Contents

Course material: `https://research.ncl.ac.uk/game/mastersdegree/programmingforgames/`

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

```
class ITouhou {
 public:
  ITouhou() { cout << "ITouhou (" << name() << ") constructed" << endl; }
  virtual ~ITouhou() { cout << "ITouhou destructed" << endl; }
  virtual string name() const { return "ITouhou"; };
};

class Youkai : public virtual ITouhou {
 public:
  Youkai() : ITouhou() { cout << "Youkai (" << name() << ") constructed" << endl; }
  virtual ~Youkai() { cout << "Youkai destructed" << endl; }
  virtual string name() const { return "Youkai"; };
};

class IFlyable: public virtual ITouhou {
 public:
  IFlyable() { cout << "IFlyable (" << name() <<") constructed" << endl; }
  virtual ~IFlyable() { cout << "IFlyable destructed" << endl; }
  virtual void fly() { cout << "IFlyable::fly()" << endl; }
  virtual string name() const { return "IFlyable"; }
};

class Yuuka : public Youkai, public IFlyable {
 public:
  Yuuka() : Youkai(), IFlyable()
    { cout << "Yuuka " << name() << " constructed" << endl; }
  virtual ~Yuuka() { cout << "Yuuka destructed" << endl; }
  virtual string name() const { return "Yuuka"; }
  virtual void fly() { cout << "Yuuka::fly()" << endl; }
};

class Reimu : public virtual ITouhou, public IFlyable
{
 public:
  Reimu() : ITouhou(), IFlyable()
    { cout << "Reimu (" << name() << ") constructed" << endl; }
  virtual ~Reimu() { cout << "Reimu destructed" << endl; }
  virtual string name() const { return "Reimu"; }
};
```

Listing 1: Polymorphism example classes

```
int main() {
  ITouhou *y1 = new Yuuka();
  delete y1;

  cout << "=====" << endl;

  Youkai *y2 = new Yuuka();
  delete y2;

  cout << "=====" << endl;

  Yuuka *y3 = new Yuuka();
  y3->fly();
  delete y3;

  cout << "=====" << endl;

  IFlyable *y4 = new Yuuka();
  y4->fly();
  delete y4;

  cout << "=====" << endl;

  IFlyable *r1 = new Reimu();
  r1->fly();
  delete r1;

  return 0;
}
```

Listing 2: Polymorphism example `main()`

```
ITouhou (ITouhou) constructed
Youkai (Youkai) constructed
IFlyable (IFlyable) constructed
Yuuka Yuuka constructed
Yuuka destructed
IFlyable destructed
Youkai destructed
ITouhou destructed
=====
ITouhou (ITouhou) constructed
Youkai (Youkai) constructed
IFlyable (IFlyable) constructed
Yuuka Yuuka constructed
Yuuka destructed
IFlyable destructed
Youkai destructed
ITouhou destructed
=====
ITouhou (ITouhou) constructed
Youkai (Youkai) constructed
IFlyable (IFlyable) constructed
Yuuka Yuuka constructed
Yuuka::fly()
Yuuka destructed
IFlyable destructed
Youkai destructed
ITouhou destructed
=====
ITouhou (ITouhou) constructed
Youkai (Youkai) constructed
IFlyable (IFlyable) constructed
Yuuka Yuuka constructed
Yuuka::fly()
Yuuka destructed
IFlyable destructed
Youkai destructed
ITouhou destructed
=====
ITouhou (ITouhou) constructed
IFlyable (IFlyable) constructed
Reimu (Reimu) constructed
IFlyable::fly()
Reimu destructed
IFlyable destructed
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 if 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

```
#include <iostream>

using namespace std;

int main()
{
  int list[10];
  int *p;

  p = list;
  for (int x = 1; x <= 10; x++)
  {
    *(p) = x; // The original line would have caused an invalid memory access
              // (element before start of array)
    p++;
    cout << &p << endl; // This outputs the address of the pointer variable p
  }

  for (int i = 0; i < 10; i++)
    cout << list[i] << endl;

  return 0;
}
```

Listing 4: Sample code

```
0x7ffe4c3b3b68
0x7ffe4c3b3b68
0x7ffe4c3b3b68
0x7ffe4c3b3b68
0x7ffe4c3b3b68
0x7ffe4c3b3b68
0x7ffe4c3b3b68
0x7ffe4c3b3b68
0x7ffe4c3b3b68
0x7ffe4c3b3b68
1
2
3
4
5
6
7
8
9
10
```

Listing 5: Output

3.2 2013/14 Question 3

```
#include <iostream>

class TwlvTime
{
  public:
    TwlvTime(int hours, int seconds)
      : hours(hours), seconds(seconds)
    {
    }

