

Contents

1	Common stupidly basic things I and most other people in the class obviously already know but are for reasons beyond my comprehension asked on the exam	2
2	Polymorphism	3
3	Past Exam Questions	7
3.1	2013/14 Question 2	7
3.2	2013/14 Question 3	8
4	2016/17 questions	9
4.1	Question 1a	9
4.2	Question 1b	10
4.3	Question 1c	11
4.4	Question 1d	12
4.5	Question 4	13

1 Common stupidly basic things I and most other people in the class obviously already know but are for reasons beyond my comprehension asked on the exam

- Variables should always be initialised before being used
- Destructors in polymorphic types should be virtual
- Classes with pointer types need a copy constructor

2 Polymorphism

```
1 class ITouhou {
2     public:
3     ITouhou() { cout << "ITouhou (" << name() << ") constructed" << endl; }
4     virtual ~ITouhou() { cout << "ITouhou destructed" << endl; }
5     virtual string name() const { return "ITouhou"; };
6 };
7
8 class Youkai : public virtual ITouhou {
9     public:
10    Youkai() : ITouhou() { cout << "Youkai (" << name() << ") constructed" << endl; }
11    virtual ~Youkai() { cout << "Youkai destructed" << endl; }
12    virtual string name() const { return "Youkai"; };
13 };
14
15 class IFlyable: public virtual ITouhou {
16     public:
17     IFlyable() { cout << "IFlyable (" << name() <<") constructed" << endl; }
18     virtual ~IFlyable() { cout << "IFlyable destructed" << endl; }
19     virtual void fly() { cout << "IFlyable::fly()" << endl; }
20     virtual string name() const { return "IFlyable"; }
21 };
22
23 class Yuuka : public Youkai, public IFlyable {
24     public:
25     Yuuka() : Youkai(), IFlyable()
26     { cout << "Yuuka " << name() << " constructed" << endl; }
27     virtual ~Yuuka() { cout << "Yuuka destructed" << endl; }
28     virtual string name() const { return "Yuuka"; }
29     virtual void fly() { cout << "Yuuka::fly()" << endl; }
30 };
31
32 class Reimu : public virtual ITouhou, public IFlyable
33 {
34     public:
35     Reimu() : ITouhou(), IFlyable()
36     { cout << "Reimu (" << name() << ") constructed" << endl; }
37     virtual ~Reimu() { cout << "Reimu destructed" << endl; }
38     virtual string name() const { return "Reimu"; }
39 };
```

Listing 1: Polymorphism example classes

```
1 int main() {
2     ITouhou *y1 = new Yuuka();
3     delete y1;
4
5     cout << "=====" << endl;
6
7     Youkai *y2 = new Yuuka();
8     delete y2;
9
10    cout << "=====" << endl;
11
12    Yuuka *y3 = new Yuuka();
13    y3->fly();
14    delete y3;
15
16    cout << "=====" << endl;
17
18    IFlyable *y4 = new Yuuka();
19    y4->fly();
20    delete y4;
21
22    cout << "=====" << endl;
23
24    IFlyable *r1 = new Reimu();
25    r1->fly();
26    delete r1;
27
28    return 0;
29 }
```

