EECS 2031 3.0 A Software Tools

Week 8: October 31, 2018

Large scale example

- RPN calculator
  - Infix versus prefix versus postfix expressions
  - 2+3*4 is complex to understand
  - 2 3 4 * + is not
- We (you) are going to build one (there is a simple one in the text).

RPN calculator

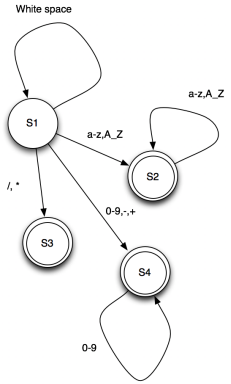
- Is basically a stack and a sequence of input tokens
  - If the token is a number, push it on the stack
  - If the token is an operator, pop the appropriate number of elements from the stack, and push the result
- You can even go farther and push code on the stack (see the language Forth, for an example).
- So most of the work involves processing the input and turning it into tokens of different types.

So the problems

- Lexical analysis - turning the input into tokens that have some meaning for the application
  - This is a very very common problem.
- Given a sequence of tokens, do the right thing for the right token type
  - Numbers - onto the stack
  - Everything else - each token has a meaning. Do the right thing for the right token.

Lexical analysis

- A real problem. We want to process some character sequence and decompose this into Tokens (primitive lexical components).
- We will assume that the process of combining the sequence into tokens can be modelled as a Regular Language (parsable by a deterministic finite state automaton). You remember these right?



Tokens
/
*****
allThis
+
-
+123
-123

Accept longest possible matching string
Maximum token size is limited

Observation

- Need to be able to peek (and put back) a character in the input. So lets write something that does this.

```
#include <stdio.h>
#include <stdlib.h>
#include "nextInputChar.h"

static FILE *fd;
static int lastChar = -1;

void setFile(FILE *d)
{
  fd = d;
}

int getChar()
{
  if(lastChar >= 0) {
    int temp = lastChar;
    lastChar = -1;
    return temp;
  }
  if(feof(fd))
    return -1;
  return fgetc(fd);
}

void ungetChar(int ch)
{
  if(lastChar >= 0) {
    fprintf(stderr,"ungetChar: max pushback is one character\n");
    exit(1);
  }
  if(ch < 0) {
    fprintf(stderr,"ungetChar: trying to push back a negative character\n");
    exit(1);
  }
  lastChar = ch;
}
```

Visible (and defined in the .h file)

```
void setFile(FILE *d);
int getChar();
void ungetChar(int ch);
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "nextInputChar.h"

static FILE *fd;
static int lastChar = -1;

void setFile(FILE *d)
{
  fd = d;
}

int getChar()
{
  if(lastChar >= 0) {
    int temp = lastChar;
    lastChar = -1;
    return temp;
  }
  if(feof(fd))
    return -1;
  return fgetc(fd);
}

void ungetChar(int ch)
{
  if(lastChar >= 0) {
    fprintf(stderr,"ungetChar: max pushback is one character\n");
    exit(1);
  }
  if(ch < 0) {
    fprintf(stderr,"ungetChar: trying to push back a negative character\n");
    exit(1);
  }
  lastChar = ch;
}
```

Not visible (local to this file)

Lexical analysis

- So the key primitive thing is a lexical token.
- Will have a type
- Will have some value
- So define it

```c
#define MAX_SYMBOL_LENGTH 129
#define LEX_TOKEN_EOF         1
#define LEX_TOKEN_IDENTIFIER  2
#define LEX_TOKEN_NUMBER      3
#define LEX_TOKEN_OPERATOR    4

struct lexToken {
  int kind;
  char symbol[MAX_SYMBOL_LENGTH];
};

void freeToken(struct lexToken *token);
struct lexToken *nextToken();
void dumpToken(FILE *fd, struct lexToken *token);
struct lexToken *allocToken();
```

Token ‘kinds’

