CS205 C/CPP Design Quiz Analysis

Note:

- Quiz has the following types of questions:
    - judgment questions (Give a statement, judge true or false)
    - Single choice/multiple choice questions (Give questions and options and select one or more of the correct items)
    - blank filling questions (Fill in a simple answer to the question)
- This material may be helpful for your class quizzes and final exams, so please keep it well.
- The producer of this material is Maystern.

1. The source code of program is as follows:

```
int main(int argc, char *argv[]) {
    ...
}
```

The source code has been compiled to program ***hello*** , when you run it as follows, what ***argc*** will be ?

```
./hello I LOVE CPP
```

【分析】在使用 `int main(int arc, char *argv[])` 时，数组 ***argv*** 从下标 0 开始依次存储输入的编译参数，而 ***arc*** 则记录输入编译参数的个数，本题中输入参数为 `./hello I LOVE CPP` 有 4 个编译参数，`./hello`、`I`、`LOVE`、`CPP` ，因此 ***argc*** 的值为 4。

【答案】4

2. The following code is the ***prototype/declaration*** of function ***mul*** .

```
int mul(int a, int b) {
    return a * b;
}
```

【分析】***prototype/declaration*** 表示声明，***definition*** 表示定义。上述代码是函数 ***mul*** 的定义而非声明，所以陈述错误。声明应该改写为：

```
int mul(int a, int b);
```

【答案】False

3. The data type of the variable ***PI*** is ***double*** .

```
#define PI 3.14
```

【分析】***PI*** 并不是变量（***variable***），虽然宏定义默认的小数是使用***double*** 类型存储的，所以陈述错误。

【答案】False

4. When a function prototype is declared in the header file you create, you do NOT need to define it in a CPP file.

【分析】进行声明（***prototype***）后，还需要进行定义（***definition***）。使用未定义但已经声明的函数时，程序可以通过编译（***compile***），但是会导致连接错误（***Link Error***），无法生成可执行文件。因此，为了实现程序功能，函数既要有声明又需要有定义，所以陈述错误。

【答案】False

5. IDE (Integrated development environment) is a powerful compiler with many useful features.

【分析】IDE，即集成开发环境，是用于提供程序开发环境的应用程序。集成了代码的编写、分析、编译、调试多种功能。而编译器，是将代码（通常为高级语言）翻译成为另外一种语言（通常为低级语言），便于计算机进行执行的程序。IDE中集成了编译器，因此 IDE 不是编译器，陈述错误。

【答案】False

6. If you get 75 for one of your projects, which situation should most likely be?

A. Finish all tasks almost perfectly

B. Finish all tasks well

C. Correct

D. Finish all tasks

【分析】根据于仕琪老师第一节课程的ppt，project评分有如下等级：

- 90 - 100 : Finish all tasks almost perfectly.
- 80 - 90 : Finish all tasks well.
- 70 - 80 : Finish all tasks.

因此，75 分对应的是 70 - 80 分档，应该属于Finish all tasks 情形，选D。

【答案】D

7. What's the output of the source code?

```
signed char c1 = 127;
signed char c2 = 1;
int csum = c1 + c2;
cout << csum;
```

【分析】在运行程度段时，由于 signed char 是 8 位有符号整数，最大可以表示数字的范围是 $[-128, 127]$。执行 int csum = c1 + c2 语句时，自动转换为 int 类型进行计算，因此会返回正确结果 128，而不会产生溢出。

【答案】128

8. sizeof() is a function and can yield the size in bytes of a type.

```
int int_num = 10;
size_t len = sizeof(int_num);
```

【分析】sizeof 是一个操作符，并不是一个函数，故陈述错误。

【答案】False

9. size_t is an unsigned integer type.

【分析】size_t 是 32 位或 64 位无符号整数，陈述正确。

【答案】True

10. What's the value of the variable ***num1, num2, num3*** ?

```
int num1 = 23 / 4 * 4;
int num2 = 23 / 4 * 4.;
int num3 = 23 / 4. * 4;
```

【分析】首先计算 ***num1*** ，从左到右依次计算，先计算`23 / 4`，由于两个操作数 $23,4$ 都是 `int` 类型，所以执行整数除法运算，答案为 `5`，同理再计算 `5 * 4` 得到答案 20；

然后计算 ***num2*** ，从左到右依次计算，先计算`23 / 4`，由于两个操作数 $23,4$ 都是 `int` 类型，所以执行整数除法运算，答案为 `5`，再计算 `5 * 4.`，操作数`4.`是`float`类型，执行浮点数乘法运算，得到答案 `20.`由于隐式类型转换 `float` 转换为 int类型不产生整数部分的误差，所以得到答案 20；

