Week 3

Character input/output

- Stdio.h features
- getchar(): Reads a character from the standard input stream, returning it as an int. Returns EOF at the end of the file.
- Char:
    - 1 byte
    - ASCII encoding (portability concerns)
    - Enum types and switch statements
    - DFA approach to track whether program is in or out

Document properly

- Describe what function does from caller's viewpoint concisely

All variable declaration must appear at the beginning

Logical operators

- No separate boolean data type
- 0 is false, not 0 is true
- Relational operators (<, >) and logical operators (!, &&, ||) produce the result 0 or 1

Week 2

Git overview

Building and running C programs

Java vs C

- Java (safer)
    - null reference checking
    - Automatic array-bounds checking and memory management (garbage collection)
    - Other safety features
- C (faster)
    - Null pointer checking
    - Manual bounds and memory management

gcc217 charcount.c –o charcount

1. Preprocess:
    a. Command: gcc217 –E charcount.c > charcount.i
    b. Functionalities
        i. Removes comments
        ii. Expanding macros
        iii. Handles preprocessor directives
    c. Examples:
        i. Preprocessor replaces #include <stdio.h> with contents of /usr/include/stdio.h

        ii. EOF replaced with a value
2. Compile
    a. Command: gcc217 -S charcount.i
    b. Functionalities
        i. Check that input file is grammatically and type correct
        ii. Translate from C to assembly
    c. Example:
        i. Compiles sees function declarations
        ii. Give compiler enough info for future calls of getchar() and printf()
        iii. Will complain with a warning if no library preprocessor directive
3. Assemble
    a. Command: gcc217 -c charcount.s
    b. Functionality
        i. Translate assembly language to machine language
4. Link
    a. Command: Gcc -static charcount.o -o charcount
    b. Functionalities
        i. Resolve references within the code
        ii. Fetch machine code from standard C library
        iii. Produce final exec
    c. Examples:
        i. Contains definitions of getchar() and printf()
        ii. Static link: resolves all references
        iii. Dynamically link: resolves references except to shared objects
            1. Produces dynamic executable, stores metadata about where remaining references can be found

Deterministic finite automaton (DFA) logic
Enumeration type
- Statetype - states should have names

Modularity (different functions)

King Ch. 1-6, 7.3, 9, 14, 20.1
#include <stdio.h> necessary to “include” information about C’s standard I/O (input/output) library

```
int main(void)
{
        printf("To C, or not to C: that is the question.\n");
        return 0;
}
```

1. Preprocessing: like an editor, obeys commands that begin with # (known as directives)
2. Compiling: translates to machine instructions

    a. Cc pun.c
3. Linking: combines object code from compiler and additional code to execute program
    a. Cc -o pun pun.c (or gcc)

Exit function
Preprocessor

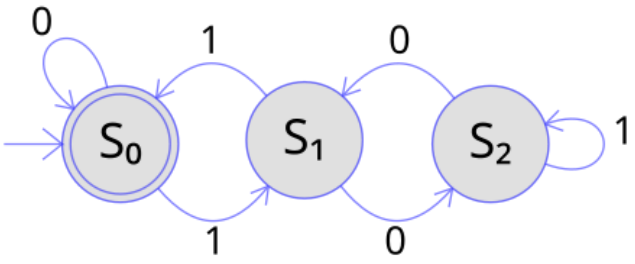
- Controlled by directives
    - #define for a macro: name that represents something else
        - Stores name of macro together with its definition
        - #define PI 3.14159
    - #include tells preprocessor to open a particular file
- Parameterized macros: function but operate in preprocessing time instead of runtime
    - #define MAX(x,y) ((x)>(y)?(x):(y))
    - :) faster and generic (can accept arguments of any type)
    - :(compiled code larger, arguments aren't type checked, can't point to macro, may evaluate arguments more than once
- # vs ## operator:
    - # converts the macro parameter into a string literal (stringification)
        - #define PRINT_VAR_NAME(var) printf(#var " = %d\n", var) → printf("x = %d\n", x);
            - X becomes the actual string b/c of the #var
    - ## concatenates (or "pastes") two tokens into a single token
        - #define CONCAT(a, b) a##b → int CONCAT(var, 1) = 10; → int var1 = 10
            - Var and 1 becomes new variable, var1, b/c of a##b
- Macros properties
    - Can call other macros
    - Preprocessor replaces ENTIRE token, not portions
    - Macros aren't local
    - May not be defined twice unless it's identical to old one
        - May be undefined by the #undef directive
- Directives
    - #if and #endif
    - Defined
        - #if defined (DEBUG) … #endif
            - Only included in program if DEBUG is defined as a macro
        - Or just use #ifdef or #ifndef
    - Applications
        - Adaptable to different operating systems and compilers
        - Default definition for a macro,
    - #error message
    - #line n "file"
    - #pragma tokens
        - Requests special behavior from compiler
        - _Pragma (string-literal) - same thing but destrings it

Bitwise operators

- << / >> left and right shift
    - i = 13; /* i is now 13 (binary 0000000000001101) */
    - j = i << 2; /* j is now 52 (binary 0000000000110100) */
- More operators
    - ~ complement
    - $ and
    - ^ exclusive or
    - | inclusive or
- Accessing bit fields
    - i = i & ~0x0070 | 0x0050; /* stores 101 in bits 4-6 */
    - i=(i&~0x0070)|(j<<4); /*stores in bits 4-6*/
- XOR encryption
    - 00100110 → XOR 01111010 → 01011100
    - (ASCII code for &) (ASCII code for z) (ASCII code for \)

Deterministic Finite Automaton (DFA)

- machine with finite states that reads an input string and either accepts or rejects it based on its state transitions
    - It consists of states, an alphabet, a transition function, a start state, and accepting states
- Deterministic (exactly one transition per input symbol in each state)



-

Week 1

Operating system:

- Piece of software that controls interaction between programs and hardware
- Also called “kernel”
    - Modern kernel examples:
        - unix linage (Linux, XNU)
        - vms lineage (windows NT)
    - Modern OS examples:
        - linux kernel (Linux/GNU, Android)
        - XNU kernel (MacOS, iOS)

- Windows NT kernel (windows)

GUI and CLI

- Graphical user interface: vv cutesy
- Command line interface: Text-based input/output

Terminal emulator: GUI → CLI and displays output

Shell: CLI or GUI for managing field and running other programs

- GUI examples: Mac finder, windows file mgr
- CLI: bash

Ssh: secure shell

- CLI program that connects sshd on another computer and replays text back + forth
- Sshd: program that runs continuously on a server, accepts network connections from ssh clients and relays with local shell

Text editor vs word processor vs IDE

- Plain text (textEdit) vs edit with formatting (Word, Pages) vs VSCode