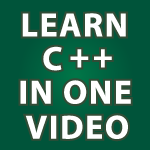
delete

[**C++ PROGRAMMING TUTORIAL**](http://www.newthinktank.com/2014/11/c-programming-tutorial/)

Posted by [Derek Banas](http://www.newthinktank.com/author/admin/) on Nov 20, 2014 in [Uncategorized](http://www.newthinktank.com/category/uncategorized/) | [8 comments](http://www.newthinktank.com/2014/11/c-programming-tutorial/#comments)

[](http://www.newthinktank.com/2014/11/c-programming-tutorial/c-programming/)In this tutorial I teach the entire C++ Programming language in one video tutorial. The cheat sheet follows the video below.

This is part of what is covered Data Types, Arithmetic, If, Switch, Ternary Operator, Arrays, For Loop, While Loop, Do While Loop, User Input, Strings, Vectors, Functions, Recursion, File I/O, Exceptions, Pointers, Reference Operator, Class, Objects, Private, Public, Prototypes, Static, Encapsulation, Constructors, Destructors, This, Inheritance, Virtual Methods, Polymorphism, Abstract Data Types and more.

If you like videos like this, I’d appreciate it if you told Google with a click here



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515  516  517  518  519  520  521  522  523  524  525  526  527  528  529  530  531  532  533  534  535  536  537  538  539  540  541  542  543  544  545  546  547  548  549  550  551  552  553  554  555  556  557  558  559  560  561  562  563  564  565  566  567  568  569  570  571  572  573  574  575  576  577  578  579  580  581  582  583  584  585  586  587  588  589  590  591  592  593  594  595  596  597  598  599  600  601  602  603  604  605  606  607  608  609  610  611  612  613  614  615  616  617  618  619  620  621  622  623  624  625  626  627  628  629  630  631  632  633  634  635  636  637  638  639  640  641  642  643  644  645  646  647  648  649  650  651  652  653  654  655  656  657  658  659  660  661  662  663  664  665  666  667  668  669  670  671  672  673  674  675  676  677  678  679  680  681  682  683  684  685  686  687  688  689  690  691  692  693  694  695  696  697  698  699  700  701  702  703  704  705  706  707  708  709  710  711  712  713  714  715  716  717  718  719  720  721  722  723  724  725  726  727  728  729  730  731  732  733  734  735  736  737  738  739  740  741  742  743  744  745  746  747  748  749  750  751  752  753  754  755  756  757  758  759  760  761  762  763  764  765  766  767  768  769  770  771  772  773  774  775  776  777  778  779  780  781 | // This is a comment  /\*  \* Multi-line comment  \*/    // Tells the compiler iostream library which contains the function cout  #include <iostream>    // Allows us to use vectors  #include <vector>    // Allows us to use strings  #include <string>    // Allow us to work with files  #include <fstream>    // Allows functions in the std namespace to be used without their prefix  // std::cout becomes cout  using namespace std;    // ---------- FUNCTIONS ----------  // The function has return type, function name and attributes with  // their data types  // The attribute data types must match the value passed in  // This data is passed by value  // You can define default values to attributes as long as they come last  // This is known as a function prototype  int addNumbers(int firstNum, int secondNum = 0){    int combinedValue = firstNum + secondNum;    return combinedValue;    }    // An overloaded function has the same name, but different attributes  int addNumbers(int firstNum, int secondNum, int thirdNum){    return firstNum + secondNum + thirdNum;    }    // A recursive function is one that calls itself    int getFactorial(int number){    int sum;  if(number == 1) sum = 1;  else sum = (getFactorial(number - 1) \* number);  return sum;    // getFactorial(2) [Returns 2] \* 3  // getFactorial(1) [Returns 1] \* 2 <This value goes above>  // 2 \* 3 = 6    }    // Doesn't have a return type so use void  // Since I'm getting a pointer use int\*  // Refer to the referenced variable with \*age  void makeMeYoung(int\* age){    cout << "I used to be " << \*age << endl;  \*age = 21;    }    // A function that receives a reference can manipulate the value globally  void actYourAge(int& age){    age = 39;    }    // ---------- END OF FUNCTIONS ----------    // ---------- CLASSES ----------  // classes start with the name class    class Animal  {    // private