Clicker

Bitte den Link

https://vc2.sonia.de/b/har-2cy-qhv-bj0

für WS 2025/26 nutzen.

Termine des Semesters

	Termin		Vorlesung	Übungen und Feedback
Vorles	Woche		Freitag ; Block 1+2	
6	8	14. Nov	Operatoren; Templates, Polymorphie	14.11; Teambildung abgeschlossen
7	9	21. Nov	STL, Iteratoren;	
8	10	28. Nov	lineare und assoziative Container; STL Algorithmen	Bis 27.11 Erste Teamaufgabe abgegeben; Demo per BBB vorher
9	11	05. Dez	Algorithmus oder Methode Klasse unique_ptr,, shared_ptr, Lambda-Ausdrücke	
10	12	12. Dez	Verschiebeoperatoren ,	
11	13	19. Dez	Rest; Vorbereitung Klausur	Finale Abgabe Teamaufgabe Mi. 17.12; Demo per BBB vorher

Klausur: Mi, 14.1.26; 11:00 – 12:30 Uhr; R. 252
Klausureinsicht: Jan 2026 xxx Uhr

Rückblick

Wiederholung, insbesondere Aufgaben aus dem einem Test für vorherige Semester

Klasse string
Klasse vector<T>, Klasse array<T,size>

Referenz oder Zeiger oder Werte oder doch was anderes?

Klassen

- Am Beispiel von Vektor, Konstruktor, Destruktor
- Konstruktoren
- +Z, -Z
- LogTrace / FunctionLog
- Tiefe- und flache Kopien; Kopierkonstruktor, Zuweisungsoperator

Clicker

```
typedef int*  IntPointer;
int i = 32;
IntPointer p1 = &i;
int* p2 = new int[2];
int** pp2 = new IntPointer[2];
int** pp1 = &p1;
(p2)[1] = 138;
pp2[1] = &(p2[1]); /*1*/
```

Welche der 4 folgenden Möglichkeiten folgt nach /* 1 */?

1. *pp1 = 140;
2. **pp1 = 140;
3. &pp1 = 140
4. pp1[0] = 140;

Geben Sie den Code syntaktisch richtig an, um über die Variable pp1 die Variable i auf dem Wert 140 zu setzen.

Aufgabe

```
void setVar(int* erg, int w) {
    int quad = w * w;
    ??? = quad; /*1*/
}
void funk3(void) {
    int i = 3;
    int res = 14;
    setVar(??? , i); /*2*/ /*res uebergeben */
    datei << "res= " << res;
}
```

Ausgabe soll 9 sein.
Was wird also jeweils für „???“ eingesetzt?

1. erg = quad; // setVar(&res, i);
2. *erg = quad; // setVar(&res, i);
3. *erg = quad; // setVar(res, i);
4. &erg = quad; // setVar(*res, i);
5. erg = quad; // setVar(res, i);

Überladen von Funktionen

```
int Add(int x, int y)
{
  cout << "Add(int,int)" << endl;
  return x+y;
}
int Add(int x, int y, int z)
{
  cout << "Add(int,int,int)" << endl;
  return x+y+z;
}
double Add(double x, double y)
{
  cout << "Add(double,double)" << endl;
  return x+y;
}
```

```
int main()
{
  cout << Add(1,2) << endl;
  cout << Add(1,2,3) << endl;
  cout << Add(1.0,2.0) << endl;
}
```

```
Add(int,int)
3
Add(int,int,int)
6
Add(double,double)
3
```

Überladen von Funktionen

```
int Add(int x, int y)
{
  cout << "Add(int,int)" << endl;
  return x+y;
}
int Add(int x, int y, int z)
{
  cout << "Add(int,int,int)" << endl;
  return x+y+z;
}
double Add(double x, double y)
{
  cout << "Add(double,double)" << endl;
  return x+y;
}
```

```
int main()
{
  cout << Add(1,2) << endl;
  cout << Add(1,2,3) << endl;
  cout << Add(1.1,2.2) << endl;
}
```

```
1:    Add(int,int)
3
2:    Add(int,int,int)
6
3:    Add(double,double)
3.3
```

