

Lösung Klausur 3a, infT

Fragen zur Sprache C++

Korrekte Antworten sind fett markiert. Vormalige Lücken sind mit fettem Text ergänzt.

- Ein Funktionsaufruf ist ein Ausdruck, dessen Typ bestimmt ist durch
 - die an die Funktion übergebenen Argumente.
 - die im Funktionskopf deklarierten Parameter.
 - den Return-Wert der Funktion.**
- In C++ ist eine Standard-Header-Datei
 - eine Objektdatei.
 - eine ausführbare Datei.
 - eine Textdatei.**
- Zur Ausführung der Anweisung `cout << "Hello world!";` (in `main` formuliert) genügt es, die Header-Datei `iostream` ins Programm zu inkludieren.
 - richtig
 - falsch**
- Beim Erzeugen eines Objekts gilt grundsätzlich, dass
 - Speicher für die Datenelemente reserviert wird.**
 - jedes Element mit einem passenden Wert initialisiert wird.**
 - Speicher auf dem Heap alloziert wird.
- Im Anschluss an die Definition `int i = 2, j;` bewirkt die Anweisung `i *= j = 4;`
 - eine Fehlermeldung des Compilers.
 - die Zuweisung von 4 für die Variable `i` und `j`.
 - die Zuweisung von 8 für `i` und 4 für `j`.**
- In C++ ist jeder Vergleich ein Ausdruck vom Typ `int` mit dem Wert 0 ("false") oder 1 ("true").
 - richtig
 - falsch**
- Nach den Definitionen `bool flag; int x = 3, y = 2;` gibt die Anweisung `cout << (flag = x == y);` Folgendes auf dem Bildschirm aus:
 - 0 (oder false)**
 - 1 (oder true)
 - 2
- Wenn eine Source-Datei geändert wird, müssen alle zu einem C++-Programm gehörenden Source-Dateien neu kompiliert werden, damit eine ausführbare Datei erzeugt werden kann.
 - richtig
 - falsch**
- Wenn beim Aufruf einer Funktion weniger Argumente übergeben werden als Parameter im Funktionskopf deklariert sind, werden übrige Parameter mit 0 als Default-Wert initialisiert.
 - richtig
 - falsch**

10. Gegeben seien die folgenden Deklarationen: `void func(double&); double x = 9.7;`. Nach Rückkehr der Programmausführung aus dem Funktionsaufruf wurde der Wert der Variablen möglicherweise geändert.
- richtig**
 - falsch
11. Angenommen, eine Funktion ist mit Default-Argumenten definiert und bei Aufruf wird ein Argument weggelassen. In diesem Fall müssen
- auch die Argumente davor weggelassen werden.
 - auch nachfolgende Argumente weggelassen werden.**
 - nie weitere Argumente weggelassen werden.
12. Die Signatur einer Funktion besteht aus
- dem Namen der Funktion und dem Return-Typ der Funktion.
 - dem Namen der Funktion, der Anzahl und dem Typ der Parameter.**
 - dem ganzen Funktionskopf.
13. Welche der folgenden Anweisungen deklarieren eine überladene Funktion?
- `int func1(double); int func1(double, double);`
 - `double func2(double); int func2(double);`
 - `double func3(double); int func3(double, double = 0);`
14. Der von lokalen, nicht als `static` deklarierten Objekten belegte Speicher wird freigegeben, sobald der entsprechende **Block** verlassen wird.
15. Ein statisches Objekt, das ausserhalb jeder Funktion definiert ist, kann
- überall im Programm
 - überall in derselben Source-File
 - nach seiner Definition in derselben Quelldatei** verwendet werden.
16. Angenommen, die Funktion `myFunc()` ist im Namensraum `mySpace` definiert und wird ausserhalb des Namensraums aufgerufen. Ohne Verwendung einer `using`-Deklaration oder `using`-Direktive wird beim Aufruf dem Namen der Funktion `mySpace::` vorangestellt.
17. Sind im aktuellen Namensraum und in einem mit der `using`-Direktive importierten Namensraum gleiche Bezeichner vorhanden, dann
- gibt der Compiler eine Fehlermeldung aus.
 - führt dies nicht automatisch zu einem Namenskonflikt.**
 - kann es zu Mehrdeutigkeiten kommen, wenn einer dieser Bezeichner angesprochen wird.**
18. Gegeben seien folgende Definitionen: `long z; long& rz = z;`. Dann wird im Ausdruck `++rz;` der Wert von `z` nicht inkrementiert.
- richtig
 - falsch**
19. Gegeben sei die folgende Funktion: `void triple(int& a) { a *= 3; }`. Dann wird beim Aufruf `triple(2);`
- der Compiler eine Fehlermeldung ausgeben.**
 - die Zahl 3 verdoppelt.
 - die Zahl 2 verdreifacht.
20. Gegeben sei der folgende Prototyp `int& func(void);`. Dann wird bei der Anweisung `func() = 7;`
- der Compiler eine Fehlermeldung ausgeben.
 - der Funktion ein neuer Return-Wert zugewiesen.
 - dem durch den Funktionsaufruf referenzierten Objekt ein neuer Wert zugewiesen.**