Visible functions

lexical.h

```c
#include <stdio.h>
#include <stdlib.h>
#include "lexical.h"
#include "nextInputChar.h"

#define STATE_S1  1
#define STATE_S2  2
#define STATE_S3  3
#define STATE_S4  4

/* function to allocate a new token */
struct lexToken *allocToken()
{
  struct lexToken *token;

  if((token = (struct lexToken *) malloc(sizeof(struct lexToken))) == (struct lexToken *)NULL) {
    fprintf(stderr,"nextToken: out of memory, aborting\n");
    exit(1);
  }
  token->symbol[0] = '\0';
  return token;
}

void freeToken(struct lexToken *token)
{
  (void) free(token);
}

void dumpToken(FILE *fd, struct lexToken *token)
{
  if(token == (struct lexToken *) NULL)
    fprintf(fd, "dumpToken: null token\n");
  else {
    switch(token->kind) {
    case LEX_TOKEN_EOF :
      fprintf(fd, "dumpToken: EOF token\n");
      break;
    case LEX_TOKEN_IDENTIFIER:
      fprintf(fd, "dumpToken: identifier token |%s|\n", token->symbol);
      break;
    case LEX_TOKEN_OPERATOR:
      fprintf(fd, "dumpToken: operator token |%s|\n", token->symbol);
      break;
    case LEX_TOKEN_NUMBER:
      fprintf(fd, "dumpToken: number token |%s|\n", token->symbol);
      break;
    default:
      fprintf(fd, "dumpToken: bad token type %d\n", token->kind);
    }
  }
}
```

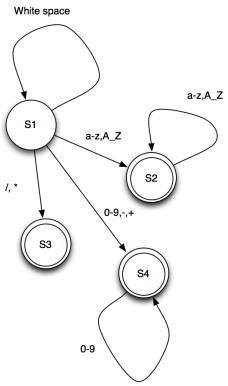
Private defines

lexical.c (part 1)

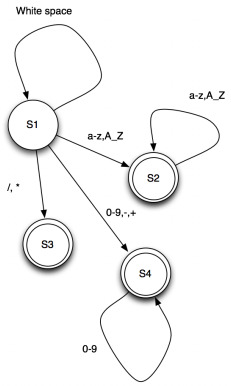
Visible functions

```
struct lexToken *nextToken()
{
  int c;
  struct lexToken *token;
  int symbolLength;
  int state = STATE_S1;

  token = allocToken();
  while(1) {
    switch(state) {
    case STATE_S1:
      switch(c = getChar()) {
      case -1: /* EOF */
        token->kind = LEX_TOKEN_EOF;
        return token;
      case ' ':
      case '\t':
      case '\n':
        break;
      case '/':
      case '*':
        token->symbol[0] = c;
        symbolLength = 1;
        state = STATE_S3;
        break;
      default:
        if(((c >= 'a') && (c <='z'))||((c >= 'A') && (c <= 'Z'))) {
          token->symbol[0] = c;
          symbolLength = 1;
          state = STATE_S2;
        } else if((c == '+')||((c == '-')||((c >= '0') && (c <= '9')))) {
          token->symbol[0] = c;
          symbolLength = 1;
          state = STATE_S4;
        } else
          fprintf(stderr,"nextToken: bad char %c in input stream, ignored\n",c);
      }
      break;
```




```
    case STATE_S2: /* identifier */
      c = getChar();
      if(((c >= 'a') && (c <= 'z'))||((c >= 'A') && (c <= 'Z'))) {
        if(symbolLength == MAX_SYMBOL_LENGTH-1) {
          fprintf(stderr,"nextToken: symbol too long, aborting\n");
          exit(1);
        }
        token->symbol[symbolLength++] = c;
      } else {
       ungetChar(c);
       token->kind = LEX_TOKEN_IDENTIFIER;
       token->symbol[symbolLength] = '\0';
       return token;
      }
      break;
    case  STATE_S3: /* operator */
      token->kind = LEX_TOKEN_OPERATOR;
      token->symbol[symbolLength] = '\0';
      return token;
    case STATE_S4: /* number */
      c = getChar();
      if((c >= '0') && (c <= '9')) {
        if(symbolLength == MAX_SYMBOL_LENGTH-1) {
          fprintf(stderr,"nextToken: symbol too long, aborting\n");
          exit(1);
        }
        token->symbol[symbolLength++] = c;
      } else {
        ungetChar(c);
        if((symbolLength == 1) && ((token->symbol[0] == '-')||(token->symbol[0] == '+'
          token->kind = LEX_TOKEN_OPERATOR;
        else
          token->kind = LEX_TOKEN_NUMBER;
        token->symbol[symbolLength] = '\0';
        return token;
      }
      break;
    default:
      fprintf(stderr,"nextToken: can't happen\n");
      exit(1);
    }
  }
}
```