Listing 2: Polymorphism example main()

```
1 ITouhou (ITouhou) constructed
2 Youkai (Youkai) constructed
3 IFlyable (IFlyable) constructed
4 Yuuka Yuuka constructed
5 Yuuka destructed
6 IFlyable destructed
7 Youkai destructed
8 ITouhou destructed
9 =====
10 ITouhou (ITouhou) constructed
11 Youkai (Youkai) constructed
12 IFlyable (IFlyable) constructed
13 Yuuka Yuuka constructed
14 Yuuka destructed
15 IFlyable destructed
16 Youkai destructed
17 ITouhou destructed
18 =====
19 ITouhou (ITouhou) constructed
20 Youkai (Youkai) constructed
21 IFlyable (IFlyable) constructed
22 Yuuka Yuuka constructed
23 Yuuka::fly()
24 Yuuka destructed
25 IFlyable destructed
26 Youkai destructed
27 ITouhou destructed
28 =====
29 ITouhou (ITouhou) constructed
30 Youkai (Youkai) constructed
31 IFlyable (IFlyable) constructed
32 Yuuka Yuuka constructed
33 Yuuka::fly()
34 Yuuka destructed
35 IFlyable destructed
36 Youkai destructed
37 ITouhou destructed
38 =====
39 ITouhou (ITouhou) constructed
40 IFlyable (IFlyable) constructed
41 Reimu (Reimu) constructed
42 IFlyable::fly()
43 Reimu destructed
44 IFlyable destructed
45 ITouhou destructed
```

Listing 3: Polymorphism example output

Notes:

- The `virtual` keyword is required for polymorphic functions, it allows the vtable to be created which is used to select the implementation of a `virtual` function to be executed
- The closest implementation of a `virtual` function to the type of the instance will be called
- The implementation of a function not declared `virtual` will be that of the type (not the type of the instance)
- For this reason destructors should always be `virtual`
- Virtual inheritance is used to avoid the "diamond pattern" problem when a class inherits from multiple children of a single base class
- Without virtual inheritance this results in multiple copies of the base class being created

3 Past Exam Questions

3.1 2013/14 Question 2

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int list[10];
8      int *p;
9
10     p = list;
11     for (int x = 1; x <= 10; x++)
12     {
13         *(p) = x; // The original line would have caused an invalid memory access
14             // (element before start of array)
15         p++;
16         cout << &p << endl; // This outputs the address of the pointer variable p
17     }
18
19     for (int i = 0; i < 10; i++)
20         cout << list[i] << endl;
21
22     return 0;
23 }
```

Listing 4: Sample code

```
1  0x7ffe4c3b3b68
2  0x7ffe4c3b3b68
3  0x7ffe4c3b3b68
4  0x7ffe4c3b3b68
5  0x7ffe4c3b3b68
6  0x7ffe4c3b3b68
7  0x7ffe4c3b3b68
8  0x7ffe4c3b3b68
9  0x7ffe4c3b3b68
10 0x7ffe4c3b3b68
11 1
12 2
13 3
14 4
15 5
16 6
17 7
18 8
19 9
20 10
```

Listing 5: Output

3.2 2013/14 Question 3

```
1  #include <iostream>
2
3  class TwlvTime
4  {
5      public:
6          TwlvTime(int hours, int seconds)
7              : hours(hours), seconds(seconds)
8              {
9              }
10
11         TwlvTime operator+(const TwlvTime& t)
12         {
13             TwlvTime time(*this);
14             time.hours += t.hours;
15             time.seconds += t.seconds;
16             return time;
17         }
18
19         public:
20             int hours;
21             int seconds;
22     };
23
24     int main()
25     {
26         TwlvTime t1(4, 37);
27         TwlvTime t2(2, 12);
28
29         TwlvTime t3 = t1 + t2;
30
31         std::cout << t3.hours << ":" << t3.seconds << '\n';
32
33         return 0;
34     }
```

Listing 6: Sample code

1 6:49

Listing 7: Output

4 2016/17 questions

4.1 Question 1a

```
1  #include <iostream>
2
3  using namespace std;
4
5  void f1(void * d, int s)
6  {
7      if (s == sizeof(double))
8      {
9          double * p;
10         p = (double *) d;
11         ++(*p);
12     }
13     else if (s == sizeof(int))
14     {
15         int * p;
16         p = (int *) d;
17         ++(*p);
18     }
19 }
20
21 int main()
22 {
23     double v1[] = {25, 75, 100};
24     int v2[] = {72, 76, 98};
25     cout << v1 << ' ' << v2 << endl;
26
27     double *p1 = v1;
28     int *p2 = v2;
29
30     f1(&v1, sizeof(p1));
31     f1(&v2, sizeof(p2));
32
33     cout << v1 << ' ' << v2 << endl;
34 }
```

Listing 8: Sample code

