**CS 162 Worksheet 3**

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)

* 1. int guess;
  2. bool done = false;
  3. while (done == false) {
  4. cout << “Enter your guess: ”;
  5. cin >> guess;
  6. if (guess < 0)
  7. done = true;

**break;**

* 1. else
  2. //process guess

10.}

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; int guess;

bool done = false; bool done = false;

while (done == false) { while (done == false) {

cout << “Enter your guess: ”; cout << “Enter your guess: ”;

cin >> guess; cin >> guess;

if (guess < 0) if (guess >=0 ) {

continue; //process guess

//process guess (here if guess >= 0) }

} }

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.

1. Reference vs. pointer syntax

Ex. For a – f, state whether the \* is 1) declaring a pointer or 2) dereferencing a pointer:

* 1. char\* p; **Declaring**
  2. x = \*p + 1; **Dereferencing**
  3. int\* ptr; **Declaring**
  4. \*ptr = 5; **Dereferencing**
  5. (\*ptr)++; **Dereferencing**
  6. char\* p1[10]; **Declaring**

Ex. For a – d , state whether the & is 1) declaring a reference variable or 2) address of:

1. string &var2 = var1; **Declaring**
2. void func (double& num1); **Declaring**
3. int\* ptr = &value; **Address**
4. int\*\* ptr2 = &ptr; **Address**
5. 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\*\*\*

1. X **int** c. ref **int\***
2. &x **int** d. &ref **int\***
3. pi **int\*** h. data **int\***
4. \*pi **int** i. &data **int\*\***
5. &pi **int\*\*`** j. data[0] **int**
6. ppi **int\*\*** m. \*ppi **int\***
7. &ppi **int\*\*\*** n. \*\*ppi **int**

Common mistake: Why is the following wrong?

int\* area (int wid, int hei) {

int ans = wid \* hei;

return &ans; // **out of scope**

}

int main () {

int w = 5, h = 10, \*a;

a = area(w, h);

cout << \*a << endl; // **segmentation fault**

return 0;

}

Takeaway: Never return a pointer to a local variable!

1. Dynamic Memory

Ex. Fill in the blanks: Free/delete the memory:

* 1. **Int\*** data = new int; **delete data;**
  2. **Char\*** data = new char; **delete data;**
  3. **Char\*** data = new char[100]; **delete [] data;**
  4. **Double\*** data = new double[20]; **delete [] data;**
  5. **String\*** data = new string; **delete data;**
  6. **char\*\*** data = new char\*[10]; **delete [] data;**

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

1. For the following arrays, draw the picture out, and state what is on the stack and what is on the heap.
   1. bool b\_array[2];

b\_array[0] = true;

b\_array[1] = false;

**All of *b\_array* is on the stack**

* 1. int \*\*arr = new int\* [3];

for (int i = 0; i < 3; i++)

arr[i] = new int [5];

* 1. char\* letters = new char [32];
  2. float matrix[2][3];

matrix[1][2] = 10.24;

**All of *matrix* is on the stack**