Dealing with operations

- So we have a token, we could have an incredibly large if statement to deal with this.
  - if(!strcmp(token.symbol, “+”)){ - but this is very boring and difficult to debug.

```
static int op_quit(struct tokenStack *stack);
static int op_print(struct tokenStack *stack);
static int op_dump(struct tokenStack *stack);
static int op_add(struct tokenStack *stack);

static struct operator_struct {
  char *name;
  int (*fn_ptr)(struct tokenStack *);
} ops[] = {
  {"quit", op_quit},
  {"print", op_print},
  {"dump", op_dump},
  {"+", op_add},
  {(char *)NULL, (int(*)(struct tokenStack *)) NULL}
};
```

Mapping from string to command to execute

```
static int op_quit(struct tokenStack *stack);
static int op_print(struct tokenStack *stack);
static int op_dump(struct tokenStack *stack);
static int op_add(struct tokenStack *stack);

static struct operator_struct {
  char *name;
  int (*fn_ptr)(struct tokenStack *);
} ops[] = {
  {"quit", op_quit},
  {"print", op_print},
  {"dump", op_dump},
  {"+", op_add},
  {(char *)NULL, (int(*)(struct tokenStack *)) NULL}
};
```

Null at the end so we know that we are at the end

```
int doOperator(struct tokenStack *stack, char *o)
{
  struct operator_struct *op = ops;
  for(op=ops;op->name != (char *)NULL; op++) {
    if(!strcmp(op->name, o))
      return op->fn_ptr(stack);
  }
  return(-1);
}
```

Linear search through list

```
static struct operator_struct {
  char *name;
  int (*fn_ptr)(struct tokenStack *);
} ops[] = {
  {"quit", op_quit},
  {"print", op_print},
  {"dump", op_dump},
  {"+", op_add},
  {(char *)NULL, (int(*)(struct tokenStack *)) NULL}
};
```

```
static int op_add(struct tokenStack *stack)
{
  int v1, v2;
  v1 = popInt(stack);
  v2 = popInt(stack);
  pushInt(stack, v1+v2);
  return(0);
}
```

Next lab

- So you get to complete this.
- Partial code on GitHub (michaeljenkin/rpncalc)
- See the lab for full details.

1. If a macro MX(A,B) is defined in C as

```
#define MX(A,B) (A>B)?A:B
```

then the C expression

```
printf("%d %d",MX(3, 2) + 3,MX(2, 3) + 3)
```

will output

A. 3 3
B. None of the other answers are correct
C. 0 0
D. 6 6
E. 3 6

2. To compile and link a source file foo.c against a library libwiringPi.a located in /opt/lib with an include file libWiringPi.h located in /opt/include, producing an executable file foo.out, one would execute

A. It is not possible to compile foo.c
B. gcc −lWall −ansi −pedantic foo.c −lwiringPi −o foo
C. None of the other answers are correct
D. gcc −lWall −ansi −pedantic foo.c −o foo
E. gcc −lWall −ansi −pedantic −I/opt/include −L/opt/lib −lwiringPi foo.c −o foo

1. If a macro MX(A,B) is defined in C as

```
#define MX(A,B) (A>B)?A:B
```

then the C expression

```
printf("%d %d",MX(3, 2) + 3,MX(2, 3) + 3)
```

will output

A. 3 3
B. None of the other answers are correct
C. 0 0
D. 6 6
E. 3 6

2. To compile and link a source file foo.c against a library libwiringPi.a located in /opt/lib with an include file libWiringPi.h located in /opt/include, producing an executable file foo.out, one would execute