最后计算 ***num3*** ，从左到右依次计算，先计算`23 / 4.`，由于操作数 4 都是 `float` 类型，所以执行浮点数除法运算，答案为 `5.75f`，再计算 `5.75f * 4`，操作数`5.75f`是`float`类型，执行浮点数乘法运算，得到答案 `23.`由于隐式类型转换 `float` 转换为 int类型不产生整数部分的误差，所以得到答案 23。

【答案】***num1 = 20; num2 = 20; num3 = 23***

11. In the following code, since the variable num is not initialized explicitly, it will be initialized to 0 automatically.

```
int num;
cout << num << endl;
```

【分析】在 $C++$ 中，未初始化的变量将会根据编译器的不同赋随机值，并非一定为 0，所以陈述错误。

【答案】False

12. auto is a placeholder type specifier in C++11. What is the value of the variable ***val*** in the following code?

```
auto val = 2 / 3;
val = 3.14 * 2;
```

【分析】在执行语句`auto val = 2 / 3;`时，由于操作数 $2,3$ 都是 `int` 类型，所以执行整数除法运算，得到结果`(int) 0`，所以变量 ***val*** 的类型为 `int`。在执行语句`val = 3.14 * 2;`时，先计算 `3.14 * 2` 执行浮点数乘法运算，得到结果`6.28f`，赋值时进行隐式类型转换，`float` 转换为 `int` 不产生整数部分的误差，因此执行所有语句后变量 ***val*** 的值为 6。

【答案】6

13. What's the output of the following source code ?

```
int x = 100;
int y = 30;
x += (y -= 10);
cout << x;
```

【分析】执行 `x += (y -= 10);` 时先执行`y -= 10`，执行完毕后 y 的值为 20，作为 `(y -= 10)`的返回值，再执行 `x += 20`，执行完毕后 x 的值为 120。

【答案】120

14. What's the output of the following source code ?

```
int x = 100;
int y = x++;
cout << y;
```

【分析】先执行 `y = x` 的赋值，再执行自增`x++`。执行完毕后，x 的值为 101，y 的值为100。

【答案】100

15. What's the output of the following source code ?

```
int x = 100;
int y = (x = 200);
cout << y;
```

【分析】先执行 `x = 200` 的赋值，将 200 作为`(x = 200)`的返回值，再执行赋值`y = 200`。执行完毕后，x 的值为 200，y 的值为 200。

【答案】200

16. What's the output of the following source code ?

```
int a = 2, b = 0;
if (a || b++) {
    cout << b;
}
```

【分析】`a` 的值为 2，隐式转化为 `bool` 值为 `true`，由于短路效应，`b++` 未被执行，因此输出 b 的值为 0；若将上述代码改为：

```
int a = 2, b = 0;
if (b++ || a) {
    cout << b;
}
```

则由于判断时先执行`b++` (false) 再逻辑或`a` (true)，判断完毕后 b 自增，因此输出 b 的值为 1。

【答案】0

17. What's the output of the following source code ?

```
int i = 0;
for (i = 1; i < 100; i += 2) {
    // something
}
cout << i
```

【分析】循环变量 i 的初始值为 1，每次循环累加 2，最后一次执行循环中内容时，$i = 99$。循环结束时，i 自增 2，输出 101。

【答案】101

18. What's the output of the source code ?

```
int x = 5;
do {
    x = 0;
} while (x);
cout << x;
```

【分析】赋值完毕后直接进入 do-while 循环，赋值x = 0，判断条件 x 为false，因此输出0。

【答案】0

19. The following source code (empty in the parentheses) can be compiled successfully.

```
for () {
    // some lines here
}
```

【分析】for 循环语句必须使用两个;分隔开初始语句、条件语句、累加语句，不能缺漏，因此陈述错误。

【答案】False

20. What is the output of the following code ?

A. Key 4

B. Key 5

C. Undefined key.

```
int num = 5;
switch (num) {
    case '4':
        cout << "Key 4" << endl;
    case '5':
        cout << "Key 5" << endl;
    default:
        cout << "Undefined key."
}
```

【分析】'4' 对应是相关字符，并不是数字，int与char 比较都应该转化为int，所以应该输出"Undefined key."，选择 C。

【答案】C

21. How many lines will be printed ?

A. Compilation error

B. 10

C. 11

D. None of the above

```
for (size_t i = 10; i ⩾ 0; i--)
    cout << "Line " << i << endl;
```

【分析】size_t 是 32 位或 64 位无符号整数，会自然溢出，所以条件i ⩾ 0 永远成立，因此将输出无限行，并非编译错误，10 或者 11行，所以选择 D。

