
ME
My Code,Doing Exactly WhaT
I'M TELLING IT TO DO

Object Oriented Programming

CS2308

Gentry Atkinson

What makes good code?

- **Readability**: other programmers should be able to understand it.
- **Reusability**: if should be easy to us in other programs.
- **Security**: it should be protected from malicious users.

Do Classes make good code?

- **Readability**: programmers can quickly understand classes in terms of their interface.
- **Reusability**: classes can be moved easily because the interface is consistent.
- **Security**: classes prevent private data from being accessed incorrectly.

Object Oriented Programming

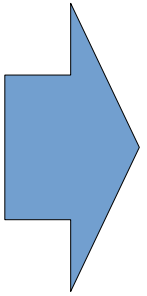
- A programming **paradigm** that asserts that all code (or as much code as possible) should be in classes.
- Programs should be written as abstractions of real world objects.
- A program is a collection of **classes.**

Other Paradigms

- **Procedural**: a program is a collection of **statements.** (We have been using this paradigm).
- **Functional**: a program is a collection of **functions.** (UT focuses on this paradigm).
- **Logical**: a program is a collection of **rules.**
- (There are many others. These are just a few examples.)

Defining Abstractions

- What defines a Dog?
  - A name, a breed, an age, and owner.
- What can a dog do?
  - Change owner.
  - Change name.
  - Get older.



```
class Dog{
  private:
    string name, breed, owner;
    int age;
  public:
    Dog(string, string, string, int);
    void getOlder();
    void changeOwner(string):
    void updateName(string);
    //TODO: add getters
};
```

Member Function Prototypes

- **class** definitions in C++ only have to include a prototype.
- Member functions can be defined elsewhere in the same file, or in a different file.
- The scope resolution operator :: is used to show that a function definition is part of a class.

Example 1

```
class C{
   private:
        string value;


    public:
        C(string);
        string getValue();
        void setValue(string);
};
```

```
C::C(string v){
    value = v;
}
string C::getValue(){
    return value;
}
void C::setValue(string v)
{
    value = v;
}
```

Example 1 cont.

```
int main(int argc, char** argv){
    C instanceOfC("CS");
    cout << instanceOfC.getValue();
    instanceOfC.setValue(" is ");
    cout << instanceOfC.getValue();
    instanceOfC.setValue("best!");
    cout << instanceOfC.getValue() << endl;
    return 0;
} //try to guess the output
```

Class variables vs. Instance variables.

- The **static** keyword can be used to make a member variable shared by the whole class.
- Every object gets its own copy of an instance (non-static) variable.
- All object from the same class share the same copy of a class (static) variable.