A. It is not possible to compile foo.c
B. gcc −lWall −ansi −pedantic foo.c −lwiringPi −o foo
C. None of the other answers are correct
D. gcc −lWall −ansi −pedantic foo.c −o foo
E. gcc −lWall −ansi −pedantic −I/opt/include −L/opt/lib −lwiringPi foo.c −o foo

3. In ANSI C, comments

A. are unnecessary as ANSI C is self−documenting
B. start with /* and end with */
C. None of the other answers are correct
D. can start with /* and end with */ or start with two forward slashes and continue until the end of the line
E. start with two forward slashes and continue until the end of the line

4. The real datatype in C

A. represents all numbers between 0.0 and 1.0 inclusive
B. approximates the real numbers
C. does not exist
D. is a 32 bit long real value
E. is a 64 bit long real value

5. What is the output of this program fragment

```
char *str = "Blue";
int a = 100;
printf(a >= 37 ? "Green" : "%s", str);
```

A. Green
B. 100
C. str
D. Blue

E. a >= 37 ? "Green"

6. The minimum number of times that the body of the "do while" loop is guaranteed to be executed is

A. variable
B. None of the other answers are correct
C. 0
D. infinity
E. 1

7. In C the #include directive is used to

A. include pre−compiled library files into C files as they are compiled
B. include pre−compiled C source files into C files as they are compiled
C. include Java .class files into C files as they are compiled
D. None of the other answers are correct
E. include header files into C source files as they are compiled

8. In C, consider the following program fragment which involves calling three functions f1, f2 and f3:

```
a = f1(23,14) * f2(37) + f(0);
```

The order in which the three functions f1, f2 and f3 will be called is given by

A. f2 then f1 then f3
B. f3 then f1 then f2
C. It is impossible to say
D. f1, then f2, then f3
E. None of the other answers are correct

9. In C, how large is the float datatype?

A. exactly 8 bytes
B. Exactly 2 bytes
C. None of the other answers are correct
D. Exactly 4 bytes
E. It depends on the system.

10. Consider the following C code fragment

```
int x = 1;
while(x);
  printf("%d\n",x--);
```

This code will print out

A. None of the other answers are correct
B. 1 0 -1
C. 1 0
D. nothing
E. 1

11. Consider the following C code fragment

```
#include <stdio.h>
void swap(int a, int b)
{
  int t = a;
  a = b;
  b = t;
}

int main(int argc, char **argv)
{
   int x = 2;
   int y = 3;
   swap(x,y);
   printf("%d %d\n",x,y);
   return 0;
}
```

This code outputs

A. 3 3

B. 2 3

C. 3 2

D. 2 2

E. None of the other answers are correct

12. In C the #define directive is used to

A. include header files into C source files as they are compiled

B. define system independent values only

C. define constants and macros

D. None of the other answers are correct

E. define external signals and system calls

13. If you are currently in the directory /usr/john which of the following shell commands does not change your directory from its current location

A. cd ..

B. cd .

C. cd /usr

D. cd /

E. None of the other answers are correct

14. The web server provided in Lab04 ...

A. None of the other answers are correct

B. sends an email when a message is sent

C. responds to motion in the environment

D. simulates an IFTTT connection

E. requires a resistor in the circuit to operate

15. In C a pointer variable is a variable that

A. None of the other answers are correct

B. stores the index of an array

C. stores the address of another item

D. stores the name of another variable

E. stores the value of another item

16. Consider the following C program fragment

```
int *p; p = (int *) malloc(sizeof(int)*10);
*p = 6;
free(p);
printf("%d", *p);
```

This code will output

A. -1

B. None of the other answers are correct

C. 6

D. 0

E. It is not possible to say

17. Consider the following C program fragment.

```
int x; x=1,2,3;
printf("%d",x);
```

What does this fragment output?