【答案】D

22. What's the output of the source code?

A. 2

B. 4

C. 6

D. 8

E. The source code cannot be compiled without error.

F. Unpredictable result.

```
#include <iostream>
using namespace std;
int main() {
    int idx = 0;
    int numbers[4] = {2, 4, 6, 8};
    idx = -1;
    cout << numbers[idx];
}
```

【分析】由于 C++ 在数组访问下标时，不会对下标进行检查，所以将使得访问未知内容，输出不可预测值。

【答案】F

23. What's the output of the source code?

A. 2

B. 4

C. 6

D. 8

E. The source code cannot be compiled without error.

F. Unpredictable result.

```
#include <iostream>
using namespace std;
int main() {
    int idx = 0;
    int numbers[4] = {2, 4, 6, 8};
    idx = 4;
    cout << numbers[idx];
}
```

【分析】由于 C++ 在数组访问下标时，不会对下标进行检查，数组下标从 0 开始，因此将使得访问未知内容，输出不可预测值。

【答案】F

24. What's the output if you compile the following source code with C++11 standard?

```
char str[16] = {"C++"};
cout << strlen(str) << endl;
```

【分析】使用 C++ 11 标准 `char str[16] = {"C++"};` 是合法的字符数组定义，`str[0] = 'C'`，`str[1] = str[2] = '+'`，`str[3] = '\0'`，执行 `strlen(str)` 计算从字符串开始到 `'\0'` 前一个字符的个数，即 3 个。

【答案】3

25. What's the output if you compile the following source code with C++11 standard?

```
char str {"C++"};
cout << str[1] << endl;
```

【分析】使用 C++ 11 标准 `char str {"C++"};` 是合法的字符数组定义，`str[0] = 'C'`，`str[1] = str[2] = '+'`, `str[3] = '\0'`，下标从0开始，`str[1]` 访问从字符串的第 1 号字符，即 `'e'`。

【答案】 e

26. What's the output?

A. C++

B. No output.

C. Compilation error.

D. Runtime error.

```
char str1[16] = {"C++"};
char str2[16];
str2 = str1;
cout << str2;
```

【分析】使用 C++ 11 标准 `char str1[16] = {"C++"};` 是合法的字符数组定义，`str1[0] = 'C'`，`str1[1] = str1[2] = '+'`, `str1[3] = '\0'`，`char str2[16];` 也是合法的字符数组定义，`str2[0] = '\0'`。字符数组不能使用`str2 = str1` 赋值，因此将会出现编译错误。

【答案】C

27. What's the output?

A. C++

B. No output.

C. Compilation error.

D. Runtime error.

```
string str1 = {"C++"};
string str2;
str2 = str1;
cout << str2;
```

【分析】使用 C++ 11 标准 `string str1 = {"C++"};` 是合法的字符串定义，`str1 = "C++"`，`char str2[16];` 也是合法的字符数组定义，`str2 = ""`。字符串能使用`str2 = str1` 赋值，因此输出 C++。

【答案】A

28. What's the output?

```cpp
#include <iostream>
union twonumbers {
    int n[2];
    double d;
};
using namespace std;
int main () {
    twonumbers tn;
    tn.n[0] = 0;
    tn.n[1] = 0;
    cout << tn.d;
    return 0;
}
```

【分析】根据 union 的特性，`int n[2]` 和 `double d` 共用一个大小为 8 字节的内存空间。`tn.n[0] = 0;` 将前 4 字节的内容清空为0，`tn.n[1] = 0;` 将后 4 字节的内容清空为0。所以整个内存空间被清空为0。所以该内存空间表示的浮点数为 0。

【答案】0

29. The output of the following code is:

```cpp
struct Person {
    bool male;
    int id;
    char label;
}
cout << sizeof(struct Person);
```

【分析】`struct` 在 C++ 中以 4 个字节对齐（1个字节的向后补齐，4 个字节以上必须是按照 4 字节对齐），male 占 1 个字节，id 占 4 个字节，label 占 1 个字节。所以实际消耗内存 4 + 4 + 4 = 12 字节，输出 12。

【答案】12

30. What's the output of the following code?

```cpp
int *numbers = new int[8];
char *pc = (char *) numbers;
*numbers = 0x0A0B0C0D;
cout << (int) pc[3];
```

【分析】`int *numbers = new int[8];` 语句开辟了 8×4 字节的空间用于存储 8 个 int 类型的值。`char *pc = (char *) numbers;` 将 pc 指针指向numbers 的首地址（由于是 char*类型，所以pc[3] 指的是从左往右第 4 个字节的 char 类型的数字）。`*numbers = 0x0A0B0C0D;` 将 numbers 的值设为 16 进制的 0A0B0C0D，即二进制的0000 1010 0000 1011 0000 1100 0000 1101，每 1 个字节（即 8 个比特）为一组，从低位到高位依次是：00001101、00001100、00001011、00001010、00000000 ... 。因此，pc[3] 表示的是 00001010，使用 (int) 进行隐式类型转换为 10。

