }

 return *this;
}
```

Wsuppurationeari.ebe here on the keyword overloading of operators in theLiterature [SK98, §16], [Str00, $11] referenced.

8.5 The print operator

eggnnot absolutely necessary, but right
nowadditionalrightoperator is the print operator
fu¨righta class, which the data output using

cout << robbi <<endl;

ermosamen target.

```
#include
<iostream.h>            output
friend ostream & operator<<(ostream & s,
                        const students & orig);
```

The declaration in students.hh

allows, thanks to the friend identifier, to use the Students class to define a new method of the ostream class (declared in iostream.h). The definition is then:

```
ostream & operator<<(ostream & s, const students
& orig)
{
     return s << orig.pfirstname << ""<<
          original.pname << " "
```

Ex851.c

```
#include
<iostream.h>
#include
"studenten.hh"

int main()                          // Default

 {              // start block
                                    // Constructor with
   args students arni("Arni", "Black", 812,

   robbi =                          // Assignment

  }             // end              // Destructor for

 students                           // Copy

 cout << "mike : " << mike << endl;

 cout << endl;
//Data in Students are public therefore:
 cout << "Access to public data : ";
 cout << "mike.pfirstname = " << mike.pfirstname
 << endl;

 return 0;
```

§ Wellrightkinwe use the example Ex643-correct.cc (or Ex752.cc) from 6.4much easier to write and extend.

The command line
LINUX> g++ Ex851.cc students.cc

generatedt h e r e sexportaudibleeProgram.

8.6 data encapsulation

The data in studentshave been classified as public, meaning anyone can access this data, as with mike.pfirstname . To protect this data from unwantedaccess to schu¨tzenand possibly nachtra the data layout¨glicH
In tok a, capsuletmomnyouedatenso into the class that they only ubheaccess methods are available. The relatedequaleClassification is by the syllogism¨sselwortprivate specified. So that achanget himselfthe declarationpart of the class students in

```
//        students2
.hh #include
<iostream.h>

class students
{
// Data in students are private
now!! private:
long int register;
        int skz;
      char *pname, *pfirstname;

//                    Methods in
students public:
//constructors, Destructor, Access operator
 ...
```

```
// Output operator
 friend ostream & operator<<(ostream & s, const students &
 orig);

// methods to access the private data
//                              Methods for data manipulation
 in students void SetFirstName(const char
 firstname[]);
 void SetName(const char name[]);
 void SetMatrikel(const
 longinternal      mat_nr); void
 SetSKZ(constintskz_nr);


//                        Methods that don't manipulate data
 in students const long int& GetMatrikel() const;
 const int& GetSKZ()
 const; const char*
 GetFirstName() const;
 const char* GetName()
 const;
Student2.hh
                          };
```

There are two const declarations in the above methods. A const at the end of the declaration line indicates that the data in Student will not be modified by the corresponding method, eg, GetSKZ. The const at the beginning of the line belongs to the result type and indicates that the data referred to with the reference int& must not be changed. This ensures that the data cannot be manipulated unintentionally using pointers or references.

The access methods are defined as follows:

students2.

```
//                              students2.c
#include "studenten2.hh"
...

void Students :: SetFirstName(const char
firstname[])
{
 if (pfirstname != 0) delete
 [] pfirstname; pfirstname =
 new
 char[strlen(firstname)+1];
 strcpy(pfirstname,firstnam
 e);
 return;
}

void Students::SetSKZ(constinternal
                                        skz_nr)
{
 skz =
 skz_nr;
 return;
}
...
const char* Students :: GetFirstName()
const
{
 return pfirstname;
```

```
//        Ex861.
#include <iostream.h>
#include <strings.h>
#include
"studenten2.hh"

int main()
{
 Students mike("Arni", "Schwarz", 812, 7938592);

 cout << "mike : " << mike
 << endl; cout << endl;
//Data in Students are private therefore -->
inaccessible:

//        Data in students are private
 cout << "Access to private data via methods: " <<
 endl;
 cout << "mike.pfirstname = " << mike. GetFirstname() <<
 endl;

// mike. GetFirstName()[3] = 'k'; // not allowed
because of 'const'
// char *pp = mike. GetFirstName();// not allowed
 because of 'const' char tmp[40];
 strcpy(tmp,mike. GetFirstName()); // allowed
```

Ex861.c thisenew access methods könen howfollowtto be used:

Some access functions, eg SetSKZ and GetSKZ, are so short that a function call is actually not worthwhile because of the effort involved in passing the parameters. In this case, the declaration and definition of a method are linked in the header file, and the method/function is defined inline. These inline lines replace the function call each time.

