CS 162 Worksheet 4

1. Useful keywords: break vs. continue
   break: ends the current loop (not if statement) and continues execution after its last statement. Note: it only stops the INNER-MOST loop, not ALL nested loops
   Ex. Modify the following program using break statement. (Stop the loop if a negative guess is entered)
   a. int guess;
   b. bool done = false;
   c. while (done == false) {
      d.      cout << “Enter your guess: ”;
      e.      cin >> guess;
      f.      if (guess < 0)
      g.          done = true;
      h.      else
      i.      //process guess
   i0.}
   continue: stops the current iteration and continues execution
   Consider two ways for repeating a loop to get a new guess if a negative guess is entered
   Note: Often continue can be eliminated by changing the if condition
   int guess;
   bool done = false;
   while (done == false) {
      cout << “Enter your guess: ”;
      cin >> guess;
      if (guess < 0)
         continue;
      //process guess (here if guess >= 0)
   i0.}

2. 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.
3. 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;

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

   • * operator (dereference):
     Applying * to a variable of type T* gives a type T \rightarrow every * in the expression cancels a * from the type of variable
     i.e.,
     int a = *ptr1; // *(int*) \rightarrow int
     *ptr2 = 1.25; // *(double*) \rightarrow double
     *ptr3 = ptr1; // *(int**) \rightarrow int*
     **ptr3 = 5; // **(int**) \rightarrow int
Ex. Consider the following declarations:
```c++
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  
   Possible options: int, int*, int**, int***
   b. &x  
   Possible options: int, int*, int**, int***
   c. ref
   Possible options: int, int*, int**, int***
   d. &ref
   Possible options: int, int*, int**, int***
   e. pi
   Possible options: int, int*, int**, int***
   f. *pi
   Possible options: int, int*, int**, int***
   g. &pi
   Possible options: int, int*, int**, int***
   h. data
   Possible options: int, int*, int**, int***
   i. &data
   Possible options: int, int*, int**, int***
   j. data[0]
   Possible options: int, int*, int**, int***
   k. ppi
   Possible options: int, int*, int**, int***
   l. &ppi
   Possible options: int, int*, int**, int***
   m. *ppi
   Possible options: int, int*, int**, int***
   n. **ppi
   Possible options: int, int*, int**, int***

Common mistake: Why is the following wrong?
```c++
int* area (int wid, int hei) {
    int ans = wid * hei;
    return &ans;
}
```
```c++
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!

5. Dynamic Memory
Ex. Fill in the blanks:
Free/delete the memory:
```c++
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.

6. 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;