【答案】10

31. The following source code is correct.

```cpp
int *ptr;
*ptr = 3;
```

【分析】由于 `int *ptr;` 定义一个空指针，并未赋值地址，因此也不能将对应地址指向的元素值置为 3，执行该段代码后会导致段错误。

【答案】False

32. What's the output of the following code on your PC (64 bit OS and CPU)?

```
int numbers[8];
cout << sizeof(numbers) << endl;
```

【分析】`int` 类型占 4 字节，开辟 8 个形成 `numbers` 数组，共占用 32 个字节。

【答案】32

33. What's the output of the following code on your PC (64 bit OS and CPU)?

```
int *numbers = new int[8];
cout << sizeof(numbers) << endl;
```

【分析】`int *numbers = new int[8];` 将开辟 8 个`int` 类型的数组变量的首地址传给 `numbers`，作为地址变量，其类型为 `size_t`，当前系统是 64 bit（即 8 字节），所以`size_t` 所占空间为 8 字节。

【答案】8

34. The following code can be compiled successfuly.

```
double value = 0.0;
const double *p = &value;
value = 2.0;
```

【分析】`const double *p = &value;`指的是无法使用 p 指针更改 `val` 的值，若不通过 p 指针直接修改 `value` 的值，可以修改成功。

【答案】True

35. The following code can be compiled successfuly.

```
double value = 0.0;
double * const p = &value;
p[0] = 2.0;
```

【分析】`double * const p = &value;`指的是无法更改 p 指针指向的地址，`p[0] = 2.0` 等价于 `*p = 2.0`，若通过 p 指针修改 `value` 的值，可以修改成功。

【答案】True

36. The following source code is correct and cannot cause bugs.

```
int *pint = (int *) malloc(8 * sizeof(int));
char * pc = (char *) pint;
pc[8] = 'a';
*(pc + 8) = 'b';
```

【分析】`int *pint = (int *) malloc(8 * sizeof(int));` 开辟了 8 个 `int` 类型变量的空间。`char * pc = (char *) pint;` 使用 `char` 指针，指向 `pint` 所指向的空间处。对 `pc[8]` 和 `*(pc + 8)` 赋值都对应 `pc` 从首地址开始向后第 8 个字节，进行赋值。显然开辟了 8 个 `int` 类型变量的空间就意味着开辟了 8×4 个字节的空间，因此程序可以运行。

【答案】True

37. What's the output of the following code?

```cpp
# include <iostream>
using namespace std;
void foo(float * p) {
    p[0] = 1.0f;
}
int main() {
    float values[4] = {3.0f, 4.0f, 5.0f, 6.0f};
    foo(values + 2);
    cout << *values << " ";
    cout << *values + 2 << " ";
    cout << *(values + 2) << " ";
    cout << values[2] << endl;
    return 0;
}
```

【分析】`foo(float * p)` 函数将 `values + 2` 位置的指针对应的值设为 1.0。因此执行完毕 `foo(values + 2);` 时实际上 `values = {3.0f, 4.0f, 1.0f, 6.0f}`。`*values` 表示首地址对应的值，为3.0；`*values + 2` 表示首地址对应的值加上 2，为 5.0；`*(values + 2)` 为首地址后移 2 位对应位置的值，为 1.0；`values[2]` 为首地址后移 2 位对应位置的值，为 1.0。

【答案】3 5 1 1

38. What's the output of the following code?

```cpp
# include <iostream>
using namespace std;
struct people {
    string name;
    int age;
};
void init(people p) {
    p.name = "No name";
    p.age = 0;
}
int main() {
    people p;
    p.age = -1;
    init(p);
    cout << p.age << endl;
    return 0;
}
```

【分析】结构体直接传入时，会复制一份，并不会对原结构体值进行修改，因此 `p.age = -1` 。

【答案】-1

39. What's the output of the following code?

A. 0

B. -1

C. < Compile Error >

D. Random Value

```cpp
# include <iostream>
using namespace std;
```

```
struct people {
    string name;
    int age;
};
void init(people * p) {
    p.name = "No name";
    p.age = 0;
}
int main() {
    people p;
    p.age = -1;
    init(p);
    cout << p.age << endl;
    return 0;
}
```

