Klausur [Modul] C++
Jahr: 2002; Dozent: Dipl.Ing. Sorber

Aufgabe 1: (3P)

Bestimmen Sie den Wert der jeweils aufgeführten Variablen nach der Ausführung des folgenden Quelltextes:

```
int m, *p, *q, n;
m = 45;
p = &m;
int &r = m;
n = (*p)++;
q = p-1;
r = *(--p) +1;
```

a) m
b) n
c) &m
d) *p
e) r
f) *q

Lösung:

a) m = r = *q + 1
b) n = 45
c) &m = Adresse von m
d) 45 (p=q)
e) r=*q+1
f) *q zeigt auf uninitialisierten Speicherbereich

Aufgabe 2: (4P)

Gegeben ist folgende Struktur:

```
struct bsp
{
  int x;
  double y;
  bsp * next;
};
```

Es ist dynamisch Speicher für ein Datenobjekt vom Typ der geg. Struktur bereitzustellen. Den Member x und y dieses Datenobjekts sollen die Werte 10 und 12.5 zugewiesen werden. Ein weiteres Datenobjekt vom Typ der Struktur soll über den Zeiger der Struktur erzeugt werden (Verkettung). Den Member x und y dieses zweiten Datenobjekts sollen die Werte 11 und 13.8 zugewiesen werden. Anschließend sollen die dynamisch erzeugten Datenobjekte freigegeben werden. Schreiben Sie dazu die entsprechenden Anweisungen.

Lösung:

```
bsp *b = new bsp;
b -> x = 10;
b -> y = 12.5;

bsp *c = new bsp;
b -> next = c;
c -> x = 11;
c -> y = 13.8;

delete b;
delete c;
```

Aufgabe 3: (6P)

Gegeben sei folgendes Programmfragment:

```
void Infile (const char* dateiname, ifstream& eindat)
{
  eindat.open (dateiname);
  if (! eindat)
  {
    cout << "Fehler beim Öffnen der Eingabedatei!";
    exit (1);
   }
}

int main ( )
{
  …
  ifstream eindat;
  Infile ("test.dat", eindat);
  …
  return 0;
}
```

Fügen sie zu diesem Programmfragment Exception-Handling hinzu.

Lösung:

```
class Except
{
        public: Except ( ) { }
};

void Infile (const char* dateiname, ifstream& eindat)
{
  eindat.open (dateiname);
  if (! eindat)
  throw Except ( );
}

int main ( )
{
  …
  ifstream eindat;
  try
  {
    Infile ("test.dat", eindat);
  }
  catch
  {
    cout << " Fehler beim Öffnen der Eingabedatei";
    exit (1);
  }
  return 0;
}
```

Aufgabe 4: (2P)

Gegeben sei folgende Funktion zum Tauschen von int -Werten im Zusammenhang mit einem Sortieralgorithmus:

```
void tausch (int& a, int& b)
{
  int temp = a;
  a = b;
  b = temp;
}
```

Realisieren Sie die Funktion als Template, so dass Typunabhängigkeit für die Funktionsargumente erreicht wird.

Lösung:

```
template <class T> void tausch (T& a, T&b)
{
   T temp = a;
   a = b;
   b = temp;
}
```

Aufgabe 5: (6P)

Schreiben Sie ein Funktionstemplate, dass in einem eindimensionalen Array aus Datenobjekten ein spezielles Datenobjekt sucht und das gefundene Datenobjekt an die aufrufende Funktion zurückgibt. Die Argumentenliste des Funktionstemplates enthalte das Array sowie das gesuchte Datenobjekt. Typ der Datenobjekte sei der Template-Parameter T.
Schreiben Sie auch die Aufrufanweisung in der main-Funktion.

Lösung:

```
template <class T> T suchen (T array [ ], T gesucht)
{
   int i =0;
   T gefunden;
   for (i; i < 100, i++)
   {
      if (gesucht = = array [i]
      gefunden = array [i];
    }
   return gefunden;
}

int main ( )
{
  int feld [50];
  int x;
    …
   x = suchen (feld, 5);
    …
}
```

Aufgabe 6: (2P)

Erläutern Sie die Begriffe frühe Bindung und späte Bindung im Zusammenhang mit dem Polymorphismus. Geben Sie an, wie die späte Bindung programmtechnisch realisiert werden muss.

Lösung:

Frühe Bindung bedeutet, dass bei Polymorphismus zur Übersetzungszeit entschieden wird, welche Methode verwendet werden soll, bei der späten Bindung wird das zur Laufzeit entschieden. Die Entscheidung findet zwischen der Methode der Basisklasse und der überschriebenen Methode der abgeleiteten Klasse statt. Soll die späte Bindung erfolgen, muss vor der Methode der Basisklasse *virtual* stehen.

Aufgabe 7: (2P)

Was verstehen Sie unter einer abstrakten Klasse?
Erläutern Sie, wie programmtechnisch eine Klasse zu einer abstrakten definiert wird.
Geben Sie ein Beispiel!

Lösung:

Eine abstrakte Klasse ist eine Klasse die mindestens eine virtuelle Methode enthält.

```
class Bsp
{
   public:
   virtual void eingeben ( ) = 0;
};
```