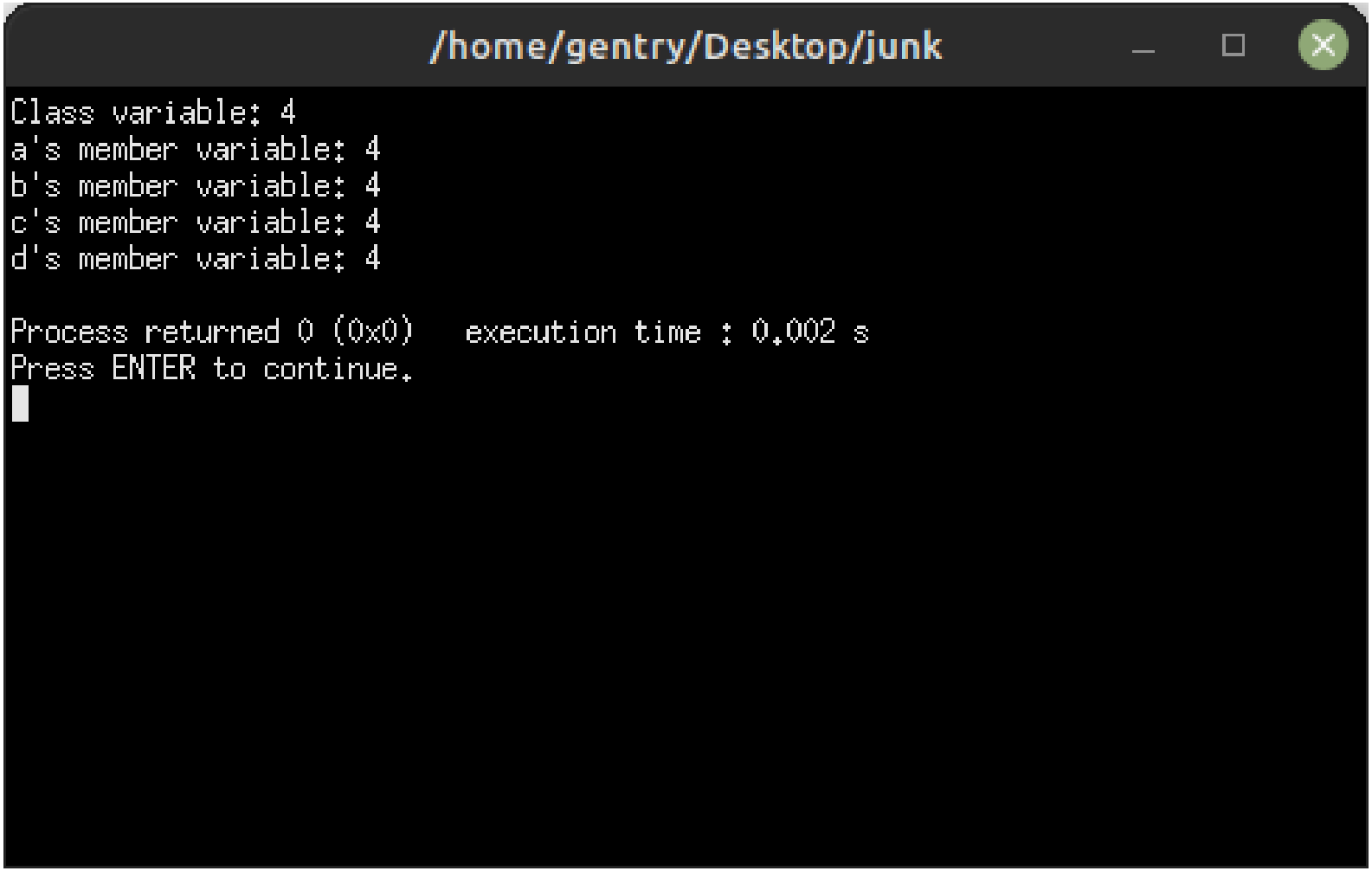
Example 2

```
class C{
    public:
        static int howMany;
        C(){howMany++;};
        int getHowMany(){return howMany;};
};
int C::howMany = 0;
```

Example 2 cont

```
int main(int argc, char** argv){

    C a, b, c, d;

    cout << "Class variable: " << C::howMany << endl;

    cout << "a's member variable: " << a.getHowMany()
<< endl;

    cout << "b's member variable: " << b.getHowMany()
<< endl;

    cout << "c's member variable: " << c.getHowMany()
<< endl;

    cout << "d's member variable: " << d.getHowMany()
<< endl;

    return 0;

} //try to guess the output
```

Example 2 cont



Classes with Class Members

- Classes can have instances of other classes as members variables.
- Class member variables are referenced using two or more dot . operators.

Example 3

```
class C{
    public: int a;
};
class D{
    private: C c;
    public:
        void setC(int c) {
          this->c.a = c;};
        int getC(){return c.a;};
};
```

```
int main(int argc, char** argv){
    D d;
    d.setC(5);
    cout << d.getC() << endl;
    return 0;
} //try to guess the output
```

Example 4

```
class C{
    private: int a;
    public:
        void setA(int a)
            {this->a = a;};
        int getA(){return a;};
};
class D{
    public: C c;
};
```

```
int main(int argc, char** argv){
    D d;
    d.c.setA(5);
    cout << d.c.getA() << endl;
    return 0;
} //try to guess the output
```