【分析】`void init(people * p)` 需要传入指针，而 `init(p);` 传入的是结构体，类型不匹配。

【答案】C

40. What's the output of the following code?

```
# include <iostream>
using namespace std;
struct people {
    string name;
    int age;
};
void init(people * p) {
    p -> name = "No name";
    p -> age = 0;
}
int main() {
    people p;
    p.age = -1;
    init(&p);
    cout << p.age << endl;
    return 0;
}
```

【分析】结构体指针传入时，会对原结构体值进行修改，因此 `p.age = 0` 。

【答案】0

41. What's the output of the following source code?

```
# include <iostream>
using namespace std;
struct people {
    string name;
    int age;
};
void init(people & p) {
    p.name = "No name";
    p.age = 0;
}
```

```
int main() {
    people p;
    p.age = -1;
    init(p);
    cout << p.age << endl;
    return 0;
}
```

【分析】 init(people & p) 中变量 p 是传入数据的 reference，因此会导致传入变量值的变化。

【答案】0

42. What's the output of the following code?

```
# include <iostream>
using namespace std;
float area(float & x) {
    return x * x;
}
int main() {
    float value = 3.0f;
    cout << area(value);
}
```

【分析】area(float & x) 传入的是 value 变量的 reference，是 float 类型的数据，可以乘法返回。

【答案】9

43. What's the output?

```
# include <iostream>
using namespace std;
int area(int &x);
int main() {
    int n = 10;
    area(n);
    cout << n << endl;
}
int area(int &x) {
    return x *= x;
}
```

【分析】area(int & x) 传入的是 n 变量的 reference，所以会改变原始变量的值。并且该程序段采用了声明、调用、定义的方式定义函数，可以编译通过。

【答案】100

44. What's the output of the following code?

```cpp
# include <iostream>
using namespace std;
float area(float & x) {
    x = x * x;
    return x;
}
int main() {
    float value = 3.0f;
    cout << area(value) << endl;
    cout << value << endl;
    return 0;
}
```

【分析】area(int & x) 传入的是 n 变量的 reference，所以会改变原始变量的值。

【答案】9

45. What's the output of the following code?

A. 3

B. 9

C. < Compile Error >

D. < None of the above >

```cpp
# include <iostream>
using namespace std;
float area(float * x) {
    return *x * *x;
}
int main() {
    float value = 3.0f;
    cout << area(value);
    return 0;
}
```

【分析】float area(float * x) 输入的是指针，area(value) 输入的是整数，应该改成 area(&value)

【答案】C

46. What's the output of function ***sum()*** ?

A. 4 or 8

B. 3

C. 12

D. 32

E. < Compile Error >

```cpp
# include <iostream>
using namespace std;
int main() {
    int cookies[8] = {1, 2, 4, 8, 16, 32, 64, 128};
    int n = sum_arr(cookies, 3);
}
int sum_arr(int arr[], int n) {
    cout << sizeof arr;
    return 0;
}
```

【分析】函数定义时，调用前必须有声明，因此此时会编译错误。

【答案】E

47. What's the output of function ***sum()*** ?

    A. 4 or 8

    B. 3

    C. 12

    D. 32

    E. < Compile Error >

```cpp
# include <iostream>
using namespace std;
int sum_arr(int arr[], int n) {
    cout << sizeof arr;
    return 0;
}
int main() {
    int cookies[8] = {1, 2, 4, 8, 16, 32, 64, 128};
    int n = sum_arr(cookies, 3);
}
```

【分析】`int sum_arr(int arr[], int n)` 中的 `int arr[]` 表示 int 类型的数组首地址下标，变量`arr` 为首数组指针。根据电脑的位数时 32 位还是 64 位，其地址所占的字节数分别为 4 字节和 8 字节。所以应该输出 4 或者 8。

【答案】A

48. There is a function template. The specialization is correctly implemented in the following code.

```cpp
template <typename T>
T sum(T x, T y) {
    return x + y;
}
struct Point {
    int x;
    int y;
};
template Point sum<Point>(Point pt1, Point pt2) {
    Point pt;
    pt.x = pt1.x + pt2.x;
    pt.y = pt1.y + pt2.y;
```

```
    return pt;
}
```

【分析】函数模板实例化的写法是 `template Point sum<Point>(Point pt1, Point pt2);`，函数特例化的写法是 `template<> Point sum<Point>(Point pt1, Point pt2){...}` 中间有在 `template` 后是否有 `<>` 的区别。

【答案】False

49. The following declaration correctly defines some default arguments.

```
int harpo(int n = 3, int m, int k = 3);
```

【分析】在 C++ 中给函数参数指定默认值时，必须从右到左赋默认值，即某一个参数的左侧一定全是无默认值，右侧一定全有默认值。显然，上述的函数定义不满足参数默认值的要求。