variables are only available to methods in the class  private:  int height;  int weight;  string name;    // A static variable shares the same value with every object in the class  static int numOfAnimals;    // Public variables can be accessed by anything with access to the object  public:  int getHeight(){return height;}  int getWeight(){return weight;}  string getName(){return name;}  void setHeight(int cm){ height = cm; }  void setWeight(int kg){ weight = kg; }  void setName(string dogName){ name = dogName; }    // Declared as a prototype  void setAll(int, int, string);    // Declare the constructor  Animal(int, int, string);    // Declare the deconstructor  ~Animal();    // An overloaded constructor called when no data is passed  Animal();    // protected members are available to members of the same class and  // sub classes    // Static methods aren't attached to an object and can only access  // static member variables  static int getNumOfAnimals() { return numOfAnimals; }    // This method will be overwritten in Dog  void toString();    };    int Animal::numOfAnimals = 0;    // Define the protoype method setAll  void Animal::setAll(int height, int weight, string name){    // This is used to refer to an object created of this class type  this -> height = height;  this -> weight = weight;  this -> name = name;  Animal::numOfAnimals++;    }    // A constructor is called when an object is created  Animal::Animal(int height, int weight, string name) {    this -> height = height;  this -> weight = weight;  this -> name = name;    }    // The destructor is called when an object is destroyed  Animal::~Animal() {    cout << "Animal " << this -> name << " destroyed" << endl;    }    // A constructor called when no attributes are passed  Animal::Animal() {  numOfAnimals++;  }    // This method prints object info to screen and will be overwritten  void Animal::toString(){    cout << this -> name << " is " << this -> height << " cms tall and "  << this -> weight << " kgs in weight" << endl;    }    // We can inherit the variables and methods of other classes  class Dog : public Animal{    private:  string sound = "Woof";  public:  void getSound() { cout << sound << endl; }    // Declare the constructor  Dog(int, int, string, string);    // Declare the default constructor and call the default superclass  // constructor  Dog() : Animal(){};    // Overwrite toString  void toString();    };    // Dog constructor passes the right attributes to the superclass  // constructor and then handles the attribute bark that remains  Dog::Dog(int height, int weight, string name, string bark) :  Animal(height, weight, name){    this -> sound = bark;    }    // toString method overwritten  void Dog::toString(){    // Because the attributes were private in Animal they must be retrieved  // by called the get methods  cout << this -> getName() << " is " << this -> getHeight() <<  " cms tall and " << this -> getWeight() << " kgs in weight and says " <<  this -> sound << endl;    }    // ---------- END OF CLASSES ----------    // This is where execution begins. Attributes can be sent to main  int main() {    // cout outputs text and a carriage return with endl  // Statements must end with a semicolon  // Strings must be surrounded by "  // << sends the text via standard output to the screen  cout << "Hello Internet" << endl;    // ---------- VARIABLES / DATA TYPES ----------  // Variables start with a letter and can contain letters, numbers and \_  // They are case sensitive    // A value that won't change is a constant  // Starts with const and it should be uppercase  const double PI = 3.1415926535;    // chars can contain 1 character that are surrounded with ' and is one byte in size  char myGrade = 'A';    // bools have the value of (true/1) or (false/0)  bool isHappy = true;    // ints are whole numbers  int myAge = 39;    // floats are floating point numbers accurate to about 6 decimals  float favNum = 3.141592;    // doubles are floating point numbers accurate to about 15 digits  double otherFavNum = 1.6180339887;    // You can output a variable value like this  cout << "Favorite Number " << favNum << endl;    // Other types include  // short int : At least 16 bits  // long int : At least 32 bits  // long long int : At least 64 bits  // unsigned int : Same size as signed version  // long