Week Four: Expressions, statements, function calls, arrays of arrays,
How to do labs/projects well?

• Start early
  – So you have time to fully understand the requirement, prepare your test cases accordingly
    • This is an important skill in software engineering
    • The first task assigned to a new CS graduate is often to write test cases and testing procedures to test whether a piece of software functions as specified
    • I encourage the students to form study groups so they can provide test cases to each other

• Use techniques discussed in lectures/labs
  – Avoid using techniques seen on the web but not discussed in the lectures/labs yet
    • There are complexities not discussed yet
    • May misunderstand the techniques and get unexpected results
    • What you see on screen output may hide some “white spaces” and cause the result to differ from the correct ones

• Subscribe to Piazza email notice and read piazza regularly
Two things about expressions to be emphasized in this lecture

- Knowing the distinction between “lvalue” and “rvalue”
  - The compiler error messages often refer to such terminologies
- Knowing the importance of precedence among different operations
Objects and Lvalues

• An *Object* in C is a named region of *storage*;
• An *lvalue* is an expression referring to an object such that we are allowed to directly assign a content to it:

\[
lvalue = rvalue;
\]

– So, “l” means the left-hand side of the assignment, and “r” the right-hand side

• For example, array name *buf* alone cannot be an lvalue, we are not allowed to write *buf = 1*; or *buf++*
• We will go through a number of examples of operators and expressions
  – Point out if an expression can be an lvalue
  – These examples are grouped by their levels of precedence
    • Higher precedence first

• Note: *parentheses* override precedence
Array base address

• An array name alone provides the base address of that array
  – It can be viewed as a pointer expression, but
  – It cannot be used as an lvalue
    • We do not want to change the base address of an array!

• For an array of array $a[][[]$, $a[i]$ also provides the base address of an array
  – It cannot be used as an lvalue either
    • Cannot write $a[i] = 0$;
    • Cannot write $a[i]++$
Postfix Expressions

• There are a number of expressions whose operations are grouped by appending one to another.
• These are called postfix expressions in C.
  – The operators are at the same level of precedence, evaluated from left to right.
• The simplest postfix expression is the identifier of a variable.
• Next, Array references: \texttt{E1[E2]}
  – \texttt{E1} is any expression of a \textit{pointer} type or equivalent to a pointer type, like an array name
  – \texttt{E2} is any expression of an \textit{int} type
  – This reference is equivalent to the pointer dereference
    \[ * ( (E1) + (E2) ) \]
• We cannot write `buf = 'a';`
• But we can write `*buf = 'a';`
• And write `*(buf + 1) = 'b';`
• For array `x[][]`, we can write `*(x[0] + 1) = 'c';`
• For pointer `p`, we can write `p++[1] = 'x';`
/* PointerAsArray.c */
#include <stdio.h>

main() {
    int c = 0, in = 0;
    char buf[2048]; char *p = buf;
    char x[10][10];

    while((c = getchar()) != EOF)
        *p++ = c;
    *p++ = '\0';
    p = buf;
    *( buf + 1) = 'c';
    *(x[0] + 1) = 'd';
    p[0] = 'a';
    *buf = 'b';
    printf("\n**buf is \t %c\n", *buf);
    p++[0] = 'b';
    p++[0] = 'c';
    printf("p[0] is \t %c\n", p[0]);
    printf("p[1] is \t %c\n", p[1]);
    printf("p[2] is \t %c\n", p[2]);
    printf("p[0] address is \t %p\n", p);
    printf("x[0][1] is \t %c\n", x[0][1]);
    printf("buf address is \t %p\n", buf);
    printf("buffer is \t %s\n", buf);
}