Contents

1 Data Types and Variables

1.1 Types

A selection of data types that always seem to be on the exam:

`int a`
: An integer.

`int *a`
: A pointer to an integer.

`char a[3]`
: An array of characters of length 3.

`int *a[3]`
: An array of pointers to integers of length 3.

`int (*a)[3]`
: A pointer to an array of integers of length 3.

`int char *(*a)[]`
: A pointer to an array of pointers to characters.

`const int * a`
: A pointer to a `const` integer.

`int * const a`
: A `const` pointer to an integer

1.2 Declaration vs Definition

Declaration

- Provides basic attributes: type and name
- Does not allocate storage (for variables)
- e.g. `extern int a;`

Definition

- Defines all attributes of a symbol
- e.g. `int a;`

A symbol being declared but never defined results in a linker error.

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()
```

Listing 1: Polymorphism example classes

```
  virtual ~Reimu() { cout << "Reimu destructed" << endl; }
  virtual string name() const { return "Reimu"; }
};

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 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