double : Not less then double    // You can get the number of bytes for a data type with sizeof    cout << "Size of int " << sizeof(myAge) << endl;  cout << "Size of char " << sizeof(myGrade) << endl;  cout << "Size of bool " << sizeof(isHappy) << endl;  cout << "Size of float " << sizeof(favNum) << endl;  cout << "Size of double " << sizeof(otherFavNum) << endl;    int largestInt = 2147483647;    cout << "Largest int " << largestInt << endl;    // ---------- ARITHMETIC ----------  // The arithmetic operators are +, -, \*, /, %, ++, --    cout << "5 + 2 = " << 5+2 << endl;  cout << "5 - 2 = " << 5-2 << endl;  cout << "5 \* 2 = " << 5\*2 << endl;  cout << "5 / 2 = " << 5/2 << endl;  cout << "5 % 2 = " << 5%2 << endl;    int five = 5;  cout << "5++ = " << five++ << endl;  cout << "++5 = " << ++five << endl;  cout << "5-- = " << five-- << endl;  cout << "--5 = " << --five << endl;    // Shorthand assignment operators  // a += b == a = a + b  // There is also -=, \*=, /=, %=    // Order of Operation states \* and / is performed before + and -    cout << "1 + 2 - 3 \* 2 = " << 1 + 2 - 3 \* 2 << endl;  cout << "(1 + 2 - 3) \* 2 = " << (1 + 2 - 3) \* 2 << endl;    // ---------- CASTING ----------  // You convert from one data type to another by casting  // char, int, float, double    cout << "4 / 5 = " << 4 / 5 << endl;  cout << "4 / 5 = " << (float) 4 / 5 << endl;    // ---------- IF STATEMENT ----------  // Executes different code depending upon a condition    // Comparison operators include ==, !=, >, <, >=, <=  // Will return true (1) if the comparison is true, or false (0)    // Logical operators include &&, ||, !  // Used to test 2 or more conditionals    int age = 70;  int ageAtLastExam = 16;  bool isNotIntoxicated = true;    if((age >= 1) && (age < 16)){  cout << "You can't drive" << endl;  } else if(!isNotIntoxicated){  cout << "You can't drive" << endl;  } else if(age >= 80 && ((age > 100) || ((age - ageAtLastExam) > 5))){  cout << "You can't drive" << endl;  } else {  cout << "You can drive" << endl;  }    // ---------- SWITCH STATEMENT ----------  // switch is used when you have a limited number of possible options    int greetingOption = 2;    switch(greetingOption){    case 1 :  cout << "bonjour" << endl;  break;    case 2 :  cout << "Hola" << endl;  break;    case 3 :  cout << "Hallo" << endl;  break;    default :  cout << "Hello" << endl;  }    // ---------- TERNARY OPERATOR ----------  // Performs an assignment based on a condition  // variable = (condition) ? if true : if false    int largestNum = (5 > 2) ? 5 : 2;    cout << "The biggest number is " << largestNum << endl;    // ---------- ARRAYS ----------  // Arrays store multiple values of the same type    // You must provide a data type and the size of the array  int myFavNums[5];    // You can declare and add values in one step  int badNums[5] = {4, 13, 14, 24, 34};    // The first item in the array has the label (index) of 0  cout << "Bad Number 1: " << badNums[0] << endl;    // You can create multidimensional arrays  char myName[5][5] = {{'D','e','r','e','k'},{'B','a','n','a','s'}};    cout << "2nd Letter in 2nd Array: " << myName[1][1] << endl;    // You can change a value in an array using its index  myName[0][2] = 'e';    cout << "New Value " << myName[0][2] << endl;    // ---------- FOR LOOP ----------  // Continues to execute code as long as a condition is true    for(int i = 1; i <= 10; i++){    cout << i << endl;    }    // You can also cycle through an array by nesting for loops  for(int j = 0; j < 5; j++){    for(int k = 0; k < 5; k++){  cout << myName[j][k];  }    cout << endl;    }    // ---------- WHILE LOOP ----------  // Use a while loop when you don't know ahead of time when a loop will end    // Generate a random number between 1 and 100  int randNum = (rand() % 100) + 1;    while(randNum != 100){    cout << randNum << ", ";    // Used to get you out of the loop  randNum = (rand() % 100) + 1;    }    cout << endl;    // You can do the same as the for loop like this  // Create an index to iterate out side the while loop  int index = 1;    while(index <= 10){    cout << index << endl;    // Increment inside the loop  index++;    }    // ---------- DO WHILE LOOP ----------  // Used when you want to execute what is in the loop at least once    // Used to store a series of characters  string