CSCI 200: Foundational Programming Concepts & Design Lecture 20

Objects and Functions

Sign in to iClicker

Previously in CSCI 200

- Create a vector of courses

Course
- enrollment: int - title: string
+ Course() + Course(string) + getTitle(): string + getEnrollment(): int + registerStudent(): void + withdrawStudent(): void

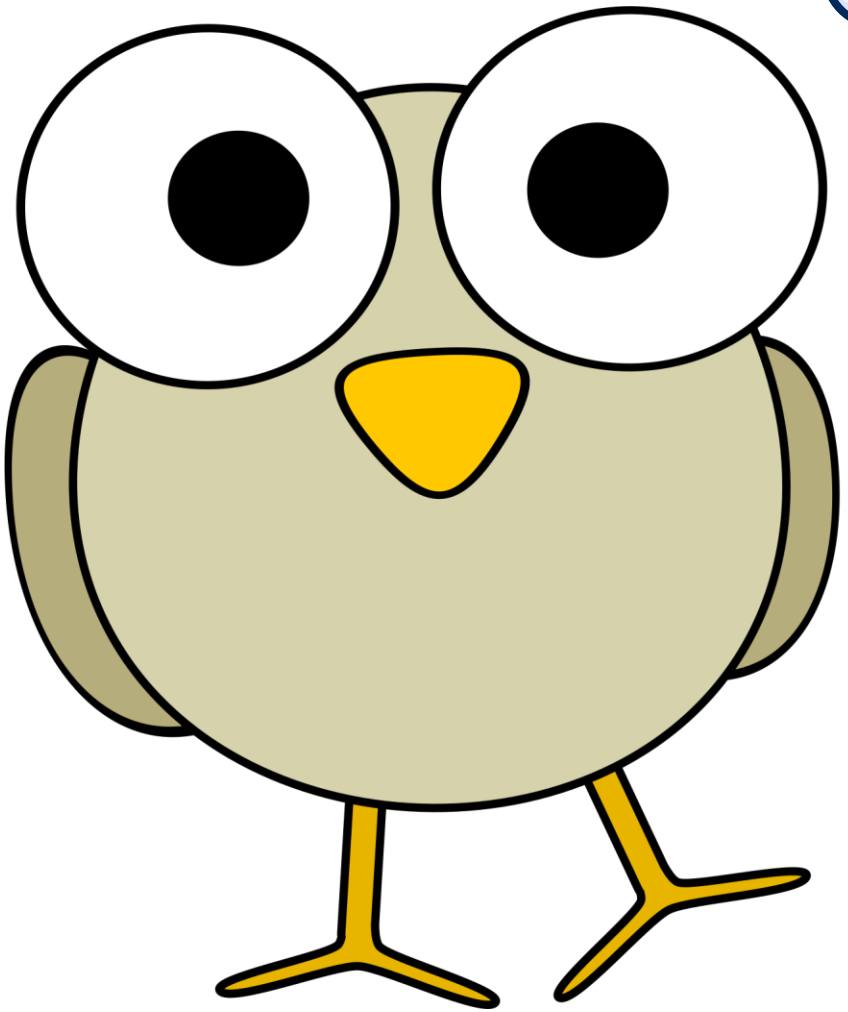
```
// initializes to zero
// initializes to CSM101

// sets title to param
// returns title of course
// returns enrollment of course
// increments enrollment by 1
// decrements enrollment by 1
```

Previously in CSCI 200

```
class Course {
public:
  Course() {
        _enrollment = 0;
        _title = "CSM 101";
  }
  Course(const string TITLE) {
        _enrollment = 0;
        _title = TITLE;
  }
  string getTitle() { return _title; }
  int getEnrollment() { return _enrollment; }
  void registerStudent() { _enrollment++; }
  void withdrawStudent() { if(_enrollment > 0) _enrollment--; }
private:
  int _enrollment;
  string _title;
};
```

Questions?



Learning Outcomes For Today

- Construct a program that accesses an element in a vector, returns the length of a vector, changes the length of the vector, and other vector operations.
- Construct a program that accesses an element in a string, returns the length of a string, changes the length of the string, and other string operations.
- Compare and contrast Procedural Programming with Object-Oriented Programming
- Explain the following terms and how they are used (1) dot operator / member access operator (2) data member (3) scope resolution operator
- Discuss the concept of scope within and outside a class & struct

On Tap For Today

- Collections of Objects
- Passing Objects to Functions
- Practice

On Tap For Today

- Collections of Objects
- Passing Objects to Functions
- Practice

Sample Class

```
class Course {
public:
  Course() {
        _enrollment = 0;
        _title = "CSM 101";
  }
  Course(const string TITLE) {
        _enrollment = 0;
        _title = TITLE;
  }
  string getTitle() { return _title; }
  int getEnrollment() { return _enrollment; }
  void registerStudent() { _enrollment++; }
  void withdrawStudent() { if(_enrollment > 0) _enrollment--; }
private:
  int _enrollment;
  string _title;
};
```