【答案】False

50. The functions and a function pointer are declared as follows. Which answers are correct?

```
float norm_l1(float x, float y); //declaration
float norm_l2(float x, float y); //declaration
float (*norm_ptr)(float x, float y); //norm_ptr is a function pointer
```

A. `norm_ptr = &norm_l1;`

B. `norm_ptr = norm_l2;`

C. `norm_ptr = norm_l2; norm_ptr(3.0f, 4.0f);`

D. `norm_ptr = &norm_l1; (*norm_ptr)(3.0f, 4.0f);`

E. `norm_ptr = norm_l1; (*norm_ptr)(3.0f, 4.0f);`

【分析】函数指针赋值时 `norm_ptr = &norm_l1;` 和 `norm_ptr = norm_l2;` 是等价的；函数指针进行调用时，`norm_ptr(3.0f, 4.0f);` 和 `(*norm_ptr)(3.0f, 4.0f);` 是等价的。

【答案】A B C D E

51. The following code correctly defines a function template:

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

【分析】函数模板就是这么定义的。

【答案】True

52. Function overloading is that multiple functions share the same function name but different signatures as the two functions below:

```
float foo(float arg);
int foo(double arg);
```

【分析】两个函数可以作为重载，当且仅当参数列表不同，函数名相同。显然上述两个函数是函数重载。

【答案】True

53. The ***this*** pointer points to the object and can be used to invoke a member as in the following code.

```
class Person {
    int num;
    public:
        static int foo() {
            return this→num;
        }
        // other members
};
```

【分析】由于方法 `static int foo()` 是静态成员方法，不与类实例绑定，所以不能使用 ***this*** 指针，陈述错误。

【答案】False

54. What's the output of the following code?

A. No name

B. < No output >

C. < Compilation error >

D. < Runtime error >

```
class Person {
    string name;
};
int main() {
    Person p;
    p.name = "No name";
    cout << p.name;
    return 0;
}
```

【分析】class 中，若不写public 还是 private，默认情况下是 private。

【答案】C

55. What's the output of the following source code?

```
class Hello {
    static int value;
    int num;
    public:
        int sum(int i, int j) {
            return i + j;
        }
};
int main() {
    cout << sizeof(Hello) << endl;
    return 0;
}
```

【分析】在类 Hello 中声明静态类变量 value（不占用内存），定义成员变量 num（占用 4 字节内存），所以共占用 4 字节内存。注意到语句int Hello::value; 出现时，才给静态类变量分配内存。

【答案】4

56. What's the output of the source code?

```
class Hello {
    static int value;
    int num;
    public:
        int sum(int i, int j) {
            return i + j;
        }
        void setValue(int v) {
            value = v;
        }
        int getValue() {
            return value;
        }
};
int main() {
    Hello h1, h2;
    h1.setValue(5);
    cout << h2.getValue() << endl;
    return 0;
}
```

A. 5

B. A random integer

C. Compilation error

D. Link error

【分析】在类外部缺少定义`int Hello::value`，造成连接错误，而编译能通过。

【答案】D

57. A class is declared as follows. Please select correct answers for creating a variable.

```
class Stock{
    public:
        Stock();
        Stock(const std::string & co, long n = 0, double pr = 0.0);
        ~Stock();
        //other members
};
```

A. `Stock st1;`

B. `Stock st2("MSFT", 3, 2.0f);`

C. `Stock st3 = Stock("MSFT", 3, 2.0f);`

【分析】上述三种方法都能根据类，生成对象。

【答案】A B C

58. Class ***Stonewt*** is declared as follows.

```
class Stonewt {
    // some members
    public:
        Stonewt(double lbs);
        Stonewt(int stn, double lbs);
        Stonewt();
        ~Stonewt();
        operator int() const;
        operator double() const;
}
```

Which function will be invoked by the following line of code ?

```
Stonewt wt = 120;
```

【分析】Stonewt wt = 120; 等价于 Stonewt wt(120); 构造函数 Stonewt(double lbs); 被调用。

【答案】Stonewt(double lbs);

59. Which function will be invoked by the following line of code ?

```
wt = 120.0; //wt is an object of type Stonewt
```

【分析】wt 是 Stonewt 类的一个对象，首先将 120.0 隐式类型转换，使用 Stonewt(double lbs); 构造出一个 wt 对象，再进行赋值。

【答案】Stonewt(double lbs);

60. Which function will be invoked by the following line of code ?

```
double f = wt; //wt is an object of type Stonewt
```

【分析】wt 是 Stonewt 类的一个对象，使用 operator double() const; 进行隐式类型转换。

【答案】operator double() const;