Überladen von Funktionen

độụčľê□Ňụľţ□îŋţ□□ắ□□îŋţ□□č□□□
□□□çộụţ□□□□□Ňụľţ¸□□□
□□□sêţʃụsŋ□ắ□□□č□
□
îŋţ□ņắîŋ□□□□
□□□çộụţ□□□□□Ňụľţ□¸□□¸□□□□□êŋđľ□
□□□sêţʃụsŋ□¸□
□

```
¸□    Ňụľţ¸□¸
¸□    Ňụľţ¸□¸
¸□    Ňụľţ¸□¸
¸□    Ňụľţ¸□_
_□    Şỳŋţẵẏğêḥľês
```

Überladen von Funktionen

```
độụčľê□Ňụľţ□îŋţ□□ắ□□îŋţ□□č□□□
□□□çộụţ□□□□□Ňụľţ¸□□□
□□□sêţụsŋ□ắ□□□č□
□
độụčľê□Ňụľţ□độụčľê□ắ□□độụčľê□č□□□
□□□çộụţ□□□□□Ňụľţ¸□□□
□□□sêţụsŋ□ắ□□□č□
□
îŋţ□ņắîŋ□□□□□
□□□çộụţ□□□□□Ňụľţ□¸□□¸□□□□□êŋđľ□
□□□sêţụsŋ□¸□
□
```

```
¸□    Ňụľţ¸□¸
¸□    Ňụľţ¸□¸
˛□    Ňụľţ¸□¸□¸
_□    Şỳŋţắẏğêḥľês
```

Wie sieht der Aufruf aus?

```
void fzeiger(Vorgang*  fp) { … }
void fref(Vorgang&  fr) { … }
void fwert(Vorgang   fw) { … }
```

```
Vorgang v;
```

```
fref(& v);  // 1
fref(   v);  // 2
fref(*  v);  // 3
   //4 keine Ahnung
```

Syntax / Semantik

Wert und Referenz (&) sind das gleiche (wenn es rein um die Syntax geht). Semantisch sind sie etwas völlig anderes

Ausgangstyp	Zieltyp	Operator/Zeichen
Wert/Referenz	Zeiger	&
Zeiger	Wert / Referenz	*
Sonst		Kein Operator, Passt schon

Zeiger und Referenz führen beide zum gleichen Assemblercode, aber mit unterschiedlicher Syntax im C++-Code. Deshalb sind beide im Sinne von Ausgangsparameter verwendbar. Das Original-Objekt wird manipuliert.

Ein Vorschlag für Namenskonventionen

1. Präfix:

Globale Variablen beginnen mit einem "g":

```
int g_zahl;
```

Funktionsargumente beginnen mit "a"

```
funk(double ad_durch,  const int ai_wert)
```

Member (Elemente) in Klassen mit "m"

```
class Mitarbeiter {...
double m_umsatz;
}
```

Lokale Variablen erhalten keinen ersten Präfix.

Ein Vorschlag für Namenskonventionen (2)

Der zweite Präfix gibt an, ob es sich um eine Referenz (r) oder um einen Zeiger (p) handelt.

```
int * p_anzahl;
funk( double & ard_durch, int * apn_zahlen)
```

Der dritte Präfix legt den Typ fest:

- i für int, s für short
- n für einen ganzzahligen Typ
- d für double, f für float
- c für eine Klasseninstanz
- t für einen Typ, der durch typedef definiert wurde
- ...

```
funk (Stack & arc_stack,  float *apf_summe)
int i_anzahl;
```

Vorlesungsplanung

- Wiederholung mit Referenz oder Zeiger oder Werte oder doch was anderes?

- MacheNix als Übung für Sie (Kopierkonstruktor und Zuweisungsoperator, selber „Hand anlegen“)
- Operatoren