21. Gegeben sei die folgende Definition: `char ch = 'A';` .
Dann hat der Ausdruck `&ch` den folgenden Typ:
- `char`
 - `char *`
 - `char&`
22. Nach den Deklarationen `void func(double *p1, double *p2); double x = 1.1, y = 2.2;` können Sie die Funktion `func()` wie folgt aufrufen:
- `func(x, y);`
 - `func(&x, &y);`
 - `func(*x, *y);`
23. Speicher für die Dateielemente wird angelegt, wenn
- die Klasse definiert wird.
 - `Y *myY = new Y;` zur Ausführung kommt.
 - die Zeile `Y yourY;` ausgeführt wird.
24. Wenn zwei verschiedene Objekte derselben Klasse angelegt werden, wird der Maschinencode für jede Methode auch zweimal erzeugt.
- richtig
 - falsch**
25. Wenn bei der Definition eines Objekts der Compiler den Konstruktor mit der passenden Signatur nicht finden kann, wird
- ein Objekt angelegt und mit Default-Werten initialisiert.
 - ein Objekt angelegt und nicht initialisiert.
 - kein Objekt angelegt und eine Fehlermeldung ausgegeben.**
26. Beim Aufruf einer Methode wird ein "verstecktes" Argument mit der Adresse des aktuellen Objekts übergeben. Dabei handelt es sich um den this-Zeiger.
27. Wenn eine Funktion eine Referenz oder einen Zeiger auf ein Objekt, das innerhalb der Methode definiert wurde, zurückgibt, muss das Objekt als static deklariert werden.
28. Gegeben sei eine Matrix `net` mit Elementen vom Typ `double`, die zwei Zeilen und drei Spalten besitzt. Geben Sie die Anweisung an, um dem letzten Element in der letzten Zeile den Wert `9.1` zuzuweisen: `net[1][2] = 9.1;`
29. Wenn einer Funktion ein Vektor als Argument übergeben wird, erhält die Funktion
- eine Kopie des Vektors.
 - nur das erste Vektorelement.
 - die Adresse des ersten Vektorelements.**
30. Die Funktion `func()`, die ein Array mit `Employee`-Objekten als Argument erwartet und keinen Return-Wert liefert, weist folgenden Prototyp auf:
- `void func(Employee *e[]);`
 - `void func(Employee e[]);`
 - `void func(Employee e);`
31. Angenommen, es soll eine Funktion `strChr()` definiert werden, die das erste Auftreten eines Zeichens in einem C-String untersucht und einen Zeiger auf das gefundene Zeichen zurückgibt. Dann kann die Funktion folgenden Prototyp aufweisen:
- `char *strChr(const char *s, int c);`
 - `char[] strChar(char s[], int c);`
 - `char strChar(const char *s, int c);`

32. Es sei `x` eine Klasse. Um einen Vektor von Zeigern auf Objekte vom Typ `x` definieren zu können, muss die Klasse einen Default-Konstruktor besitzen.
- richtig
 - falsch**
33. Mit der Definition `long *ptr = new int;`
- wird ein Objekt vom Typ `long` erzeugt.
 - wird ein Objekt vom Typ `int` angelegt und implizit in `long` konvertiert.
 - wird vom Compiler eine Fehlermeldung ausgegeben.**
34. Im Anschluss an die Definitionen `long double z = 1.1, *ptr = &z;` kann der von `z` belegte Speicher wie folgt freigegeben werden: `delete ptr;`
- richtig
 - falsch**
35. Nach dem Aufruf des `delete`-Operators kann nicht überprüft werden, ob der dynamisch reservierte Speicher freigegeben wurde.
- richtig**
 - falsch
36. Angenommen, der Operator `new` reserviert dynamisch den Speicher für ein Objekt einer Klasse. Dann sorgt er auch dafür, dass
- alle von ihm belegten Bytes mit 0 initialisiert werden.
 - ein passender Konstruktor für das Objekt aufgerufen wird.**
 - der New-Handler aufgerufen wird, wenn kein Speicherplatz verfügbar ist.**
37. Angenommen, es soll eine Klasse `MeasureArr` entwickelt werden, die einen Vektor variabler Länge mit Elementen einer vorgegebenen Klasse `Measure` darstellt. Dann ist bei der Definition der Klasse
- ```
class MeasureArr {
private:
 int n; // Vektorlaenge
 // . . .
};
```
- ein dynamisches Element `msPtr` für den Vektor wie folgt zu deklarieren:
- `Measure msPtr[n];`
  - `Measure *msPtr;`
  - `MeasureArr *msPtr;`
38. Gegeben sei folgender Ausschnitt aus einer Klassendefinition:
- ```
class LongArr {
private:
    long *ptr;
    // . . .
public:
    LongArr(int n); // Erzeuge einen Vektor der Laenge n
    // . . .
};
```
- In der Definition des Konstruktors wird dann mit folgender Anweisung der erforderliche Speicher reserviert: `ptr = new long[n];`
39. Angenommen, zwei verschiedene Zeiger `ptr1` und `ptr2` zeigen auf denselben dynamisch reservierten Speicher. Dann wird mit der Anweisung `delete[] ptr1;` der dynamische Speicher nicht freigegeben, da der Zeiger `ptr2` immer noch dorthin zeigt.
- richtig
 - falsch**