Clicker

Bitte den Link

https://vc2.sonia.de/b/har-g4r-32s-nc6

nutzen.

Termine des Semesters

Fr 20. Oktober Betreuung, Mo 27.11 DLR Besuch

Termin				Vorlesung	Bewertete Übungen
Vorles	Woche			Montag bzw. Freitag ; Block 1+2	
8	6	23. Okt	Mo	tiefe und flache Kopie: Kopierkonstruktor, LogTrace	tiefe, flache Kopie, minimale Std-Schnittstelle;
	7	30. Okt	Mo	Operatoren, Templates	Abgabe So 5.11
9	8	06. Nov	Mo	Verschiebeoperatoren, Templates	
11	9	13. Nov	Mo	STL, Iteratoren	
12	10	20. Nov	Mo	Algorithmus versus Methode, C	
13	10	24. Nov	Fr		Unterstützung bei Übungen, sofern erforderlich
	11	27. Nov	Mo	DLR-Besuch	
14	12	04. Dez	Mo	Intelligente Zeiger, Klasse unique_ptr,, shared_ptr, Lambda-Ausdrücke	Wettbewerb Teil 1
15	13	11. Dez	Mo	Vererbung, Polymorphi	

Klausur: 4.1.24; 11 Uhr; WF-EX-2/127
Klausureinsicht: 19.1.2024 xxx Uhr

Ab 6.11 Unterstützung durch Herrn Schmidt

Aufgabe 03; Zweite Abgabe

7. Vorlesung; Mo. 23.11.2023 / 6. Woche

Vorlesung

Wiederholung / Ankündigung (31.10.2022) tiefe und flache Kopie (31.10.2022)
Weitere Informationen zu Konstrukturen (Selbststudium als Clickeraufgaben) (31.10.2022)

Übungsaufgaben WS 2023/24; Sprechfunk-Annotation, keine direkte Bewertung; Voraussetzung für Aufgabe 03, So 05.11.2023, 23:59 Uhr über SVN

Exercise: Classes and deep and shallow copy (20.10.2023)

Übungsaufgaben

Infos zu cmake (29.09.2023)
DynVorgangsArray mit Problemen VS 2022 bzw. CMake (31.10.2022)
Vektor mit Kopier-Problemen VS 2022 bzw. CMake (31.10.2022)

Rückblick

Wiederholung, insbesondere Aufgaben aus dem Test WS 2019/2020

Klassen beginnen

- Am Beispiel von Vektor, Konstruktor, Destruktor
- Selber programmiert als Übung
- Konstruktoren
- +Z, -Z begonnen

Clicker

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

```
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*/
```

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

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.

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 Test(int& a, int& b){
   datei << "Test1" << "\n";
   return a + b;
}
int Test(int a, int b, int c){
   datei << "Test2" << "\n";
   return a + b + c;
}
```

```
double Test(int& a, double b = 1.0){
   datei << "Test3" << "\n";
   return a + b;
}
void TestCall(){
   int i = 11; double d = 19.3;
   Test(i, i);
   d = Test(i, i);
   i = Test(i);
   Test(i, i, i);
}
```

1. Test1 Test1 Test3 Test2
2. Test1 Test3 Test3 Test2
3. Test1 Test3 Test1 Test2
4. Test1 Test1 Test1 Test2

Noch mal ganz langsam (6)

```
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.

Noch mal ganz langsam (6)

```
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
```

fr ist Ausgangsparameter der Funktion fref (&).
fr ist eine Referenz, d.h. die Adresse eines Vorgangs.
Die Umwandlung eines Vorgangs in eine Adresse übernimmt jedoch der Compiler. Deshalb ist ein ganzer Vorgang zu übergeben,, d.h. //2 ist richtig.

Ergebnis:

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

HHe1 Hartmut Helmke; 22.10.2021

Clicker-“Abstimmung“

```
void f(Vorgang* fp,  Vorgang fv)
{ … }
```

```
int main() {
   Vorgang* pv = new Vorgang();
   Vorgang v;

1.       f(pv, &v);
2.       f(&v, *pv);
3.       f(v, pv);
4.       f(**v, &pv);
```

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;
```

Besuch im DLR von Interesse?

Termin Montag, 27.11.2023 im Rahmen der Vorlesung.
Eintreffen zwischen 08:30 und 09:00
Ende 11 Uhr
Genauer Termin wird noch festgelegt, ggf. abhängig von ÖPNV

Interesse?
1. Werde teilnehmen
2. Wahrscheinlich
3. Eher nicht
4. Nein
5. Weiß nicht

Vorlesungsplanung

Block 1

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

- Aufgabe 03
- +Z, - Z: Wie man sich erklären kann, wann Konstruktor und Destruktor aufgerufen werden; ab Folie 22

- MacheNix
- Kopierkonstruktur, Zuweisungsoperator (tiefe + flache Kopie)
- Debugging mit FunctionLog

Block 2

- Kopierkonstruktor und Zuweisungsoperator, selber „Hand anlegen“