```
// students3.hh

#include

<iostream.h>

class Students
{
 ...                                // inline
 void
 SetSKZ(constintskz_nr)
                                    // inline
  { skz = skz_nr; };
```

students3.h

Chapter 9

File input and output

The objects cin and cout used for I/O are (in iostream.h)predefined variables of class type stream. In order to read from or write to files, new stream variables are now created, namely of the type ifstream for the input and of the type ofstream for the output. The file name is transferred when the variable is created (C++ constructor).

Ex912.c

9.1 Copy files

```
//Ex911.cc
#include
<iostream.h>
#include
<fstream.h>

int main()
{
 char
 infilename[100],outfilename[100];
 char str[100];

 cout << " Input file: "; cin >>
 infilename; cout << "Output file: ";
 cin >> outfilename;

 ifstreaminfile(infilenam
 e);
 ofstreamoutfile(outfilename);

 while (infile.good())
 {
  infile >>
  str; outfile
```

Ex911.c The following program copies an input file to an output file, but without spaces, tabs, line breaks.

On the other hand, if you want to copy the file identically, you have to read in and out character by character. The get and put methods from the corresponding stream classes are used for this.

```
//                              Ex912.
#include
<iostream.h>

int main()
{
 char
 infilename[100],outfilename[100];
 char ch;

 cout << " Input file: "; cin >>
 infilename; cout << "Output file: ";
 cin >> outfilename;

 ifstreaminfile(infilenam
 e);
 ofstreamoutfile(outfilename);

 while (infile.good())
 {
   infile.get(ch);
  outfile.put(ch);
 }
```

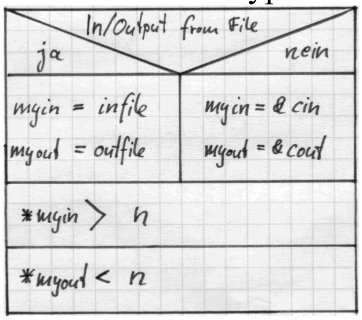
9.2 Data input and output via file

```
//                FileIO a
#include
<iostream.h>              // needed for ifstream and

int main()
{
 int n_t, n_f;
// input file
 ifstream infile("in.txt");

 cout << "Input from
 terminal: ";cin>> n_t;

 cout << "Input from file "
 << endl; infile >> n_f;
//check                   it
 cout << endl;
 count <<"Inputfrom terminal was " << n_t
 cout << "Output          file was " << n_f
 from
 cout << endl;           output
 ofstream outfile("out.txt");

 cout << "This is an output to the

 terminal" << endl; outfile << "This is an

 output to the file" << endl; return 0;
}
```

_FileIO Data input and output via file and terminal can be used in combination.

9.3 Switching input/output

_FileIO sometimeslis a problemabhabusysSwitching between file IO and terminal IO wu¨desswheor nmyUnfortunately, in this case you have tolike to work on the types istream and ostream.



```
//            FileIO  b
#include <iostream.h>
#include <fstream.h>

int main()
{
 inside,
 tf; bool
 bf;
//variables for IO streamsistream*myin;

 ostream*myout;          input file
 istream*       = new
 infile                  ifstream("in.tx
 ostream* outfile = new

//       Still standard OK
//       Decide whether terminal-IO or file-IO
 cout << "Input from terminal/file -
 Press 0/1 : ";
 cin>>

 if (bf)
   {                     // Remaining IO via
     myin=
     infile; myout
     = outfile;

   }                     // Remaining IO via
 else
   {
     myin=     ?

 (*myout) << "Input: ";
 (*I sold)>> n;
//check
 (*myout) << endl;
 (*myout) << Input was << n << endl;
 (*myout) << endl;

 (*myout) << "This is an additional

 deleteoutfil          // don't
 e;

 return
}
```

FileIO	__	FileIO

Aevery comfortable Monequalsinceof switching the input/output usingCommand line parameters can be found in the examples.

output formatting

Tueeoutputeandbhestreamings(<<)cannvmostfor mattwearth. Ae smallA selection of formatting is given here, more on this in the literature.

mat. We use thosevariables