numberGuessed;  int intNumberGuessed = 0;    do {      cout << "Guess between 1 and 10: ";        // Allows for user input      // Pass the source and destination of the input      getline (cin,numberGuessed);        // stoi converts the string into an integer      intNumberGuessed = stoi(numberGuessed);      cout << intNumberGuessed << endl;        // We'll continue looping until the number entered is 4  } while (intNumberGuessed != 4);      cout << "You Win" << endl;    // ---------- STRINGS ----------  // The string library class provides a string object  // You must always surround strings with "  // Unlike the char arrays in c, the string object automatically resizes    // The C way of making a string  char happyArray[6] = {'H', 'a', 'p', 'p', 'y', '\0'};    // The C++ way  string birthdayString = " Birthday";    // You can combine / concatenate strings with +  cout << happyArray + birthdayString << endl;    string yourName;  cout << "What is your name? ";  getline (cin,yourName);    cout << "Hello " << yourName << endl;    double eulersConstant = .57721;  string eulerGuess;  double eulerGuessDouble;  cout << "What is Euler's Constant? ";  getline (cin,eulerGuess);    // Converts a string into a double  // stof() for floats  eulerGuessDouble = stod(eulerGuess);    if(eulerGuessDouble == eulersConstant){    cout << "You are right" << endl;    } else {    cout << "You are wrong" << endl;    }    // Size returns the number of characters  cout << "Size of string " << eulerGuess.size() << endl;    // empty tells you if string is empty or not  cout << "Is string empty " << eulerGuess.empty() << endl;    // append adds strings together  cout << eulerGuess.append(" was your guess") << endl;    string dogString = "dog";  string catString = "cat";    // Compare returns a 0 for a match, 1 if less than, -1 if greater then  cout << dogString.compare(catString) << endl;  cout << dogString.compare(dogString) << endl;  cout << catString.compare(dogString) << endl;    // assign copies a value to another string  string wholeName = yourName.assign(yourName);  cout << wholeName << endl;    // You can get a substring as well by defining the starting index and the  // number of characters to copy  string firstName = wholeName.assign(wholeName, 0, 5);  cout << firstName << endl;    // find returns the index for the string your searching for starting  // from the index defined  int lastNameIndex = yourName.find("Banas", 0);  cout << "Index for last name " << lastNameIndex << endl;    // insert places a string in the index defined  yourName.insert(5, " Justin");  cout << yourName << endl;    // erase will delete 6 characters starting at index 7  yourName.erase(6,7);  cout << yourName << endl;    // replace 5 characters starting at index 6 with the string Maximus  yourName.replace(6,5,"Maximus");  cout << yourName << endl;    // ---------- VECTORS ----------  // Vectors are like arrays, but their size can change    vector <int> lotteryNumVect(10);    int lotteryNumArray[5] = {4, 13, 14, 24, 34};    // Add the array to the vector starting at the beginning of the vector  lotteryNumVect.insert(lotteryNumVect.begin(), lotteryNumArray, lotteryNumArray+3);    // Insert a value into the 5th index  lotteryNumVect.insert(lotteryNumVect.begin()+5, 44);    // at gets the value in the specified index  cout << "Value in 5 " << lotteryNumVect.at(5) << endl;    // push\_back adds a value at the end of a vector  lotteryNumVect.push\_back(64);    // back gets the value in the final index  cout << "Final Value " << lotteryNumVect.back() << endl;    // pop\_back removes the final element  lotteryNumVect.pop\_back();    // front returns the first element  cout << "First Element " << lotteryNumVect.front() << endl;    // back returns the last element  cout << "Last Element " << lotteryNumVect.back() << endl;    // empty tells you if the vector is empty  cout << "Vector Empty " << lotteryNumVect.empty() << endl;    // size returns the total number of elements  cout << "Number of Vector Elements " << lotteryNumVect.size() << endl;    // ---------- FUNCTIONS ----------  // Functions allow you to reuse and better organize your code    cout << addNumbers(1) << endl;    // You can't access values created in functions (Out of Scope)  // cout << combinedValue << endl;    cout << addNumbers(1, 5, 6) << endl;    cout << "The