**C++ 2019/20**

**3.10.2019:**

Datentypen:

int → ganzzahlige Zahlen

float → Gleitkommazahlen

double → Gleitkommazahlen mit doppelter Genauigkeit

char → einzelner Buchstabe

string → Zeichenkette

bool → kann nur wahr oder falsch sein

Schleifen:

while-Schleife: !Wird sofort geprüft!

(Bedingung) {

Operatione, Inhalt

}

for-Schleife:

(Variable; Bedingung; Operationen) {

Inhalt

}

do-while-Schleife: !Wird nachher geprüft!

do {

Inhalt der ganz ausgeführt wird

} while (Bedingung) {

Inhalt der überprüft wird

}

Verzweigungen:

if-Verzweigung:

(Bedingung) {

Inhalt

} else {

Inhalt

}

!ODER!

} else if (Bedingung) {

Inhalt

}

switch-Verzweigung;

(Variable) {

case 1 : cout << …

case 2 : cout << ...

…

(default) : cout << … () = nicht notwendig

kann nur mit Grunddatentypen verwendet werden (char, int, …)

Variable:

Datentyp Name {}; {} --> Initialisierung

(int i) --> falsch

Int i = 3; //Initialisierung

Int i {}; Init. Mit Std. Wert vom Datentyp

Funktionen:

Datentyp Funktionsname () --> kein Parameter oder (Datentyp N., Datentyp N.)

Void --> wenn die Funktion nicht zurückgibt

Int sum (int & a int b) { //Deklaration

Operatoren:

+, -, \*, /, +=, -=, /=, \*=, --, ++

Int i=2 i += 2 --> i=i+2

Cout << i -- i; //2

Cout << --; //0

&&, !, ||

Bitoperatoren:

& = 0x0001 & 0x1001 = 0x0001

XOR = 0x0001 XOR 0x1001 = 0x1000

|= 0x0001 | 0x1001 = 0x1001

~= 0x0001 ~ 0x1110

0x0001 << 2 à 1 à 0x0100 = 4

0x0100 >> 2 à 4 à 0x0001 = 1

Int i = 4; //0x0100

I = i >> 2;

I = i << 2;

String:

Ist ein Klasse

String S = “Beispiel”;

S = 'B’’e’’i’’s’’p’’i’’e’’l’\0

1 2 3 4 5 6 7 8 9

s.length()

S[0]

For (int i=0; i < s.length; i+=2)

Cout << s[i];

For (int i = 0; s[i]!=’\0’; i++)

If(i%2 == 0)

Cout << s[i];

Sting char2bit (char x) ‘A’=65 --> “0x0100001”

Erg =”0x”;

D = 128

If (X/D!=0) {

erg+=”1” ; x-=d;

} else {

Erg+=”0”;

D /= 2;

Int changeS(“Al-ex”);

“aL-EX”

Return 4;

Zeichen in eine Oktale/Hexadezimale Zahl umwandeln:

Modularisierung:

Schnittstelle wird definiert à steht immer in einer header-Datei (andere Endung **.h**)

**.h** #include „.h“ à Datei für include in “”

Zirkelbezug à führt zu Endlosschleifen

Guard à #ifndef m4\_H

#define m4\_H

#endif à Ende

Int a = 2, b = 3;

Cout << a/b; //0

Cout << a/(float)b; //0,666... --> ist die alte Variante/casting

Casting = etwas uminterpretieren

Cout << a/static\_cast<float>(b);

Vector<int>x;

For(auto I:x) { //i = int

…

}

Compiler benutzt immer größeren Datentyp

Constandten sind Variablen die ihren Wert nach der Initialisierung nicht mehr ändern

Compile Zeit:

Constexpr int PI=3.14;

Static\_assert(PI<3.14||PI>3.15, “Pi zu ungenau!”);

Laufzeit:

const int x = func();

#define NDEBUG --> streicht assert

#incude<cassert>

...  
assert (log Bedingung); //T

Assertion --> für Testfälle

#include<vector>

Vector<int> v={1,3,5,6};

For(auto I:v) {

Cout << I;

}

Vector --> Container

<int> --> Template

Log. Bedingung:

For(int I = 0; I<10, I++)

Cout << I;

Wert einer Variable:

Int I = 3;

Switch(x) { case 1: …, break;

Case 2: …, break;

Default:

}

Datentyp:

Try {

…

} catch(int &e) {

…

} catch(logic\_error) {

…

}

Rekursionen:

**In Python-Visualizer**

Int sum(unsigned int) {

Int erg = 0;

For(;x>0;x--){

Erg += x;

}

Return erg;

}

GV --> so wenig wie möglich

Main

X f1

E --> mit jedem aufruf eines Programms --> muss sich

Rücksprungadresse merken

Compiler arbeitet im Stack --> baut auf und ab

Int sum(unsigned int x)

if(I==1) {

Return 1;

} else {

Return (x-1) + x;

}

**Kommt zur TA**

Sum(4) = sum(3) + 4;

Sum(3) = sum(2) + 3;

Sum(2) = sum(1) + 2;

Sum(1) = 1;

Sum(2) = 1 + 2 = 3;

Sum(3) = 3 + 3 = 6;

Sum(4) = 4 + 6 = 10;

Int fact(int x) {

If(x == 1) {

Return 1;

} else {

Return(x – 1) \* x;

}

}

String srev(string s) {

If(s.length() == 1) {

Return s;

} else {

Return(s.erase(0,1), s.length - 1);

}

}

Ggt(2,4) = 2

A,b

A == b --> return a;

A > b --> return ggt(a - b, b);

A < b --> return ggt(b – a, a);

int ggT1(int a, int b) {

If(a==0) {

Return b;

} else if(a > b) {

Return ggt(a-b, b);

} else if(a < b) {

Return ggt(b-a, a);

}

}

Ggt(2,4) = ggt(2,2)

Ggt(2,2) = 2;

Ggt(2,4) = 2

Ggt(5,3) = ggt(2,3)

Ggt(2,3) = ggt(1,2)

Ggt(1,2) = ggt(1,1)

Ggt(1,1) = 1

Ggt2(int a, int b) {

If(b==0) {

Return a;

} else {

Return ggt(b, a%b);

}

}

Ggt(2,4) = ggt(4, 2)

Ggt(4,2) = ggt(2, 0)

Ggt(2,0) = 2

Ggt(5,3) = ggt(3, 2)

Ggt(3,2) = ggt(2,1)

Ggt(2,1) = ggt(1, 0);

Ggt(1,0) = 1

Ggt(0,2) = ggt(2,0)

Ggt(2,0) = 2

Segmentation fault --> Ausgabe ohne Return in der Funktion

Nr --> It

Nr:

Int fact1(int n) {

If(n==1) {

Return 1;

} else {

Return fact1(n-1) \* n;

}

Arbeiten beim Raufgehen

Er;

Int fact2(int n, int acc=1) {

If(n==1) {

Return acc;

} else {

Return fact2(N –1, n\*acc);

}

Arbeiten beim Runtergehen

Fact2(3) = fact2(2,3\*1)

Fact2(2,2) = fact2(1, 2\*3)

Fact2(1,6) = fact2(6)

Fact2(2,3) = 6

Fact2(3) = 6;

Sr:

Int fact3(int n, int acc = 1) {

If(n==1) {

Return acc;

} else {

Acc = n \* acc;

N = n – 1;

Return fact3(n, acc);

}

Arbeiten beim Reigehen. Es werden nur Parameter übergeben, nicht mehr gerechent

Ir:

Int fi(int I, int acc=1) {

While(true){

, If(n==1) {

Return acc;

} else {

Acc=(n--)\*acc

}

}

Fi(3) = 6

**Kommt zur TA**

Fibonacci:

X==0 --> 0

X==1 -->

X f(x-1) + f(x-2)

Fib(4) = 3

2 1

Fib(3)+fib(2)

1

fib(2)+fib(1) + fib(1) + fib(0)  
 1 1 1 0

Fib(1) + fib(0)

1 1

Array:

* + Größe ist nicht mehr änderbar
  + Sizeof --> Anzahl der Elemente berechnen (nur im Main)
  + Funktionen haben nur Zugriff auf die Adresse des Elements
  + Per Definition --> call by referenz

Enum:

* + Bei mehreren Aufzählungsklassen --> enum class
    - Mit Klassen kann ein Wert mehrmals definiert werden
  + Sonst nur enum