```
double da = 1.0/3.0, db = 21./2, dc = 1234.56789;
```

- Standard output:
  ```
  cout << da << endl << db << endl << dc << endl << endl;
  ```
- mehrightgoodvalideDigits (here 12) in the output:
  ```
  cout.precision(12); c out << ...
  ```
- Fixed number (here 6) of decimal places:
  ```
  cout.precision(6); cout.setf(ios::fixed, ios::floatfield); c out << ...
  ```
- Output with exponent:
  ```
  cout.setf(ios::scientific, ios::floatfield); c out << ...
  ```

•

Ru¨ckputouchfstandar
d
editionand:cout.setf(0,
ios::floatfield); c out <<
...

•
alignmentG(right-
hand¨end)U.N.i.eWildcard (16
characters):cout.setf(ios::right,
ios::adjustfield); cout.width(16);
cout << da <<
endl;
cout.width(16)
; cout << db
<< endl;
cout.width(16)
; cout << dc
<< endl;

Aegeneral Lo¨sunGusing standard manipulators is in [Str00, § 1.4.6.2, pp.679].

- Hexadecimal output of integers: cout.setf(ios::hex, ios::basefield); cout << "127 = " << 127 << endl;

tips and tricks

11.1 Pr[illegible]

Wirightalready know the Praprocessor appice
#include <math.h>

preproc. § which inserts the content of the file math.h at the appropriate place in the source file before the actual compilation. Similarly, certain parts of the source code can be included or ignored when compiling, depending on the dependency of the test (analogous to an alternative as in 4.3) which is carried out with a preprocessor variable.

variablenensPreprocessorswearthby means of
#define MY_DEBUG
definetand we könnalsoHtesting,whether they are defined:

```
#ifdef MY_DEBUG
   cout << "In debug mode"
<< endl; #endif
```

Analog can with

```
#ifndef
MY_DEBU
G#define
MY_DEBU
G #endif
```

students4.h zunächsttestedtwearth,whether the variable MY_DEBUG has already been defined. If not, then it is defined now. This technique will haufigurebusedaround to prevent the declarations of a header file from being included more than once in the same source text.

```
//        students4.h
#ifndef FILE_STUDENTS
#define FILE_STUDENTS
//declarations of the header
file
- - -
```

OnerightPreprocessor variablencannalsoH onebe assigned value

#define SEE_PI 5

wmoosethen in Praprocessor tests(orrightin the program as a constant)can be used:

#if (SEE_PI==5)
 cout << " PI = " <<
M_PI << endl; #else
// empty or
statements #endif

AeHoffrightenedapplendingbexistsinenright allocation oneWfirst to onePrePer-processor variable if not already defined

#ifndef M_PI
#define M_PI
3.14159 #endif

Deswfesterknockout¨nenmacrosdefined with parameters

MAX(x,y) (x>y ? x #define : y)

and used in the source code.

 cout << MAX(1.456 , a) << endl;

mehrightandbhePrprocommandistetc. in[God98th]U.Ni.e[Str00,$A.11] toFind.

11.2 timing in the program

Ex1121.c To theperimeterGbeforen C++giveHHon somefunctions,wmoose it allowyoue runningto determine the time of certain program sections (or the entire code). The corresponding declarations are provided in the header file time.h.

```
//        Ex1121.
..
#include                          // contains

int main()
{
 ...
 double time1=0.0,               // time measurement

//        read
 ..
 tstart =                        //

//        some
 ..
 time1 += clock() -              //
 tstart;
 ...                             // rescale to

 count <<"time = " << time1 << " sec."
 << endl;

}
```

Esknockout¨nenany number of time measurements can be made in the program (at some pointbut these in turn slow down the program!). Each of these time measurements needs a start and an end, but the times of different measurements can be accumulated (by simply adding them up).

Ex1121.cIn the Ex1121.cc file, the function value of a polynomial of degree 20 at the

$k=$positione x ie, $s = \Sigma_{20}$ $a.k \cdot xk$,bcalculated.Tuee21Coefficients ak and the value x

are provided in the file input.1121. The function value is calculated in two mathematically identical ways in the program. Variant 1 uses the pow function, while variant 2 calculates the value of xk by continuous multiplication.

The different runtime behavior (cause !?) can now be proven by time measurement and improved by progressively activating compiler options for program optimization, e.g

```
LINUX>g++
Ex1121.ccLINUX>
g++ -O
Ex1121.ccLINUX>
g++ -O3
Ex1121.cc

LINUX>g++ -O3 -ffast-math
Ex1121.cc
```

The program is started by

```
usingLINUX>a.out <
input.1121
```

11.3 profiling

of courselicHkdone in a program the time measurement in each functionwrite to determine the runtime behavior of the functions and methods. However, this is not necessary since many development environments already provide tools for performance analysis, ie profiling. At a minimum, the time spent in the functions and the number of function calls are output (often graphically). Sometimes this can be resolved down to single lines of source code. In addition to the professional (and fee-based) profiling and debugging tools, simple (and free) commands for this are also available under LINUX/UNIX.