A. 1,2,3

B. 1

C. 3

D. Nothing. The fragment contains a syntax error.

E. 2

18. A Makefile is

A. a textfile used by the 'make' command to automate the process of building some output, typically some target application or library

B. an executable file that is used to build target applications or libraries

C. None of the other answers are correct

D. an IDE that is used to build C programs

E. an executable file that is used to invoke an IDE

19. Suppose you have a Makefile to compile foo.c producing foo.out, as given below

```
OBJS=foo.o
CC=gcc
CFLAGS=-Wall -pedantic
RM=rm -f

foo.out: $(OBJS)
	$(CC) $(CFLAGS) $(OBJS) -o foo.out

%.o:	%.c
	$(CC) $(CFLAGS) -c -ansi $<

cleam:
	$(RM) *.out *.o core
```

Typing "make clean" will

A. compile foo.c into foo

B. compile foo.c into foo.out

C. None of the other answers are correct

D. cause all .o files, all files ending in .out and any core file to be removed

E. compile foo.c into a.out

20. To compile a C program in a file foo.c resulting in an executable file foobar, one would execute the following command line using gcc

A. gcc foo.c –o foobar.out

B. gcc –c foo.c –o foobar

C. gcc foo.c –o foobar

D. gcc foo.c –o foobar.c

E. gcc foo.c

E. gpk pkkk

21. Consider the following C program fragment

```
int foo(int x, float y) { return 1; }
```

This fragment

A. defines a function foo of two or more arguments that returns an int.
B. defines a function foo of two arguments that returns an int. The function takes an int and float as parameters.
C. provides a forward definition for a function foo of two arguments that returns an int. The function takes an int and float as parameters.
D. None of the other answers are correct.
E. provides a forward definition for a function foo of two or more arguments tnat returns an int.

22. Consider the two C program fragments below. Fragment I

```
int i;
i = 0;
while (i < 10)
{
  printf("statement1\n");
  continue;
  printf("statement2\n");
  i++;
}
```

and Fragment II


```
int i;
i = 0;
while (i < 10)
{
  printf("statement1\n");
  break;
  printf("statement2\n");
  i++;
}
```

Notwithstanding any warnings that may be generated by the compiler, the actions of Fragment I and Fragment II can be described as

A. Fragment I prints out 'statement 1' once, Fragment II loops forever printing out 'statement 1' in each loop.
B. Fragment I prints out 'statement 1' once, Fragment II prints out 'statement 1' once.
C. None of the other answers are correct
D. Fragment I loops forever printing out 'statement 1' in each loop, Fragment II outputs 'statement 1' once.
E. Fragment I loops forever printing out 'statement 1' in each loop, Fragment II loops forever printing out 'statement 1' in each loop.

23. The minimum number of times that the body of the "for" loop is guaranteed to be executed is

A. None of the other answers are correct
B. 1
C. 0
D. infinity
E. variable

24. In C, which of the following is not a valid variable declaration?

A. float 3_a;
B. float __3;
C. float _a;
D. None of the other answers are correct
E. float a_a;

25. What is the output of this C program fragment?

```
do
  printf("In loop ");
while(1);
printf("Out of loop\n");
```

A. In loop Out of loop
B. It prints nothing
C. It prints out an infinite number of copies of "In loop"
D. None of the other answers are correct
E. Out of loop

26. When connecting a LED to a GPIO output pin using a Raspberry Pi, it was necessary to insert a resistor in the circuit in order to

A. limit the current passing through the LED
B. provide a simple electrical connection to ground in your circuit
C. None of the other answers are correct
D. reduce the possibility of short circuits between the LED and the resistor
E. stop electricity running backwards to the Raspberry Pi

27. In Linux, the command apt−get upgrade

A. None of the other answers are correct
B. Compiles a C program producing an executable
C. Installs newer versions of the packages you have.
D. Logs you out of the shell.
E. Updates the list of available palcages and their versions, but does not install them.

27. In Linux, the command apt−get upgrade

A. None of the other answers are correct
B. Compiles a C program producing an executable
C. Installs newer versions of the packages you have.
D. Logs you out of the shell.
E. Updates the list of available palcages and their versions, but does not install them.