61. Which function will be invoked by the following line of code ?

```
Stonewt wt(120);
```

【分析】wt 是 Stonewt 类的一个对象，使用 Stonewt(double lbs); 构造该对象。

【答案】Stonewt(double lbs);

62. A conversion function is defined outside of the declaration of class ***Stonewt*** as follow.

```
Stonewt::operator double() const {
    return pounds;
}
```

【分析】隐式类型转换，使用 operator double() const {...}，在类外部定义该函数，应该使用Stonewt::operator double() const {...}。

【答案】True

63. Assignment operator '=' can be overloaded by a non-member function.

【分析】所有赋值符号，无论参数列表如何，都必须定义为成员函数。

【答案】False

64. We can change operators' precedence by overloading.

【分析】重载不能改变符号的优先级，运算符的优先级别是 C++ 语言设计者设计的。

【答案】False

65. If the friend function is defined as in the following source code, it is a member function in the class.

```
class Time {
    private:
        int hours;
        int minutes;
    public:
        Time();
        Time(int h, int m = 0);
        void AddMin(int m);
        void AddHr(int h);
        void Reset(int h = 0, int m = 0);
        Time operator + (const Time & t) const;
        Time operator - (const Time & t) const;
        Time operator * (double n) const;
        void show() const;
        friend Time operator * (double mult, Time in);
};
```

【分析】友元函数定义在类之外，在类内进行声明，所以不是成员函数。

【答案】False

66. If we define a member function as follows for class Time

```
Time Time::operator*(double mult) const
```

then we can calculate as follows

```
a = 3.3 * b;//a and b are objects of type Time
```

【分析】Time Time::operator*(double mult) const 只定义了 Time * double 类型的运算，而没有定义 double * Time 的运算。

【答案】False

67. operator+() overloads the + operator, and it can only be used for mathematical addition.

【分析】operator+() 中可以定义新运算，不只可以执行 mathematical addition，还可执行其他运算。

【答案】False

68. You can define two constructors as follows for class Person.

```
Person(){...}
Person(int m = 0) {...}
```

【分析】Person(int m = 0) {...} 在定义时，自动生成空参数构造函数，不能同时存在两个空参数构造函数，陈述错误。

【答案】False

69. Please read the following code and choose correct answers:

```
class Person {
    char *name;
    public:
        Person() {
            name = new char [128];
        }
        ~Person() {
            delete name;
        }
};
```

A. The code can be compiled without error.

B. Runtime error.

C. It can cause memory double free problem.

D. It can cause memory leak.

【分析】由于析构函数中仅对字符数组 `name` 的首地址删除内存，并未对后续元素删除内存，虽然能通过编译，但是会由于`double free` 造成内存泄漏而产生运行时错误。

【答案】A B C D

70. For class Person, which of the constructors is a copy constructor?

A. `Person::Person();`

B. `Person::Person(const Person & p);`

C. `Person::Person(int m);`

D. `Person::Person(int m, int n);`

【分析】拷贝构造函数应该是`Person::Person(const Person & p);`，而 A. `Person::Person();` 是空构造函数（默认构造函数），C 和 D 是自定义参数的构造函数。

【答案】B

71. For class Person, which of the constructors is its default constructor?

A. `Person::Person();`

B. `Person::Person(const Person & p);`

C. `Person::Person(int m);`

D. `Person::Person(int m, int n);`

【分析】默认构造函数应该是`Person::Person();`而 B. `Person::Person(const Person & p);`拷贝构造函数，C 和 D 是自定义参数的构造函数。

【答案】A

72. If assignment operator is not defined in class Person, the following code will invoke default assignment operator.

```
p1 = p2 = p3; //p1, p2 and p3 are objects of type Person
```

【分析】赋值符号是 `Person & Person::operator = (const Person & person);`，若未定义编译器会自动生成默认的赋值符号，拷贝所有的非静态成员变量后赋值。

【答案】True

73. If you do not define a default constructor for a class explicitly, then no default constructor for that class.

【分析】若不定义默认构造函数，编译器会自动生成 `Person::Person(){}`的默认构造函数。

【答案】False

74. What is the output?

```
class Animal {
    private:
        int weight;
    public:
        Animal(int w = 0) {
            weight = w;
        }
        void print() {
            cout << weight << endl;
        }
};
class Dog: public Animal {
    public:
        Dog(int w = 0): Animal(w) {}
        void print() {
            cout << "Dog ";
            Animal::print();
        }
        void speak() {
            cout << "wangwang" << endl;
        }
};
int main() {
    Dog dog(5);
    Animal * p = & dog;
    p -> print();
    return 0;
}
```

