CS 162 Recitation 3 Worksheet

1. Understand terms:
   - Reference – an alternative name that is refer to an existing variable.
   - Pointer – a variable that holds a memory address where a value lives.
   - Dereference – access the value in memory location pointed to by a pointer.
   - Compile time memory – memory created during compile time, lives on stack. Also called static memory.
   - Runtime memory – memory created during runtime, lives on heap. Also called dynamic memory.
   - Allocate – create memory, usually refers to heap memory. In C++, use “new” to allocate dynamic memory during runtime.
   - Deallocate – delete memory, usually refers to heap memory. In C++, use “delete” to deallocate dynamic memory during runtime.

2. Reference vs. pointer syntax
   Ex. For a – f, state whether the * is 1) declaring a pointer or 2) dereferencing a pointer:
   a. char* p;
   b. x = *p + 1;
   c. int* ptr;
   d. *ptr = 5;
   e. (*ptr)++;
   f. char* p1[10];

   Ex. For a – d, state whether the & is 1) declaring a reference variable or 2) address of:
   a. string &var2 = var1;
   b. void func (double& num1);
   c. int* ptr = &value;
   d. int** ptr2 = &ptr;

3. Understand Types with & or *
   - & operator (Address-of):
     Applying & to a variable of type T gives a type T* → & adds a * to the resulting type
     i.e.,
     int x;
     double z;
     int* ptr1 = &x; // &(int) → int*
     double* ptr2 = &z; // &(double) → double*
     int** ptr3 = &ptr1; // &(int*) → int**
• * operator (dereference):

   Applying * to a variable of type T* gives a type T → every * in the expression cancels a *
   from the type of variable

   i.e.,
   
   int a = *ptr1; // *(int*) → int
   *ptr2 = 1.25; // *(double*) → double
   *ptr3 = ptr1; // *(int**) → int*
   **ptr3 = 5; // **(int**) → int

Ex. Consider the following declarations:

   int x = 100, data[3] = {1, 2, 3};
   int *pi = &x;
   int **ppi = &pi;
   int &ref = x;

For each of the following expressions, indicate the data type.
Possible options: int, int*, int**, int***

   a. x          c. ref
   b. &x         d. &ref
   e. pi         h. data
   f. *pi        i. &data
   g. &pi        j. data[0]
   k. ppi        m. *ppi
   l. &ppi       n. **ppi

Common mistake: Why is the following wrong?

   int* area (int wid, int hei) {
     int ans = wid * hei;
     return &ans;
   }

   int main () {
     int w = 5, h = 10, *a;
     a = area(w, h);
     cout << *a << endl;
     return 0;
   }

Takeaway: Never return a pointer to a local variable!
4. Dynamic Memory

Ex. Fill in the blanks:

a. ________ data = new int; ______________;

b. ________ data = new char; ______________;

c. ________ data = new char[100]; ______________;

d. ________ data = new double[20]; ______________;

e. ________ data = new string; ______________;

f. ________ data = new char*[10]; ______________;

For a-f above, how would you free/delete the memory? Write them down individually.

5. For the following arrays, draw the picture out, and state what is on the stack and what is on the heap.

a. bool b_array[2];
   b_array[0] = true;
   b_array[1] = false;

b. int **arr = new int* [3];
   for (int i = 0; i < 3; i++)
     arr[i] = new int [5];

c. char* letters = new char [32];

d. float matrix[2][3];
   matrix[1][2] = 10.24;