Questions:

1. Declare a pointer to a float and a pointer to a char.

float\* fptr;

char\* cptr;

1. What do we use the *sizeof()* operator for? If we have two integers, *ax* and *bx*, defined one after the other, how can we simulate the purpose of the *sizeof()* operator?

Use the *sizeof()* operator to get the byte-size of a variable or data type.

int ax, bx;

cout << &bx << “ - “ << &ax << “ = “

<< ( int(&bx) - int(&ax) ) << endl; // the space between each address is

// sizeof(int)

Example output: <http://ideone.com/gC7ThD>

0xbfbd9f7c - 0xbfbd9f78 = 4

1. Explain, in words, what the following code snippet does. Make sure to use the words “dereference”, “pointer” and “reference (&)” inside your explanation.

double num1 = 7.0; // Define a double variable named num1, set it to 7.0

double& num2 = num1; // Define a reference to num1 named num2

double\* dptr = &num1; // Define a double pointer named dptr and point it to

// the address num1 is located at.

cout << num1 << endl; // Print out num1’s value.

cout << &num1 << endl; // Print out the address num1 is located at

num2 = 33.4; // Set num2 (i.e. num1) equal to 33.4

\*dptr = 1234.567; // Set the variable located to where dptr is pointing

// to as 1234.567.

cout << num2 << endl; // Print out the value located at num2 (i.e. num1)

cout << &num2 << endl; // Print out the address num2 refers to (i.e. num1)

1. Suppose the operating system you were working on defines a pointer to an int as 8 bytes (64 bits) wide. How wide would a pointer to a short be? A string pointer? Explain.

Both a pointer to a short or a pointer to a string would be of size 8 bytes as well as any other pointer defined on this operating system.

As an address on a 64 bit operating system is 64 bits wide (or 8 bytes, as 8 bits is 1 byte), and a pointer of any type holds an address, each pointer would thus be 8 bytes as well.

1. Explain, using your knowledge of the relationship between pointers and arrays, on why the following code produces its output.

|  |  |
| --- | --- |
| **Syntax** | **Output** |
| #include <iostream>  using namespace std;  void printArrayMultByTwo(int \* int\_list, const int SIZE)  {  for(int i = 0; i < SIZE; ++i)  {  int\_list[i] \*= 2;  cout << int\_list[i] << " ";  }  cout << endl << endl;  }  int main()  {  const int LIST\_SIZE = 5;  int my\_list[] = {27, 32, 55, 1, 3752};  printArrayMultByTwo(my\_list, LIST\_SIZE);  printArrayMultByTwo(my\_list, LIST\_SIZE);  printArrayMultByTwo(my\_list, LIST\_SIZE);    return 0;  } | 54 64 110 2 7504  108 128 220 4 15008  216 256 440 8 30016  *int\_list* is a pointer that points to the memory location (the address) of where *my\_list* is located. From here, it has access to the entirety of *my\_list* by using offsets (i.e. using the bracket [] operator, or pointer arithmetic).  [This page](http://www.tutorialspoint.com/cplusplus/cpp_pointers_vs_arrays.htm) has some really good info on the relationship between arrays and pointers. |

1. Using the code on the left, make a main function that creates a pointer that dynamically creates a Rectangle, sets its width to 3.0 and height to 4.0, prints out the rectangle, and deletes the Rectangle afterwards. Assume all the methods are well-defined.

|  |  |
| --- | --- |
| **Given Code** | **Your Main Function** |
| class Rectangle  {  double width, height;  public:  Rectangle(); // defaults width = height = 1.0  Rectangle(double w, double h);    // getters  double getWidth();  double getHeight();    // setters  void setWidth(double w);  void setHeight(double h);  // miscellaneous  double getArea();  void printSelf();  }; | int main()  {  Rectangle\* my\_rect = new Rectangle(3.0, 4.0);  // or  // Rectangle\* my\_rect;  // \*(my\_rect).setWidth(3.0);  // \*(my\_rect).setHeight(4.0);  \*(my\_rect).printSelf();  // or:  // my\_rect->printSelf();  delete my\_rect;  return 0;  } |