```
LINUX>g++ -pg
Jacobi.cc
matvec.ccLINUX>a.out

LINUX>gprof -b
a.out >
outLINUX>
        less out
```

The compiler switch -pg accommodates some additional functions in the program so that the runtime behavior can be analyzed by gprof after the program run. The last command (can also be an editor) displays the redirected output of this analysis on the screen.

11.4 debugging

It is often necessary to follow the program flow step by step and, if necessary, to have variable values etc. output for control purposes. Next enrightalways working, but annoyingend,method

```
...
cout << "AA" << variable << endl;
...
cout << "BB" << variable << endl;
...
```

areOften professional debugging tools are availableavailable. Here is one again(free) program under LINUX presented.

```
LINUX> g++ -g Ex1121.cc
LINUX> ddd a.out &
```

The handling of the various debuggers differs greatly. With the ddd debugger, the input file can be specified with set args < input.1121 and the test run is started with run, which is stopped at previously set break points. There, the program can be followed step by step using the source code.

bibliography

[Cap01] Derek Capper. Introducing C++ for Scientists, Engineers and Mathematicians. Springer, 2001.

[CF88] MatthewsClauss and Gu¨ntherightfisherman. Programming with C. VerlagTechnique, 1988.

[Cor93] microsoft corp Get in the right way*in C++*. Microsoft Press, 1993. [Dav00] Stephen R Davis. C++ fu*rightdummies*. boarding school Thomson Publ., Bonn/Albany/Attenkirchen, 2nd edition, 2000.

[Erl99] HelmutErlenkø

CprogramnbeforenAbeginningan.

Rowohlt,1999[God98] Edward Gode. ANSI C++:

short & good. O'Reilly, 1998.

[Her00] dietrightHerrmann. C++ fu¨*rightnaturalist*. Addison-Wesley,Bonn, 4th edition, 2000.

[Her01] Dietmar Herrmann. Effective programming in C and C++. vieweg, 5th edition, 2001.

[HT03] Andrew Huntand David Thomas. The Pragmatic Programmer. Hanser textbook, 2003.

[Jos94] Nicolai Josuttis. Object-oriented programming in C++: from the class to the class library. Addison-Wesley, Bonn/Paris/Reading, 3. edition, 1994.

[KPP02] Ulla Kirch-Prinz and Peter Prinz. Everything about object-oriented programming. Galileo Press, 2002.

[KPP03] Ulla Kirch-Prinz and Peter Prinz. C++ fu *rightC programmer*. GalileoPress, Oct. 2003.

[Mey97] Scott Meyers. Programming C++ more effectively. Addison-Wesley, 1997.

[Mey98] Scott Meyers. Programming effectively in C++. Addison-Wesley, 3rd, updated edition, 1998.

[OT93] Andrew Oram and SteveTalbott. *Managing projects with make*. O'Reilly, 1993.

[SB95] Gregory Satir and Doug Brown. *C++: The Core Language*. O'Reilly, 1995.

[SK98] Martin Schader and Stefan Kuhlins. Programming in C++. RowohltVieweg, 5th revised edition, 1998.

[Str00] Bjarne Stroustrup. The C++ programming language. Addison-Wesley, 4. updated edition, 2000.

[Str03] Thomas Strasser. *Programming with style. A systematic introduction*. dpunkt, 2003.

index

#define, 109
#if, 110
#ifdef, 109
#ifndef, 109
#include, 109

abort test, 41
rejecting cycle, 38
    abort test, 38
alternative, 27
Instruction, 3, 4, 25
argc, 84
argv[], 84
array, see field assembler, 5
atod(), 85
atoi(), 85
onzaestyp,45, 57
Expression, 13 edition
    cost, 103
    Files, 103
    formatting, 107
    new line, 9

char, 7
class, 93-101
    data encapsulation, 99
    declaration, 94 method

seeMethod, 93
compile, 20