Consider V1

```
vector<Course> courseCatalog;
courseCatalog.push_back( Course() );
courseCatalog.push_back( Course("CSCI 200") );

// enroll students
for(size_t i = 0; i < courseCatalog.size(); i++) {
  courseCatalog.at(i).registerStudent();
  courseCatalog.at(i).registerStudent();

}

// print enrollments
for(size_t i = 0; i < courseCatalog.size(); i++) {
  cout << courseCatalog.at(i).getTitle() << " "
       << courseCatalog.at(i).getEnrollment() << endl;
}

// what does it print?
```

Consider V2

```
vector<Course> courseCatalog;
courseCatalog.push_back( Course() );
courseCatalog.push_back( Course("CSCI 200") );

// enroll students
for(size_t i = 0; i < courseCatalog.size(); i++) {
  Course currentCourse = courseCatalog.at(i);
  currentCourse.registerStudent();
  currentCourse.registerStudent();
}

// print enrollments
for(size_t i = 0; i < courseCatalog.size(); i++) {
  cout << courseCatalog.at(i).getTitle() << " "
       << courseCatalog.at(i).getEnrollment() << endl;
}

// what does it print?
```

Storing Objects on the Free Store

- Use a pointer!

```
int *pNumCars = new int;      //*pNumCars  initialized to 0
int *pNumCars2 = new int(5); //*pNumCars2 initialized to 5

Course *pCSM101 = new Course;// automatically calls constructor
Course *pCSCI200 = new Course("CSCI 200");
```

- **new** – "Computer, allocate enough memory in the free store for one object and tell me the starting address where the object will be stored. Initialize the object at that location."

Access Members of Object Pointers

- Must first dereference pointer before accessing

```
Course *pCSCI200 = new Course("CSCI 200");
(*pCSCI200).getEnrollment();
```

- But this gets ugly when members return pointers

```
vector<Course*> *pCourses = new vector<Course*>;
(*(*pCourses).at(0)).getEnrollment();
```

Use the Arrow Operator

- Dereference and access in one operation

```
Course *pCSCI200 = new Course("CSCI 200");
pCSCI200->getEnrollment();
```

- Much cleaner interface and denotes what type of thing we are working with at each level

```
vector<Course*> courses;
courses.at(0)->getEnrollment();
```

Precedence Table

Category	Precedence	Operator	Associativity
Parenthesis	1	( )	Innermost First
Scope Resolution	2	S::	Left to Right
Postfix Unary Operators	3	a++ a-- a. p-> f()	
Prefix Unary Operators	4	++a --a +a -a !a ~a (type)a &a *p new delete	Right to Left
Binary Operators	5	a*b a/b a%b	Left to Right
	6	a+b a-b	
Shift Operators	7	a<<b a>>b	
Relational Operators	8	a<b a>b a<=b a>=b	
	9	a==b a!=b	
Bitwise Operators	10	a&b	
	11	a^b	
	12	a|b	
Logical Operators	13	a&&b	
	14	a||b	
Assignment	15	a=b a+=b a-=b a*=b a/=b a%=b a&=b a^=b a|=b	Right to Left

Consider V2 - solved

```
vector<Course*> courseCatalog;
courseCatalog.push_back( new Course() );
courseCatalog.push_back( new Course("CSCI 200") );

// enroll students
for(size_t i = 0; i < courseCatalog.size(); i++) {
  Course* pCurrentCourse = courseCatalog.at(i);
  pCurrentCourse->registerStudent();
  pCurrentCourse->registerStudent();
}

// print enrollments
for(size_t i = 0; i < courseCatalog.size(); i++) {
  cout << courseCatalog.at(i)->getTitle() << " "
       << courseCatalog.at(i)->getEnrollment() << endl;
}

// what does it print?
```

On Tap For Today

- Collections of Objects
- Passing Objects to Functions
- Practice

Passing Vectors to Function?

- Like any other single value: PBV or PBR or PBP

```
void print_vector_b_v( vector<int> vec ) {
  for( int i = 0; i < vec.size(); i++ )
    cout << vec.at(i) << endl;
}
void print_vector_b_r( vector<int>& vec ) {
  for( int i = 0; i < vec.size(); i++ )
    cout << vec.at(i) << endl;
}
void print_vector_b_p( vector<int>* pVec ) {
  for( int i = 0; i < pVec->size(); i++ )
    cout << pVec->at(i) << endl;
}
```

- Be aware of PBV / PBR / PBP implications. Concerns?