【分析】`Animal * p = & dog; p -> print();` 编译器在这里调用 `Animal` 的成员函数 `print()`。

【答案】5

75. What is the output?

A. wangwang

B. < Compilation Error >

C. None of above

```
class Animal {
    private:
        int weight;
    public:
        Animal(int w = 0) {
            weight = w;
        }
        void print() {
            cout << weight << endl;
        }
};
class Dog: public Animal {
    public:
        Dog(int w = 0): Animal(w) {}
```

```
        void print() {
            cout << "Dog ";
            Animal::print();
        }
        virtual void speak() {
            cout << "wangwang" << endl;
        }
};
int main() {
    Dog dog(5);
    Animal * p = & dog;
    p -> speak();
    return 0;
}
```

【分析】`Animal * p = & dog; p -> speak();` 编译器在这里调用 `Animal` 的成员函数 `speak`，发现没有，编译错误。

【答案】B

76. What is the output?

```
class Animal {
    private:
        int weight;
    public:
        Animal(int w = 0) {
            weight = w;
        }
        virtual void print() {
            cout << weight << endl;
        }
};
class Dog: public Animal {
    public:
        Dog(int w = 0): Animal(w) {}
        void print() {
            cout << "Dog ";
            Animal::print();
        }
        void speak() {
            cout << "wangwang" << endl;
        }
};
int main() {
    Dog dog(5);
    Animal * p = & dog;
    p -> print();
    return 0;
}
```

【分析】`Animal * p = & dog; p -> speak();` 编译器在这里调用 `Animal` 的成员函数 `print`，是虚函数，所以调用子类中的真实方法 `print()`。

【答案】Dog 5

77. Please choose the right answer(s) for declaring a class template

A. template <class Type> class ClassName{...}

B. template <typename Type> class ClassName{...}

【分析】使用 class 和 typename 完全等价。

【答案】A B

78. Matx and Matx12f are declared in the following figure. Please choose the correct statement(s).

A. Matx is a class tempate.

B. Matx is a tempate class.

C. Matx12f is a class template.

D. Matx12f is a template class.

```
template <typename _Tp, int m, int n> class Matx {
    public:
        enum {
            rows = w,
            cols = n,
            channels = rows * cols,
#ifdef OPENCV_TRAITS_ENABLE_DEPRECATED
            depth = traits::Type<_Tp>:: value,
            type = CV_MAKETYPE(depth, channels),
#endif
            shortdim = (m < n ? m : n)
        };
};
typedef Matx<float, 1, 2> Matx12f;
typedef Matx<double, 1, 2> Matx12d;
typedef Matx<float, 1, 3> Matx13f;
typedef Matx<double, 1, 3> Matx13d;
typedef Matx<float, 1, 4> Matx14f;
typedef Matx<double, 1, 4> Matx14d;
typedef Matx<float, 1, 6> Matx16f;
typedef Matx<double, 1, 6> Matx16d;
```

【分析】模板类（template class）指的是通过类模板产生的类，类模板（class template）指的是一类类的模板，具有通用性。显然 Matx 是模板类，而 Matx12f 是类模板。

【答案】A D

79. What's the output of the following code?

```
#include <iostream>
using namespace std;
template <typename Type>
class Stack {
    private:
        Type items[16];
    public:
        size_t size() {
            return sizeof(items);
        }
};
int main() {
    Stack<float> st;
```

```
    cout << st.size() << endl;
    return 0;
}
```

【分析】float 类型的变量是 4 字节，所以输出 4 * 16 = 64。

【答案】64

80. When an exception is thrown, the program must be terminated.

【分析】C++ 可以使用 throw 和 catch 抛出和捕获异常，进行异常处理，并不意味着程序终止。

【答案】False

81. A try block can be followed by multiple catch blocks.

【分析】try {expression} catch (Exception1 &ex1) {...} catch (Exception1 &ex2) {..} 是合法的 try-catch语句，因此 try 程序块中可以有多个 catch 快。

【答案】True

82. The following source code cannot be compiled successfully.

```
double gmean(double a, double b) {
    if (a < 0 || b < 0)
        throw string("bad arguments");
    return std::sqrt(a * b);
}
int main() {
    try {
        gmean(3, -3);
    }
}
```

【分析】try-catch 语句至少需要使用一个 catch，因此会编译错误。

【答案】True

83. The following code cannot be compiled successfully since 'try' is commented.

```
double gmean(double a, double b) {
    if (a < 0 || b < 0)
        throw string("bad arguments");
    return std::sqrt(a * b);
}
int main() {
    gmean(3, -3);
}
```

【分析】程序中try-catch并不是必须的，程序将被成功编译，但由于抛出异常，会运行时出错，程序终止。

【答案】False