1 Aufbau eines Programmes

```
#include <iostream>
#include <vector>
#include <cmath>
#include <time.h>   // Zeitmessung
#include "headerfile.h"
#define N 10  // defines

//structs, functions, enums
int main(void)
{
//programm code
return 0;
}
```

2 Variablen

char	1 byte 8 bits.
char16_t	At least 16 bits
char32_t	At least 32 bits.
signed char	Min 8 bits.
signed short int	Min 16 bits.
signed int	Min 16 bits.
signed long int	Min 32 bits.
signed long long int	Min 64 bits.
unsigned char	Min 8 bits.
ßhort int	Min 16 bits.
ïnt	Min 16 bits.
"long int	Min 32 bits.
"long long int	Min 64 bits.

2.1 Variablennamen

Keine Leerzeichen, Satzzeichen oder _ Symbole Keine Zahl oder am Anfang case sensitivity – Gross - Kleinschreibung beachten

2.2 Einfache Variablen deklarieren

```
int a,a2;   int b (1);
int b = 10;  int b {1};
float c = a*b - 0.5;
```

2.3 Casts

Änderung einer Variable in einen anderen Type

```
double a = 1.5; int b;
b = int (a);
b = (int) a; // b=1
7/2 = 3 , 7/(double)2 = 7/2.0 = 3.5
double(7/2) = 3.0 , int(19/10.0) = 1
```

2.4 Enum

Enum ist ein Aufzählungstyp. Die Konstanten aus der Enum kann man im Programm verwenden.

```
enum farbe {ROT, BLAU, GELB};
farbe f = ROT;
if(f != BLAU) { };
```

2.5 Hexadezimaler Code & Adressen

0,1,...,9,A,B,C,D,E,F (hex) anstelle von 0,1,...,14,15,16 (dec) Adressen werden hexadezimal angegeben. $a, a+1, a+2, a+3, ...$

int,float(4byte)	double (8byte)
0x22ff70	0x22ff70
0x22ff74	0x22ff78
0x22ff78	0x22ff80

2.6 Fliesskommazahlen

float	1b→sign, 8b→exp, 23b→mantisse Wert $= (-1)^S \cdot 2^{(E-127)} \cdot (1.F)$ Bsp: $0.125 = 2^3 \Rightarrow S \rightarrow 0, E \rightarrow 124, F \rightarrow 0$	
	0 | 01111100 | 00000000...0	$= 0.125$
	0 | 01111111 | 00000000...0	$= 1$
	1 | 01111111 | 11000000...0	$= -1.75$
	0 | 00000000 | 00000000...0	$= 0$
	0 | 11111111 | 00000000...0	$= +\text{infty}$
	0 | 00000000 | 00000000...0	$= \text{NaN}$
double	1b→sign, 11b→exp, 52b→mantisse Wert $= (-1)^S \cdot 2^{(E-1023)} \cdot (1.F)$	

3 Operatoren

$+ - * / \hat{}$	mathematische Operatoren
%	Modulo
x += i;	x = x + i; ebenso *=, /=, -=
$1.1E^{-5}$	$= 1.1 \cdot 10^{-5}$
i++, i--	erhöt / verkleinert i um 1

b=5; c=b++;→c=5,b=6 verwende ++b für c=6,b=6
Für weitere mathematische Funktionen

```
#include <cmath>
fabs(), sqrt(), exp(), log(), cos(),
acos()
```

4 Logische Konstrukte

<, <=, >, >=	grösser, grössergleich, kleiner
||	oder
&&	und
==	gleichheit
!= ungleich	
!	nicht

5 iostream

```
using namespace std;
cout << "a =" << endl; //Ausgabe
cin >> a; //Eingabe
"\n" //Zeilenende "\t" //tabulator
"\"" //Anführungszeichen
```

6 Umrechnung Binär-Dezimal

13	1	Dezimalzahl durch 2 teilen und rest
6	0	notieren. Bits von unten nach oben
3	1	lesen.
1	1	6 bsp: 13 = 1101
0		

$1001 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 0 + 0 + 1 = 9$

7 Kontrollstrukturen

if()

```
if(a==10){
b=15;
}
else if(a==11) b=14;
else b=10;
// oder kurz
a==10?b=15:a==11?b=14:b=10;
```

for()

```
for(int i=0; i<10; i++)
{
a=a+i;
} // abbruch mit break;
```

while()

```
while(b<20){
b++;
}
```

oder

```
do {
b--; //min 1 x
}
while(b!=15);
```

abbrechen mit break (immer nur die innere Schleife) Überspringen des Rests des Rumpfes zur nächsten Auswärtung mit continue;

switch case()

```
switch(a) {
case 15:cout<<"a=15";break;
case 14:cout<<"a=14";break; //a==14
default:cout<<"a!=15,a!=14"; //else
}
```

