

CS262 Lecture 02

Chapters 1&2

Zoran Duric
Department of Computer Science



- see makefile and word-count.c

standard c library

- assert.h //define macro **assert**
- ctype.h //classify and transform individual characters
- limits.h //defines the limits of **integral** types
- float.h //describes the characteristics of **floating** types
- math.h //trigonometric, exponential, logarithmic, ..
- string.h //string, array and memory manipulation
- stdio.h //file IO
- stdlib.h //string conversion, random number, memory management, searching, sorting, termination

Outline

- variables
 - types (int, char, float, double, enum, union, struct)
 - type conversion
 - array and char array (i.e. string)
 - scopes (automatic, static, extern)
- functions
 - parameters (caller), arguments (callee)
 - call by value vs. call by reference
- operators (almost identical with java)
 - arithmetic, bitwise, boolean,
 - order of evaluation

enumerate (enum)

- **enum bool {false, true};**
 - bool flag=false;
- **enum error {no_error=1, io_error, mem_error};**
 - error msg=no_error;
- **union package { int id; char type; };**
 - package a;
 - usage: a.id=0; or
 - a.type='c';
 - but not both!

array

- very similar to java but there are some important differences
- **int tmp[3];**
- **int tmp[3]={0,0,0};**
- **const int tmp[]={0,0,0};**
- **tmp** is a pointer to the first element of the array
- cannot use variable to define the size of the array
 - ex: you cannot say: **int x=10; double tmp[x];**
 - this won't compile, but you can say:
 - **#define x 10**, then **double tmp[x];**

char array (string)

- c has no string (class)
- a string is simply a “char array” with the last element being null ('\0')
- ex:
 - `char msg[]="hello\n";`

h	e	l	l	o	\n	\0
---	---	---	---	---	----	----

- `char msg[7]="hello\n"; //size 7 or larger`

- see `digit-count.c`

function

- call-by-value
 - the arguments are local variables whose values are copied from the callers
 - each function call allocates all these local variables which are placed on the top of the **call stack**
 - **ex:** long ans=fib(n); //in ex4.c
 - variable n in main function and variable n in fib function are different variables even though they have the same value.
 - **ex:** void swap(int a, int b); //won;t work
 - void swap(int * a, int * b); //need to use pointers

function

- Since array variables are pointers so:
 - `char A[]="GMU", B[]="UMD";`
 - `swap(A,B);` //call by value
 - `void swap(int X[], int Y[]){...}`
 - X will have address A
 - Y will have address B
- java is also "call-by-value" and "references" (i.e. pointers) are passed when arguments are objects
 - so, java does have pointers (references), but you cannot manipulate them

- see longest-line-1.c

scopes

- scopes
 - life span (global, local)
 - visibility (static, extern)
- Life span
 - variables **outside** all functions are global variables (has life span of the program)
 - variables **inside** a function is local to a **function call** (does not span different calls) unless “static” is used
 - `int foo(){ static int x=0; printf(“x=%d”,x++); }`
 - call foo multiple times will output different values

scopes

- Visibility (for global variables)
 - similar to private, protected, public in java/c++
 - **static** means “only visible to the file contains that variable”
 - **extern** means “visible to the entire program”
 - this is default for all global variables
 -

- see longest-line-1.c

Assignment

- see Lab02Spring12.txt