CIS 014 – C++ Programming

Lecturer: Joseph Su

REFERENCES

Optional Textbook:

Programming: Principles and Practice Using C++, 2nd ed, B. Stroustrup, Addison-Wesley, 2014

PDF:

http://www.cplusplus.com/files/tutorial.pdf

Online:

http://www.cplusplus.com/doc/tutorial/

The C++ Programming Language, 4th ed.

B. Stroustrup, Addison-Wesley, 2013

C++ How to Program, 10th ed

Deitel & Deitel, Pearson Hall, 2016

C++ Primer, 5th ed

S. Lippman, J. Lajoie, and B. Moo, Addison-Wesley, 2012

READING ASSIGNMENTS

ONLINE

- Difference between const int*, const int * const, and int const *

REFERENCES

ASCII http://www.cplusplus.com/doc/ascii/
BOOLEAN http://www.cplusplus.com/doc/boolean/
RAND(): http://www.cplusplus.com/reference/cstdlib/rand/

http://www.cplusplus.com/files/tutorial.pdf (pages 1-85)
http://www.cplusplus.com/doc/tutorial/

- ✓ Program Structure
  - Complete all chapters
- ✓ Compound Data Types
  - Complete all chapters

TODAY

- `const` Keyword/Qualifier
- Pointers: Compute String Length

- Reviews:
  - NULL (nullptr) Pointers
  - Pointers and Arrays
  - Pointer Arithmetic
  - Passing Parameters
    - By Pointers
    - By Reference
    - By Declared Array (decaying into pointer)

THE CONST QUALIFIER

- `const` qualifier is used on a value that cannot be changed or modified.

Examples:

(1)
```
const int* ptr = &k; // ptr is not constant; the value that ptr
                     // points to is constant
*ptr = 5;            // NOT ALLOWED!!!
```

(2)
```
int* const ptr = new int;   // constant ptr, which means it
                            //cannot be changed, modified,
                            //or re-assigned
ptr = &k;                   // NOT ALLOWED!!!
```

POINTERS: COMPUTE STRING LENGTH

Write the following function to calculate the length of a const C-string, using pointer:

```
int length(const char* str);
```

Recall a C-string is always terminated with a '\0'. You may use arithmetic and dereference operators, but not the indexing operator ([]).

POINTERS: COMPUTE STRING LENGTH

```
int length(const char* str);
```

- str is a pointer pointing to the beginning address of the first character in this C string
- The original string that str points to cannot be changed or modified
- We need to iterate through each character pointed to by str with a loop such as WHILE
- We will loop until str advances to the location of the (NULL) nullptr terminator, or ‘\0’, in the C string

CAN YOU IMPLEMENT THE FUNCTION?

POINTERS: COMPUTE STRING LENGTH

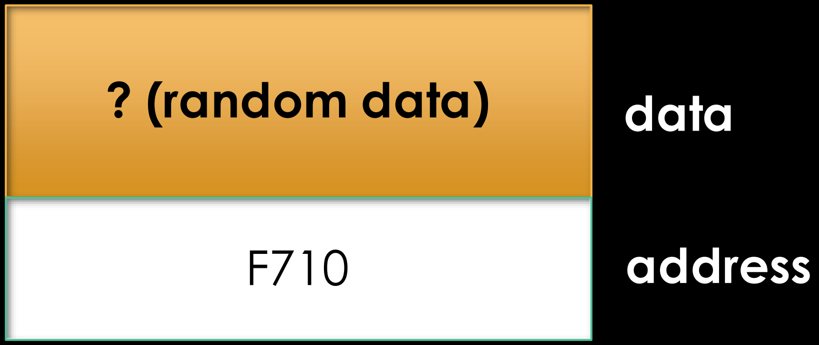
```
int length(const char* str);
```

There are several ways to solve this. One of them:

```
int length(const char* str) {
   int length = 0;
   while (*str != ‘\0’) {
      str++;
      length++;
   }
   return length;
}
```

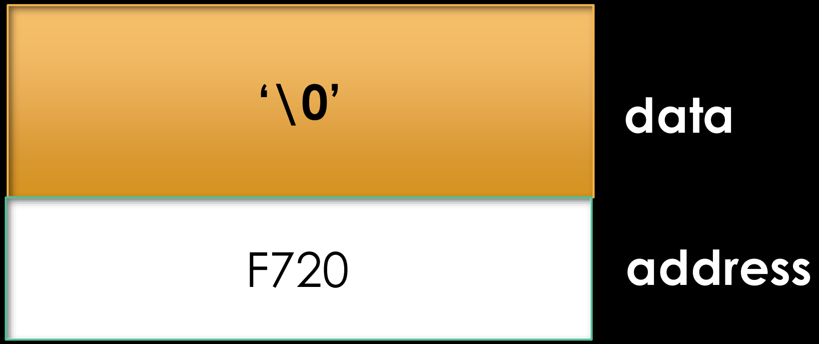
REVIEW: NULL (nullptr) POINTERS

- Declaring a pointer variable:



char* p;

- Assigning nullptr to p:



char* p = nullptr;

- Note p’s data is a memory address, NOT 0.
- *p produces an error: invalid memory access at addr 0.

REVIEW: POINTERS AND ARRAYS

- The name of an array is a pointer to the first element in the array. Two examples:

```
// statically declared on stack
char arr[3]; c

// dynamically allocated on heap
char* arr = new char[3];
```

In either instance:

- arr is a pointer, pointing to the first element arr[0]
- arr[2] returns the element that is 2 elements away from the starting element, arr[0], of arr.
- &arr[2] returns the address of the element that is 2 elements away from the starting element of arr

REVIEW: POINTER ARITHMETIC

- OK: subtraction & addition of pointers.
- Objective: to move around between locations in memory.

Example:

```
int arr[4] = {1, 4, 23, 6};
int* pPtr = arr;
pPtr++;
int* qPtr = arr + 3;
```

- p++ moves the p pointer down to the next element in array.
- qPtr – pPtr gives the number (2) of array elements between pPtr and qPtr.
- Recall pPtr was at 2nd element of array after pPtr++.

REVIEW: PASSING BY POINTERS

- Pointer is a data type storing the memory address of another variable.
- The address of a variable can be passed into a function using the & operator.
- The address of a variable is nothing more than a pointer:

```
 int main () {
    int k = 10;
    func(&k);
    return 0;
 }

void func(int* ptr) {
   *ptr = 20;  //dereferencing ptr points to k’s data
}
```

REVIEW: PASSING BY REFERENCE

```
int main () {
   int k = 10;
   func(k);
   return 0;
}
void func(int& ref) {
   ref = 20;       // ref is alias to k; both point to
                   // the same data
}
```

- The reference of a variable is simply itself.
- Both the reference and the original variable point to the same data in memory space.
- In fact, we can create a reference variable/object:

```
 int y = 2;
 int& x = y;
```

- In the above example, x is a reference to y; changing x also changes y.

REVIEW: PASSING BY DECLARED ARRAY

```
int main () {
   int arr[3] = {0};
   func(arr);
   return 0;
}
void func(int a[]) {
   …
}
```

- Recall a statically declared arr is decayed into a pointer when called.
- arr points to the address of the first element in arr.
- In `func()`, a is a pointer to the array, arr.
- Passing parameters by declared array is passing by pointers.