Äquivalente Strukturen Man kann verschiedene Strukturen verwenden um ein und dasselbe auszudrücken:

```
int i=0;
do {
i=i+1;
if (i==10) break;
}while(true); //aka immer
```

...ist äquivalent zu...

```
for(int i=0;i!=10;i++){}
```

Endlosschleifen Man kann verschiedene Strukturen verwenden um ein und dasselbe auszudrücken:

```
int i=10;
do {
i=i+1;
if (i==10) break;
}while(true);
for(int i=3;i!=20;i=(i+3)%300)
int i=99;
while(i>10){
i--;
if(i==15) i*=6;}
```

8 Arrays

9 Structures

9.1 Pointer und Referenzen als Rückgabewert und Parameterübergabe

10 Funktionen

Funktionen sind Unterprogramme, die häufig verwendeten Code enthalten.
Ein Beispiel:

```
int add (int a, int b);   //Prototyp

//PRE: a, b > 0
//POST: true, wenn eine das doppelte
  der anderen ist
bool timestwo (int a, int b){
  bool c=false;
  return a==add(b, b) || b==add(a,a);
}
```

Rückgabewert ist immer genau **ein** Variabeltyp (Workaround: Structs). Ohne Rückgabewert schreibt man `void`.

10.1 Aufbau

```
rückgabewert funktionsname (argument){
  funktionskörper
  return ;
}
```

10.2 Pre- und postconditions

Preconditions beschreiben den Input der Funktion, Postcondition den Output und die Wirkung der Funktion. Preconditions prüft man mit
`assert (a>0 && b>0)`

10.3 Prototyp und Gültigkeitsbereiche

Falls eine Funktion `g`, die Funktion `f` benötigt muss diese vorab definiert sein, da sich der Gültigkeitsbereich einer Funktion nur unterhalb seiner Defintion befindet. Die formalen Argumente verhalten sich wie Variabeln und haben nur einen Lokalen Gültigkeitsbereich im Funktionsblock.

10.4 Rekursion

Wenn eine Funktion sich selber wieder auruft, nennt man das Rekursion. Dabei muss es eine Abbruchbedingung geben, die auch erreicht wird. Dann wird von innen aufgelöst.

```
int fak (int n){
  if(n==1) return 1;
  return n* fak(n-1);
}
```

11 Pointer und Referenzen

Bei Variablenübergabe (call by value) werden Kopien übergeben, welche nicht verändert werden können.
Bei Referenzübergabe (call by reference) kann die Subroutine die Werte bleibend verändern.
Objekte einer Klasse und Strukturvariablen sollen immer by reference übergeben werden!

11.1 call by reference

statisch:

```
void swap(int&
a, int& b){
  int tmp = a;
  a = b;
  b = tmp;
}
int main(){
  int x = 4;
  int y = 3;
  swap(x, y);//
   OK!
  return 0;
}
```

dynamisch:

```
void swap(int*
a, int* b){
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
int main(){
  int x = 4;
  int y = 3;
  swap(&x, &y);
// OK!
  return 0;
}
```

11.2 call by value

```
void swap(int a, int b){
  int tmp = a;
  a = b;
  b = tmp;
}
int main(){
  int x = 4;
  int y = 3;
  swap(x, y); // keine Auswirkung
  return 0;
}
```

11.3 return by reference

```
int& inc(int& i){
  return ++i;
}
```

Der Funktionsaufruf ist nun selbst ein L-Wert, was nun Ausdrücke wie `inc(inc(x))` oder `++inc(x)` erlaubt. **Achtung** Gültigkeitsbereiche: Return by reference auf lokale Variable ist undefined behavior. //**Edit sobald Pointer in Vorlesung**

12 Vektoren

Vektoren dienen zum Speichern gleichartiger Daten.

12.1 Initialisierung

```
std::vector<int> vec(3);
//{0, 0, 0}
std::vector<int> vec(4, 2);
//{2, 2, 2, 2}
std::vector<int> vec{4,3,2,1};
//{4, 3, 2, 1}
std::vector<int> vec;
//leerer Vektor
```

12.2 Zugriff

Das erste Element eines Vekotrs hat index 0. Ein Zugriff auf Elemente ausserhalb der gültigen Grenze führt zu undefinierten Verhalten. C++ bietet eine optionale überprüfung.

```
std::vector<int> vec(3);
vec.at(3) = 1;  //Error compiler
vec[3] = 1; //undefined behaviour
```

12.3 Befehle

```
std::vector<int> vec{0,1};
vec.size(); //Länge des Vektors: 2
vec.push_back(3) //hängt wert an:
  {0,1,3}
vec.clear(); //löscht Inhalt : {0,0,0}
vec.resize(2); //ändert Grösse: {0,0}
```