factorial of 3 is " << getFactorial(3) << endl;    // ---------- FILE I/O ----------  // We can read and write to files using text or machine readable binary    string steveQuote = "A day without sunshine is like, you know, night";    // Create an output filestream and if the file doesn't exist create it  ofstream writer("stevequote.txt");    // Verify that the file stream object was created  if(! writer){    cout << "Error opening file" << endl;    // Signal that an error occurred  return -1;    } else {    // Write the text to the file  writer << steveQuote << endl;    // Close the file  writer.close();    }    // Open a stream to append to whats there with ios::app  // ios::binary : Treat the file as binary  // ios::in : Open a file to read input  // ios::trunc : Default  // ios::out : Open a file to write output  ofstream writer2("stevequote.txt", ios::app);    if(! writer2){    cout << "Error opening file" << endl;    // Signal that an error occurred  return -1;    } else {    writer2 << "\n- Steve Martin" << endl;  writer2.close();    }    char letter;    // Read characters from a file using an input file stream  ifstream reader("stevequote.txt");    if(! reader){    cout << "Error opening file" << endl;  return -1;    } else {    // Read each character from the stream until end of file  for(int i = 0; ! reader.eof(); i++){    // Get the next letter and output it  reader.get(letter);  cout << letter;    }    cout << endl;  reader.close();    }    // ---------- EXCEPTION HANDLING ----------  // You can be prepared for potential problems with exception handling    int number = 0;    try{    if(number != 0){  cout << 2/number << endl;  } else throw(number);    }  catch(int number){    cout << number << " is not valid input" << endl;    }    // ---------- POINTERS ----------  // When data is stored it is stored in an appropriately sized box based  // on its data type    int myAge = 39;  char myGrade = 'A';    cout << "Size of int " << sizeof(myAge) << endl;  cout << "Size of char " << sizeof(myGrade) << endl;    // You can reference the box (memory address) where data is stored with  // the & reference operator    cout << "myAge is located at " << &myAge << endl;    // A pointer can store a memory address  // The data type must be the same as the data referenced and it is followed  // by a \*    int\* agePtr = &myAge;    // You can access the memory address and the data  cout << "Address of pointer " << agePtr << endl;    // \* is the dereference or indirection operator  cout << "Data at memory address " << \*agePtr << endl;    int badNums[5] = {4, 13, 14, 24, 34};  int\* numArrayPtr = badNums;    // You can increment through an array using a pointer with ++ or --  cout << "Address " << numArrayPtr << " Value " << \*numArrayPtr << endl;  numArrayPtr++;  cout << "Address " << numArrayPtr << " Value " << \*numArrayPtr << endl;    // An array name is just a pointer to the array  cout << "Address " << badNums << " Value " << \*badNums << endl;    // When you pass a variable to a function you are passing the value  // When you pass a pointer to a function you are passing a reference  // that can be changed    makeMeYoung(&myAge);    cout << "I'm " << myAge << " years old now" << endl;    // & denotes that ageRef will be a reference to the assigned variable  int& ageRef = myAge;    cout << "ageRef : " << ageRef << endl;    // It can manipulate the other variables data  ageRef++;    cout << "myAge : " << myAge << endl;    // You can pass the reference to a function  actYourAge(ageRef);    cout << "myAge : " << myAge << endl;    // When deciding on whether to use pointers or references  // Use Pointers if you don't want to initialize at declaration, or if  // you need to assign another variable  // otherwise use a reference    // ---------- CLASSES & OBJECTS ----------  // Classes are the blueprints for modeling real world objects  // Real world objects have attributes, classes have members / variables  // Real world objects have abilities, classes have methods / functions  // Classes believe in hiding data (encapsulation) from outside code    // Declare a Animal type object  Animal fred;    // Set the values for the Animal  fred.setHeight(33);  fred.setWeight(10);  fred.setName("Fred");    // Get the values for the Animal  cout << fred.getName() << " is " << fred.getHeight() << " cms