conditional, 109
g++,4-6, 83, 111, 112
gcc, 3

debugging, 112
delete, 62
do-while loop, see non-rejecting cycle
double, 7

Notepad, 3 input
cin, 103
Files, 103
Decision operator, 29 enum, see enumeration
type

		false, 15, 17, 57
Library, 5, 6, 20, 80, 83		field, 45
update, 83		declaration, 46
generate, 83		Dimensions, 46
left, 83		dynamic, 62–66, 78
binalogarithm,3 8		allocate, 62
Bisection, 40		deallocate,64
(, 86		one-dimensional, 45, 62
), 91		field elements, 46
bit, 17		initialization, 46
blocks, 3, 4, 25		multidimensional, 51, 63
beginning, 25		Numbering, 46
end, 25		static, 45–52, 77
Locationẗ,26		Dimensions, 46
boolean, 7, 57		String, 47
break, 43		Fibonacci, 49

bytes, 17	115	find

Unix command 23
float, 7
float.h, 23, 37, 51
_FLT DIG, 23
_FLT EPSILON, 23, 37
_FLT MAX, 23, 51
_FLT MIN, 23
for loop, *please refer* Zaoil cycleFunction, 71–91
Define, 71
Declaration, 71, 80
functional co¨rphe,7 1
in-line, 101
parameters, see parameter return value,see function
on result recursively, see recursion signature,
71–73
function result,74
void, 74
Floating point number, 9, 14, 15
ü brun,37
Accuracy, 36, 41
running variable, 35
Header file, 3-6, 80, 101
in-line, 101
Heaviside, 28
if-then-else, see alternative inline, 101
int, 7
comment
C, 3, 4
C++, 4
Constant, 9–11
characters, 9

floating point number, 9
global, 88
integer, 9
mathematical, 20
thong, 9
symbolic, 10
running variable, 34
floating point number, 35
limits.h, 23
_INT MAX, 23
_INT MIN, 23

Left, 5, 20, 83

macro, 10
main(), 3, 4, 84
math.h, 20
acos(), 20
salted(), 20
atan(), 20
cell(), 20, 37
body(), 20
exp(), 20
fabs(), 20
floor(), 20
fmod(), 20
log(), 20
_ME, 20
_MPI, 20
pow(), 20
sin(), 20
sqrt(), 20
tan(), 20
Matrix, 52, 63, 64, 77, 78
Multiple choice, 42
method, 93
copy constructor, 95

Define, 101
declaration, 101
destructor, 96
in-line, 101
parameter constructor, 95
print operator, 97
Default constructor, 94
assignment operator, 97methods
access methods, 100

new, 62
non-shedding cycle, 38
abort test, 38
zeros, 39

object code, 5
object file, 6
operands, 13
operator, 13
arithmetic,14
bit-oriented, 17
more logical, 17
comparison operator, 15parameter

by address, 75
by reference, 75
by value, 75
const, 75
Field, 76
function, 88
main(), 84
Matrix, 77, 78
optional argument, 89
Vector, 76 pointers, see pointers
Preprocessor,5,10,109
Profiling, 111 program
exportear,3, 4
Source file, 6, 80
compile, 3, 4, 83
edit, 3, 4 source text, see source file

reference, 68
recursion, 86
abort test, 86
function, 86
Segmentation fault, 90

Signum, 29, 72
sizeof(),8 memory
allocate, 62
deallocate,64
Segmentation fault, 90
string.h, 21
strcat(), 21
strchr(), 21
strcmp(), 21
strcpy(), 21

strlen(), 21 struct, see Structure Structure, 45, 52-55
in structures, 54
Pointer up, 67
Pointer in, 65
college student, 82
library, 83
Class, 94
structure, 53
Student2, 65, 82
suffix, 6

switch
Multiple choice, 42
time.h
clock(), 110
__CLOCKS PER SEC, 110
true, 15, 17, 57
union, 45, 56
using namespace, 6
variable, 7–8
storage class, 7
type, 7
Vector, 45, 76
Branches, see alternative void, 74
truth table, 18
while loop,*please refer* more dismissivecycle
Zaoil cycle,33
abort test, 35
hands, 59–69
address operator, 60
Arithmetic, 61
on structure, 67
declaration, 59
Dereference operator, 60
in structure, 65

Null pointer, 60
reference operator, 60
undefined, 60
access operator, 60
assignment operator,16