Passing Vectors to Function

- Like any other single value: PBV or PBR or PBP

```
void add_to_vector_b_v( vector<int>   vec ) {
  vec.push_back( 100 );
}
void add_to_vector_b_r( vector<int>&  vec ) {
  vec.push_back( 200 );
}
void add_to_vector_b_p( vector<int>* pVec ) {
  pVec->push_back( 300 );
}
// ...
vector<int> myVec;
add_to_vector_b_v(  myVec ); cout << myVec.size() << endl;
add_to_vector_b_r(  myVec ); cout << myVec.size() << endl;
add_to_vector_b_p( &myVec ); cout << myVec.size() << endl;
```

String Parameter Beware

```
void string_func_v( string   str ) {...}
void string_func_r( string&  str ) {...}
void string_func_p( string* pStr ) {...}
...
string word = "does this work?";
string_func_v(  word );
string_func_r(  word );
string_func_p( &word );
string_func_v( "does this work?" );
string_func_r( "does this work?" );
string_func_p( "does this work?" );
```

String Parameter Beware

```
void string_func_v( string   str ) {...}
void string_func_r( string&  str ) {...}
void string_func_p( string* pStr ) {...}
...
string word = "does this work?";
string_func_v(  word );                  // YES
string_func_r(  word );                  // YES
string_func_p( &word );                  // YES
string_func_v( "does this work?" ); // YES
string_func_r( "does this work?" ); // NO :(
string_func_p( "does this work?" ); // NO :(
```

How Much Memory Used?

```
void string_func_v( string   str ) {...}
void string_func_r( string&  str ) {...}
void string_func_p( string* pStr ) {...}
...
string word = "does this work?";
string_func_v(  word );
string_func_r(  word );
string_func_p( &word );
string_func_v( "does this work?" );
```

How Much Memory Used?

```
void string_func_v( string   str ) {...}
void string_func_r( string&  str ) {...}
void string_func_p( string* pStr ) {...}
...
string word = "does this work?";
string_func_v(  word );                 // 2 copies
string_func_r(  word );                 // 1 copy
string_func_p( &word );                 // 1 copy + pointer
string_func_v( "does this work?" ); // 1 copy
```

When To Use PBV/PBR/PBP ?

```
void string_func_v( string   str ) {...}
void string_func_r( string&  str ) {...}
void string_func_p( string* pStr ) {...}
...
string word = "does this work?";
string_func_v(  word );                 // 2 copies
string_func_r(  word );                 // 1 copy
string_func_p( &word );                 // 1 copy + pointer
string_func_v( "does this work?" ); // 1 copy
```

For Maximum Flexibility - Overload

```
void string_func( string   str ) {...}
void string_func( string* pStr ) { string_func(*pStr); }
...
string word = "does this work?";
string_func(  word );                  // 2 copies
string_func( &word );                  // 2 copies + pointer
string_func( "does this work?" ); // 1 copy
```

For Maximum Efficiency – PBR

```
void string_func( string&  str ) {...}

...

string word = "does this work?";

string_func(  word );                  // 1 copy

// string_func( "does this work?" ); // not supported
```

How About Vectors?

- Memory Usage? Runtime?

```
void vector_v( vector<int>   vec ) {...}
void vector_r( vector<int>&  vec ) {...}
void vector_p( vector<int>* pVec ) {...}
...
vector<int> numbers = {...};
vector_v(  numbers );
vector_r(  numbers );
vector_p( &numbers );
```

How About Vectors?

- Memory Usage? Runtime?

```
void vector_v( vector<int>   vec ) {...}
void vector_r( vector<int>&  vec ) {...}
void vector_p( vector<int>* pVec ) {...}
...
vector<int> numbers = {...};
vector_v(  numbers );  // 2 copies - O(n)
vector_r(  numbers );  // 1 copy - O(1)
vector_p( &numbers );  // 1 copy + pointer - O(1)
```

Vector Operations

- Element Access – O(1)
- Vector Traversal – O(n)

- (Will continue to add to)

How About Vectors?

- Use PBR always!

```
// RW
void vector_rw( vector<int>& vec ) {...}
// R only
void vector_r( const vector<int>& VEC ) {...}
...
vector<int> numbers = {...};
vector_rw( numbers );      // 1 copy - O(1)
vector_r(  numbers );      // 1 copy - O(1)
```

On Tap For Today

- Collections of Objects
- Passing Objects to Functions
- Practice

To Do For Next Time

- Wednesday: File I/O + vector & string Quiz
- Friday: Final Project Proposal due
- Fall Break!
- Thursday: A3 due