    TwlvTime operator+(const TwlvTime& t)
    {
      TwlvTime time(*this);
      time.hours += t.hours;
      time.seconds += t.seconds;
      return time;
    }

  public:
    int hours;
    int seconds;
};

int main()
{
  TwlvTime t1(4, 37);
  TwlvTime t2(2, 12);

  TwlvTime t3 = t1 + t2;

  std::cout << t3.hours << ":" << t3.seconds << '\n';

  return 0;
}
```

Listing 6: Sample code

```
6:49
```

Listing 7: Output

4 2016/17 questions

4.1 Question 1a

```
#include <iostream>

using namespace std;

void f1(void * d, int s)
{
  if (s == sizeof(double))
  {
    double * p;
    p = (double *) d;
    ++(*p);
  }
  else if (s == sizeof(int))
  {
    int * p;
    p = (int *) d;
    ++(*p);
  }
}

int main()
{
  double v1[] = {25, 75, 100};
  int v2[] = {72, 76, 98};
  cout << v1 << ' ' << v2 << endl;

  double *p1 = v1;
  int *p2 = v2;

  f1(&v1, sizeof(p1));
  f1(&v2, sizeof(p2));

  cout << v1 << ' ' << v2 << endl;
}
```

Listing 8: Sample code

```
0x7ffc2c8bcd30 0x7ffc2c8bcd20
0x7ffc2c8bcd30 0x7ffc2c8bcd20
```

Listing 9: Output

4.2 Question 1b

```
#include <iostream>

using namespace std;

void f1(void * d, int s)
{
  if (s == sizeof(double))
  {
    double * p;
    p = (double *) d;
    ++*p;
  }
  else if (s == sizeof(int))
  {
    int * p;
    p = (int *) d;
    ++*p;
  }
}

int main()
{
  double v1[] = {25, 75, 100};
  int v2[] = {72, 76, 98};
  cout << v1 << ' ' << v2 << endl;

  double *p1 = v1;
  int *p2 = v2;

  f1(&v1, sizeof(p1));
  f1(&v2, sizeof(p2));

  cout << v1 << ' ' << v2 << endl;
}
```

Listing 10: Sample code

```
0x7ffd669a21c0 0x7ffd669a21b0
0x7ffd669a21c0 0x7ffd669a21b0
```

Listing 11: Output

4.3 Question 1c

```
#include <iostream>

using namespace std;

void f1(void * d, int s)
{
  if (s == sizeof(double))
  {
    double * p;
    p = (double *) d;
    *p++;
  }
  else if (s == sizeof(int))
  {
    int * p;
    p = (int *) d;
    *p++;
  }
}

int main()
{
  double v1[] = {25, 75, 100};
  int v2[] = {72, 76, 98};
  cout << v1 << ' ' << v2 << endl;

  double *p1 = v1;
  int *p2 = v2;

  f1(&v1, sizeof(p1));
  f1(&v2, sizeof(p2));

  cout << v1 << ' ' << v2 << endl;
}
```

Listing 12: Sample code

```
0x7ffe0603b500 0x7ffe0603b4f0
0x7ffe0603b500 0x7ffe0603b4f0
```

Listing 13: Output

4.4 Question 1d

```
#include <iostream>

using namespace std;

void f1(void * d, int s)
{
  if (s == sizeof(double))
  {
    double * p;
    p = (double *) d;
    *p++;
  }
  else if (s == sizeof(int))
  {
    int * p;
    p = (int *) d;
    *p++;
  }
}

int main()
{
  double v1[] = {25, 75, 100};
  int v2[] = {72, 76, 98};
  cout << v1 << ' ' << v2 << endl;

  double *p1 = v1;
  int *p2 = v2;

  f1(&v1, sizeof(p1));
  f1(&v2, sizeof(p2));

  cout << &v1 << ' ' << &v2 << endl;
}
```

Listing 14: Sample code

```
0x7ffdb5f5d9b0 0x7ffdb5f5d9a0
0x7ffdb5f5d9b0 0x7ffdb5f5d9a0
```

Listing 15: Output

4.5 Question 4

```
#include <iostream>

using namespace std;

int main()
{
  char a[] = {1, 2, 3, 4, 5};
  char b[] = {6, 7, 8, 9, 10};

  char *ptr1 = (char*) (&a + 1);
  char *ptr2 = (char*) (a + 1);

  cout << (int)*(ptr1 - 1) << ' ' << (int)*(ptr2 - 1) << endl;
  cout << (int)a[0] << ' ' << &a << endl;
}
```

Listing 16: Sample code

```
5 1
1 0x7ffe357e6cc0
```

Listing 17: Output