Zeiger:

|  |  |
| --- | --- |
| 3 VAR  X | 0x01  4  Y | 0x02 | 0x01 Zeiger  Px | 0x03  0x03  Ppx | 0x04  P immer vor Pointer  & --> Adressoperator  Px zeigt auf x  Pro \* ein Sprung |
| Int x = 3;  Int y = 4; | Int \*px=&x;  Cout << px; //0x01  Cout << \*px //3  Px = &y;  Int \*\*ppx=&px  Cout << \*\*ppx; //4  Cout << ppx; //0x03  Cout << &ppx //0x04  \*ppx = &x;  Cout << \*\*ppx //3  \*\*ppx = -1; |

Unterschied zwischen Referenz und Zeiger: Eine Referenz ist wie ein Zeiger light --> kann nach init nicht mehr verbogen werden.

Man darf die Adresse einer Referenz niemals ändern.

Größe des Pointers=8 Byte

Int ar[3]{1,2,3} im Speicher: 1 2 3

0x1 0x5 0x9

Ar

Int \*p1=ar;

Int \*p2=&ar[2];

E[2] = \*(e + 2)

Inline:

Für kleine Funktionen --> Var = call by referenz

Compiler schriebt den Funktionsaufruf um --> keine Stack Verwendung

Inline schreibt den Code um: ersetzt den Funktionsaufruf --> Stack wird nicht verwendet

Gut für Performance --> schlecht für Programmcode; --> wird nur gemacht wenn es sinnvoll ist

Kommandozeilenparameter:

Int main() --> keine Kommandzeilenparameter

Argc --> Anzahl der Parameter --> 1 = Programmname

Argv --> Parameter

String p --> Conversion von altem Char auf String

Schleife: string p = argv[i];

Parameter in der Kommandozeile mit Abstand getrennt

/ --> gibt das Zeichen sicher weiter und verarbeitet es nicht

Programmname ausgeben --> cout << argv[0];

Kommandozeile auf Gülitgkeit überprüfen --> wenn falsch: Rückmeldung an den Nutzer

Bei Optionen --> Schalter definieren

Tool für Kommandozeilenparameter --> docopt

Lamda Funktionen:

Funktionen ohne Namen

[] --> Erfassungsliste

() --> Parameter

Auto var (par) --> Parameter

[=] --> auf Kopien der Var nur lesend zugreifen --> arbeite lesend --> Var des Kontext

[&] --> auf Var schreibend zugreifen

Vergleichsfunktion bei Vektoren

Besonderheit --> auf alle Variablen des aufrufenden Kontexts zugreifen

Einzeiliges if: if(a>b):

Return a;

Else

Return b;

Return a>b?a:b;

Fehler immer auf --> Stderr --> cerr

Make:

Man kann die Zielerreichung festlegen

Make ist von der Zeit abhängig --> wenn eine CPP neuer ist als mein Ziel wird diese neu erstellt

Verwendet ein Makefile

Wenn kein Ziel angegeben --> erste Ziel wird verwendet

CMake:

Benötigt CMakeLists.txt

Erstellt ein Makefile

Für Programmiersprache C

Compilieren mit Make

X -> Projektverzeichnis

-->src

-->include

-->build --> cmake .. -> Makefile, make, ./x 1 2 3, eingabe.txt

-->CMakeLists.txt

Dateiverarbeitung:

#include<fstream>

Ofstream --> output stream

.is\_open --> ob die Datei geöffnet werden kann --> Methode

Immer Fehlervearbeitung --> mindestens is.open()

<< --> in Datei schreiben

Myfile.close() --> Datei schließen

Ressourcen die ich öffne muss ich auch wieder schließen

Pfade werden immer von dem Pfad gerechnet, wo man das Programm startet

[**www.cplusplus.com/doc/tutorial/files**](http://www.cplusplus.com/doc/tutorial/files)

Sortieralgorithmen:

Bubble-Sort

Insertion-Sort --> fügt das Element an die richtige Position ein

Selection-Sort --> setzt immer kleinstes Element an die erste Stelle --> pro Element nur eine Vertauschung

**Kommt zur TA**

Laufzeitverhalten --> z.B. Zwei Schleifen -> O(2n) --> O(Problemgröße)

Schleife in Schleife -> O(n2)

Kleiner ist besser

**->** macht den Rückgabewert explizit

Heep:

New --> legt Var. v. Datentyp an Heep an

Delete --> löscht diese Var.

New normal

Delete normal

New []

Delete []