1 Aufbau eines Programmes

```
#include <iostream> // Standart In-/
    Output stream
#include <vector>   // Vector library
#include <cmath>    // Für math.
    funktionen
#include <time.h>   // Zeitmessung
#include "headerfile.h" // Einbiden
    Headerfile
#define N 10  // defines jeglicher art

//structs, functions, enums

int main(void)
{
//programm code
return 0;
}
```

2 Variablen

Group	Type names	Notes on size / precisio
Character types	char	One byte 8 bits.
	char16_t	At least 16 bits
	char32_t	At least 32 bits.
	wchar_t	Largest character set
Integer types (signed)	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.
Integer types (unsigned)	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.

Mögliche Initialsisations von vaiablen

```
int x;
int x = 1;
int x (1);
int x {1};
```

2.1 Pointer und Referenzen als Rückgabewert und Parameterübergabe

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!

2.2 call by reference

```
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;
}
```

```
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;
}
```

2.3 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 Ausw.
return 0;
}
```

3 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.

3.1 Aufbau

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

3.2 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)`

3.3 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.

3.4 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);
}
```

4 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!

4.1 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;
}
```

4.2 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;
}
```

4.3 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**

5 Vektoren

Vektoren dienen zum Speichern gleichartiger Daten.

5.1 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
```

5.2 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
```

5.3 Anwendungsmöglichkeiten

Einige Funktionen aus der `vector` Bibliothek:

```
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}
vec.insert(1,3); //fügt Wert ein:
  {0,3,0}
```

5.4 Multidimensionale Vektoren

Eine Matrix (2. dimensionaler Vektor) ist ein Vektor, dessen Einträge Vektoren sind.

```
std::vector<std::vector<int>> mat = {
{00,01,02},
{10,11,12},
{20,21,22}};
std::cout<<mat[1][2]; //Output: 12
```

Wichtig: Arrays werden fast immer per Referenz übergeben.

6 Strings

Strings sind Arrys/Vektoren vom typ char. Mit Strings speichert man folglich längere Zeichenketten und benötigt `#include<string>`. Dank überladener Operatoren haben Strings einige Zusatzfunktionen zu Vektoren.

6.1 Initialisierung / Funktionen

```
std::string text(3, 'u');   // {u, u,
  u}
std::string name = "Cedric";
name += " Renda";
std::cout<< name = "Robin von Reding";
   //false
std::cout<< name = "Cedric Renda"; //
  true
```

6.2 ASCII-Tabelle

Werte vom Typ `int` und `char` lassen sich einfach konvertieren.

```
int i=97;
char c=i;
std::cout<<c; //Output: a
c = 'A';
i = c;
std::cout<<i; //Output: A
```

Der Compiler geht dabei nach folgender Tabelle vor.

0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BBS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SOH	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

00-31:	NUL, ...	32:	SPACE
48-57:	0-9	65-90:	A-Z
97-122:	a-z	127:	DEL