```
1  0x7ffc2c8bcd30 0x7ffc2c8bcd20
2  0x7ffc2c8bcd30 0x7ffc2c8bcd20
```

Listing 9: Output

4.2 Question 1b

```
1  #include <iostream>
2
3  using namespace std;
4
5  void f1(void * d, int s)
6  {
7      if (s == sizeof(double))
8      {
9          double * p;
10         p = (double *) d;
11         ++*p;
12     }
13     else if (s == sizeof(int))
14     {
15         int * p;
16         p = (int *) d;
17         ++*p;
18     }
19 }
20
21 int main()
22 {
23     double v1[] = {25, 75, 100};
24     int v2[] = {72, 76, 98};
25     cout << v1 << ' ' << v2 << endl;
26
27     double *p1 = v1;
28     int *p2 = v2;
29
30     f1(&v1, sizeof(p1));
31     f1(&v2, sizeof(p2));
32
33     cout << v1 << ' ' << v2 << endl;
34 }
```

Listing 10: Sample code

```
1  0x7ffd669a21c0 0x7ffd669a21b0
2  0x7ffd669a21c0 0x7ffd669a21b0
```

Listing 11: Output

4.3 Question 1c

```
1  #include <iostream>
2
3  using namespace std;
4
5  void f1(void * d, int s)
6  {
7      if (s == sizeof(double))
8      {
9          double * p;
10         p = (double *) d;
11         *p++;
12     }
13     else if (s == sizeof(int))
14     {
15         int * p;
16         p = (int *) d;
17         *p++;
18     }
19 }
20
21 int main()
22 {
23     double v1[] = {25, 75, 100};
24     int v2[] = {72, 76, 98};
25     cout << v1 << ' ' << v2 << endl;
26
27     double *p1 = v1;
28     int *p2 = v2;
29
30     f1(&v1, sizeof(p1));
31     f1(&v2, sizeof(p2));
32
33     cout << v1 << ' ' << v2 << endl;
34 }
```

Listing 12: Sample code

```
1  0x7ffe0603b500 0x7ffe0603b4f0
2  0x7ffe0603b500 0x7ffe0603b4f0
```

Listing 13: Output

4.4 Question 1d

```
1  #include <iostream>
2
3  using namespace std;
4
5  void f1(void * d, int s)
6  {
7      if (s == sizeof(double))
8      {
9          double * p;
10         p = (double *) d;
11         *p++;
12     }
13     else if (s == sizeof(int))
14     {
15         int * p;
16         p = (int *) d;
17         *p++;
18     }
19 }
20
21 int main()
22 {
23     double v1[] = {25, 75, 100};
24     int v2[] = {72, 76, 98};
25     cout << v1 << ' ' << v2 << endl;
26
27     double *p1 = v1;
28     int *p2 = v2;
29
30     f1(&v1, sizeof(p1));
31     f1(&v2, sizeof(p2));
32
33     cout << &v1 << ' ' << &v2 << endl;
34 }
```

Listing 14: Sample code

```
1  0x7ffdb5f5d9b0 0x7ffdb5f5d9a0
2  0x7ffdb5f5d9b0 0x7ffdb5f5d9a0
```

Listing 15: Output

4.5 Question 4

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      char a[] = {1, 2, 3, 4, 5};
8      char b[] = {6, 7, 8, 9, 10};
9
10     char *ptr1 = (char*) (&a + 1);
11     char *ptr2 = (char*) (a + 1);
12
13     cout << (int)*(ptr1 - 1) << ' ' << (int)*(ptr2 - 1) << endl;
14     cout << (int)a[0] << ' ' << &a << endl;
15 }
```

Listing 16: Sample code

```
1  5 1
2  1 0x7ffe357e6cc0
```

Listing 17: Output