Arrays and Classes

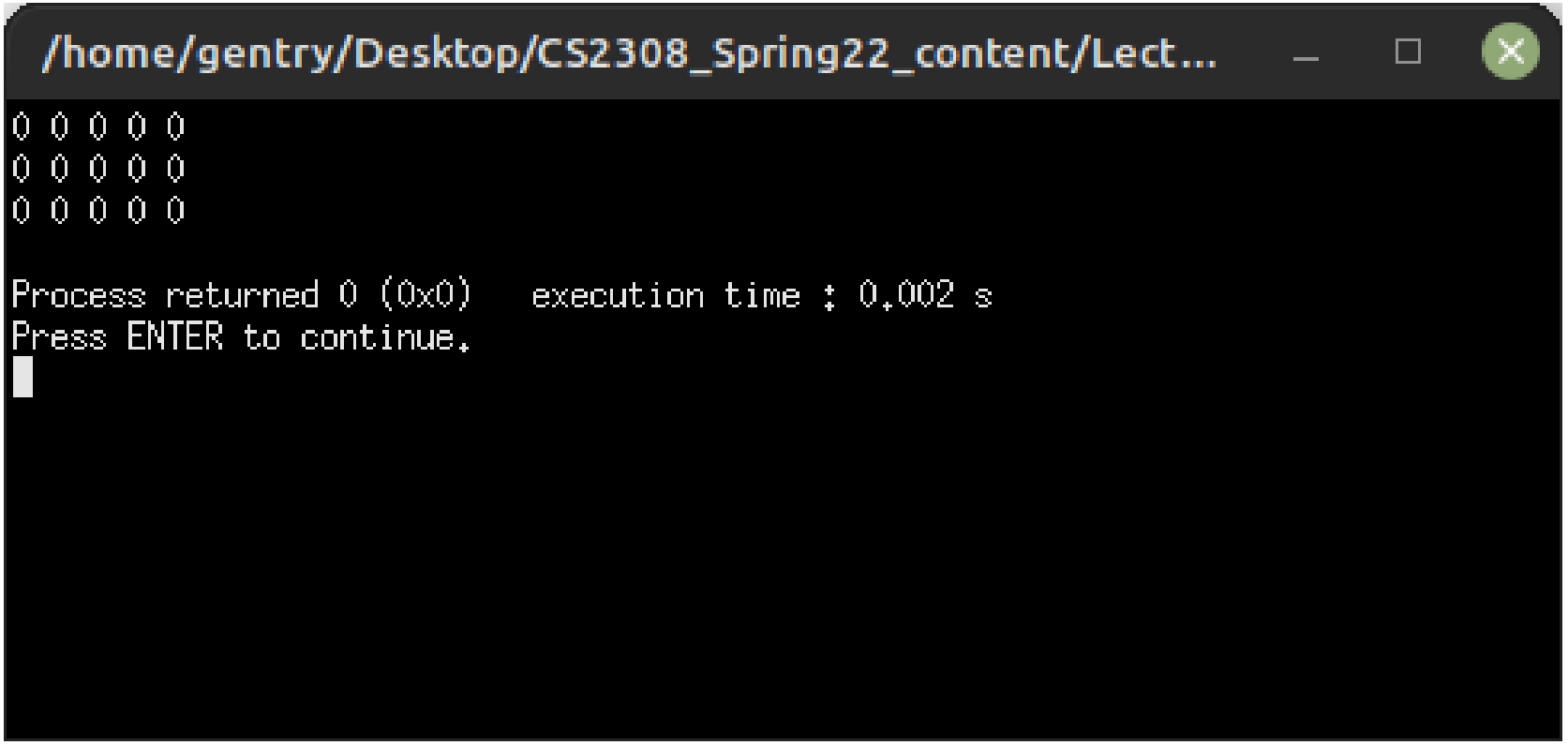
- Classes can have arrays as member variables.
- Objects can be stored in arrays.

Example 5

```
class C{
    private:
        int a[5] = {};
        int SIZE = 5;
    public:
        void printNums(){
            for(int i=0; i<SIZE;i++)
                cout << a[i] << ' ';
            cout << endl;
        }
};
```

```
int main(){
    C c[3];
    c[0].printNums();
    c[1].printNums();
    c[2].printNums();
    return 0;
} //try to guess the output
```

Example 5


/home/gentry/Desktop/CS2308_Spring22_content/Lect...
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.

Pointers and Classes

- Objects can be pointed to by pointers.
- Classes can have pointers as member variables.

Example 6

```
class C{
    private:
        int *a;
    public:
        void setA(int* a){
            this->a=a;
        }
        void printA(){
            cout << *a;
        }
};
```

```
int main(){
    C c;
    int b = 5;
    c.setA(&b);
    c.printA();
    return 0;
} //try to guess the output
```

Classes and Dynamic Allocation

- Pointer member variables of classes can reference dynamically allocated memory.
- Objects can be dynamically allocated
- Classes that create dynamically allocated video should use their destructor to de-allocate that memory.

Example 7

```
class C{
    private:
        int *a;
    public:
        C(){
            a = new int;
            *a = 5;
        }
        ~C(){delete a;}
        void printA(){
            cout << *a;
        }
};
```

```
int main(){
    C* c = new C;
    c->printA();
    delete c;
    return 0;
} //try to guess the output
```

Questions or Comments?