# Intermediate Programming / C and C++
# CSCI 2212

Review of CS 110

August 28, 2015

Arrays

Characters and Character Operations

# Part 1

1. Arrays

Terminology
Declarations
Initializers
Array Processing

# Terminology

An array is a variable with one name, one type and several slots for values.

- **base type:** the type you give in the array declaration.
- **length:** the number of slots in the array.
- **size:** the number of slots * the size of the base type.
- **slot:** also called position
- **subscript:** the slot number, also called index.
- **value:** also called element, the contents of a slot

# Declaring an array.

To create an array object, write square brackets after the name in a declaration. Here, I declare three arrays. All of them have three slots.

```
double v1[] = { 2.1, -.2, 10 };
double v2[3] = { 4.8, 2, -1.3 };
double v3[3];
```

Unlike Java, it is NOT necessary to write `new` to create an array object.

# Array initializers.

- You can declare an array WITHOUT initialization, like v3, above. For this kind of declaration, you must give the array length in the square brackets.
- You can initialize an array in the declaration using { } around a series of values (like v1, above). For this kind of declaration, you may omit the array length in the square brackets.
- You can also give BOTH give the array length AND an initializer (like v2, above).

# More on Initializers

- If you declare an array length longer than the number of initial values you give, the initial values will be used to fill the first part of the array, and zeros will fill the remaining slots;.
- If you write a empty braces, $\{\}$, all slots will be 0.
- If you write too many values in the braces, it is a compile-time error.

Examples: Suppose we were studying the families of 35-year-old men. For each man, we want to store the ages of all family members, up to 10 children:

```
int ages[12] = {35}; // Dad's age = 35, others are unknown.
int siblings[10] = {}; // Initialize all slots to 0.
```

# For loops were made for arrays.

Use a `for` loop to process an array.

- Usually, the loop variable starts at zero.
- Usually, the loop ends when the loop variable reaches the array length.
- Here, we add two arrays and put the answer in a 3rd array.
  `for(int k=0; k<3; ++k) v3[k] = v1[k] + v2[k] ;`
- Please leave a space after each semicolon in your loop.

# Using arrays.

What can you do with arrays?

- Carry out a computation on every slot: above, we added two vectors to get a third.
- Input/output data to/from an array.
- Find the maximum or the minimum of the array values.
- Send the array and its contents into a functions.
- Send an empty array into a function and have the function store data in it. That data will come back to the caller.

# Example: Reading data into an array.

```
#define N  24;

    double  temperature[N];  // To atore the input data
    int k;                   // loop counter

   puts( "Storing hourly temperatures for yesterday." ):
   puts( "Enter 24 temps, starting with time 0:00" ):
    for( k=0; k<N; ++k )
        scanf ("%lg", &temperature[k] );
```

# Example: Compute the maximum.

```
// Set initial position and value of max to slot 0.
int k, slot = 0;
double max = temperature[0];

 // Start comparisons with slot 1.
for(k=1; k<N; ++k) {
    // If current temp is highest so far,
    if( temperature[k] > max) {
        max = temperature[k];  // save it,
        slot = k;              // and save its position.
   }
}
```

# Parallel Arrays: Chapter 10.3

We can use a set of arrays to represent a set of properties of a series of objects.

- Define a global constant for the array length.
```
#define NPEOPLE 100
```
- Declare all your arrays using this constant.
```
int k;
int age[NPEOPLE];
float hourlyPay[NPEOPLE];
```
- Use one subscript (representing one object) for all arrays.
```
int k;
printf( "%6i %4.2f", age[k], hourlyPay[k] );
```

# Array Parameters: Chapter 10.4

```
float fill( float* data ); // OR float data[]
float hourlyPay[N];
```

When you call a function with an array parameter,

- The call should give the name of the array.
  `float payroll = fill( hourlyPay );`
- The function receives a pointer to the beginning of the array, that is, `&hourlyPay[0]`
- The function and the caller SHARE the memory of the array.
- In the function, either subscript syntax or pointer syntax can be used to access the array. Sometimes we use both

```
float total += data[k];
float* end = data+k;
```

# Part 2: Chapter 8

## 2. Characters and Character Operations

Character types.
Character operations.

# Data type char

A char is a dual-purpose data type:

- Representing characters like 'A' and 'z' and '5' and '}'.
- Representing very small integers. Note: '5' != 5.

Corresponding to the two purposes, a `char` has a split personality:

- Use `" %c"` to read a character and `"%c"` to write one. This reads a single keystroke.
- Use `"%i"` to read a small integer into a char variable. This could read a number with more than one keystroke.

**Note the space** between the quote and the percent. This tells C to skip leading whitespace. This is often necessary if the input contains blanks or newline characters. If you forget to write that space in the format, your second input line may not read correctly.

# Signed and unsigned chars

Like other integer types, chars can be either signed or unsigned.

- Some operating systems use signed chars for characters, others use unsigned chars.
- Most of the time you don't know and you don't need to know.
- ASCII code can be stored in a signed char, since it is a 7-bit code, the sign bit is not important.
- International ASCII is an 8-bit code and requires unsigned char.
- Modern systems are moving to Unicode, a 16-bit character representation.

# Operations on chars

You can do some kinds of computations with characters.

| | |
|---|---|
| Copy them: | `(ch == 'A')` |
| Compare them: | `if (ch == 'A' \|\| ch =='a')` |
| Add an integer: | `nextLetter = ch + 1` |
| Subtract a char to get an int: | `position = ch - 'A';` |
| Use as a subscript: | `counter[ch]++` |
| Use in a switch: | `switch( ch ) {` |
| | `case 'A':...` |

You need char subtraction and/or a char subscript to do program 6.

# Character Processing Functions

The ctype library contains functions for processing character data.

| | |
|---|---|
| `isalpha(ch)` | true for A...Z and a...z, false otherwise |
| `isdigit(ch)` | true for 0...9, false otherwise |
| `isalnum(ch)` | same as `isalpha() || isdigit()` |
| `isspace(ch)` | true for a whitspace character, false otherwise (space, tab, newline, CR, vertical tab or formfeed) |
| `islower(ch)` | true for a...z, false otherwise |
| `isupper(ch)` | true for A...Z, false otherwise |
| `tolower(ch)` | If ch is A...Z, return a...z. Else return ch unchanged. |
| `toupper(ch)` | If ch is a...z, return A...Z. Else return ch unchanged. |

You need isalpha() and toupper() or tolower() to do program 6.