* defining a function

void happy(int size, int dog)

* calling a function

happy(12, 34)

* function prototypes

void happy(int, int);

* passing parameters (by value and by reference)

happy(size, dog)

happy(&size, &dog)

* defining classes

in .hpp

#ifndef DOG\_HPP

#define DOG\_HPP

class Dog()

{

private:

string name;

int age;

public:

Dog(string, int);

void setAge(int);

void setName(string);

}; **Ends with a ;**

#endif

In .cpp

#include Dog.hpp

Dog::Dog(string dog, int years){}

Void Dog::setAge(int){}

* creating and using objects

Dog dog1(“Darcy”, 3)

dog1.getAge

* constructors and destructors

Dog::Dog(string dog, int years){

setName(dog);

setAge(years);

}

**Learn more about destructors**

**Dog::~Dog()**

* private and public

private can’t be accessed outside of class, but public can.

i.e. age=dog1.age or age=age error

age=dog1.getAge() correct

* arrays

int years[3]={1923, 1384, 2016}

* accessing and modifying arrays

years[0]=1925 changes the first value to 1925

years[1]=1925 changes second value to 1925

* passing arrays as arguments to functions

void dog(int []);

void dog(int years[]){}

* multidimensional arrays

int dog[3][3]; is a 3 row 3 column array of ints.

* C-style strings

An array of char that make a c-string

* pointer variables

int \*ptr is value. Ptr is location.

\*(number+1) is the same as \*number[1]

Number->getScore() is same as (\*number).getScore()

* the address operator (&)

int\* ptr;

ptr=&value

& is the location of value, not the value of value

* using pointers
* dynamic memory allocation

array = new int[size]

delete [] array

Some things that will **not** be on the exam are:

* enum
* structs
* parallel arrays
* pointer arithmetic
* typedef
* jagged arrays
* vectors