tall and "  << fred.getWeight() << " kgs in weight" << endl;    fred.setAll(34, 12, "Fred");    cout << fred.getName() << " is " << fred.getHeight() << " cms tall and "  << fred.getWeight() << " kgs in weight" << endl;    // Creating an object using the constructor  Animal tom(36, 15, "Tom");    cout << tom.getName() << " is " << tom.getHeight() << " cms tall and "  << tom.getWeight() << " kgs in weight" << endl;    // Demonstrate the inheriting class Dog  Dog spot(38, 16, "Spot", "Woof");    // static methods are called by using the class name and the scope operator  cout << "Number of Animals " << Animal::getNumOfAnimals() << endl;    spot.getSound();    // Test the toString method that will be overwritten  tom.toString();  spot.toString();    // We can call the superclass version of a method with the class name  // and the scope operator  spot.Animal::toString();    // When a function finishes it must return an integer value  // Zero means that the function ended with success  return 0;  } |



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70 | #include <iostream>  using namespace std;    // Virtual Methods and Polymorphism  // Polymorpism allows you to treat subclasses as their superclass and yet  // call the correct overwritten methods in the subclass automatically    class Animal{  public:  void getFamily() { cout << "We are Animals" << endl; }    // When we define a method as virtual we know that Animal  // will be a base class that may have this method overwritten  virtual void getClass() { cout << "I'm an Animal" << endl; }  };    class Dog : public Animal{  public:  void getClass() { cout << "I'm a Dog" << endl; }    };    class GermanShepard : public Dog{  public:  void getClass() { cout << "I'm a German Shepard" << endl; }  void getDerived() { cout << "I'm an Animal and Dog" << endl; }    };    void whatClassAreYou(Animal \*animal){  animal -> getClass();  }    int main(){    Animal \*animal = new Animal;  Dog \*dog = new Dog;    // If a method is marked virtual or not doesn't matter if we call the method  // directly from the object  animal->getClass();  dog->getClass();    // If getClass is not marked as virtual outside functions won't look for  // overwritten methods in subclasses however  whatClassAreYou(animal);  whatClassAreYou(dog);    Dog spot;  GermanShepard max;    // A base class can call derived class methods as long as they exist  // in the base class  Animal\* ptrDog = &spot;  Animal\* ptrGShepard = &max;    // Call the method not overwritten in the super class Animal  ptrDog -> getFamily();    // Since getClass was overwritten in Dog call the Dog version  ptrDog -> getClass();    // Call to the super class  ptrGShepard -> getFamily();    // Call to the overwritten GermanShepard version  ptrGShepard -> getClass();    return 0;  } |

Part 3 of C++ Tutorial

C++



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | #include <iostream>  using namespace std;    // Polymorpism allows you to treat subclasses as their superclass and yet  // call the correct overwritten methods in the subclass automatically    class Animal{  public:  virtual void makeSound(){ cout << "The Animal says grrrr" << endl; }    // The Animal class could be a capability class that exists  // only to be derived from by containing only virtual methods  // that do nothing    };    class Cat : public Animal{  public:  void makeSound(){ cout << "The Cat says meow" << endl; }    };    class Dog : public Animal{  public:  void makeSound(){ cout << "The Dog says woof" << endl; }    };    // An abstract data type is a class that acts as the base to other classes  // They stand out because its methods are initialized with zero  // A pure virtual method must be overwritten by subclasses    class Car{  public :  virtual int getNumWheels() = 0;  virtual int getNumDoors() = 0;  };    class StationWagon : public Car{  public :  int getNumWheels() { cout << "Station Wagon has 4 Wheels" << endl; }  int getNumDoors() { cout << "Station Wagon has 4 Doors" << endl; }  StationWagon() { }  ~StationWagon();    };    int main(){    Animal\* pCat = new Cat;  Animal\* pDog = new Dog;    pCat -> makeSound();  pDog -> makeSound();    // Create a StationWagon using the abstract data type Car  Car\* stationWagon = new StationWagon();    stationWagon -> getNumWheels();    return 0;  } |