28. Which of the following is consistent with the precidence of arithmetic operators in C from highest to lowest?

A. / * + −
B. * + − /
C. + * / −
D. / + * −
E. None of the other answers are correct

29. Consider the following C program fragment

```
int x = 3, y = 7, z = 7;
printf("%d %d\n",x=z,y==z);
```

This fragment will output

A. Nothing, there is a syntax error
B. 1 1
C. 7 7
D. 7 1
E. 1 7

30. Given the following C program fragment

```
int a = 16777215;
char *s = (char *) &a;
*s = 12;
```

then afterwards

A. None of the other answers are correct
B. the value of a is 12
C. the value of a is 0
D. the value of a is 16777215
E. the value of a is machine dependent

31. Suppose you have a project stored in a GitHub repository and locally you have changed a file foo.c and you wish to update the master branch with the changes in foo.c and push all of these changes to GitHub. Assuming that no one else is editing this project, except you on the local machine, you would execute the following

A. It is not possible to do this
B. git commit −m "making edits"; git push
C. None of the other answers are correct
D. git push; git commit −m "making edits"; git add
E. git add; git commit −m "making edits"; git push

32. What is the output of this C program fragment?

```
int x = 1;
do
  printf("%d ", x);
while(++x<=1);
```

A. No output
B. 1 2
C. Compile time error
D. 1
E. None of the other answers are correct

33. In C, header (.h) files are primarily used to

A. share common definitions of functions, variables and macros across C source files (.c files)
B. define local funtions and local variables
C. break up C source files into smaller compilation units
D. define local functions and global variables
E. define global variables and local functions

34. Consider the following C program fragment

```
int f(int a, int b)
{
  int a;
  a = 20;
  return a;
}
```

This code will not compile. What error will be detected by the compiler?

A. The function f should be defined as returning an int.
B. The argument a is redeclared within the body of the function.
C. The argument b is not used.
D. The function f must have a prototype definition.
E. The function f should be defined as returning a float.

35. When you compile using gcc's –Wall flag,
    A. you cause the compiler to be consistent with the ANSI compiler standard

    B. None of the other answers are correct
    C. you cause the compiler to generate code to check for invalid memory accesses
    D. you cause the compiler to enable checking for all warnings
    E. you cause the compiler to write all warnings to the global wall

36. In C, which of the following lists only unary operators (separated by commas)?
    A. sizeof, ~, !
    B. ->, ~, &&
    C. ++, %, @
    D. None of the other answers are correct
    E. ||, |, &&, &

37. Consider the following C program fragment.

```
int x=3; (x&1) ? printf("blue") : printf("green");
```

    What does this fragment output?
    A. nothing
    B. blue
    C. green blue
    D. green
    E. blue green

38. After the execution of the following C code fragment,

```
int x=6;
int i;
for(i=0;x<5;i++) {
  x =x+i;
}
```

    x has the value
    A. "hello world"
    B. 6
    C. 11
    D. 256
    E. -1

39. Which of the following statements about ANSI C is correct
    A. The size of an int pointer will always be 32 bits
    B. None of the other answers are correct
    C. C variable names can have arbitrary length and all characters in the name are considered when comparing names for equality
    D. The C language guarantees the order in which all evaluations will take place
    E. When compiling code in different source files, the full length of global names will be used when resolving external definitions.

40. Of the following list; do, while, for, int, float, Double. Which are not reserved words in C ?

    A. Double
    B. All of the words in the list are reserved words in C
    C. None of the other answers are correct
    D. int, float, Double
    E. int, float

41. If you wanted to define a function void foo() in the file one.c such that it was not accessible outside of the file one.c, you would define the function foo() with a definition that begins with
    A. static void foo()
    B. void foo(static)
    C. foo()
    D. None of the other answers are correct
    E. void foo()

42. Consider the following C code fragment

```
int a = 1;
if(a)
  printf("ONE ");
  printf("TWO ");
else
  printf("THREE");
printf("\n");
```

    This code will output
    A. THREE
    B. ONE
    C. TWO THREE
    D. ONE THREE
    E. Nothing, there is a syntax error

43. Given the following C program fragment

```
int x = 2;
int z = 5;
int *ptr;
ptr = &x;
*ptr = 3;
```

    then afterwards
    A. ptr has the value 3
    B. x has the value 3
    C. z has the value 3
    D. *ptr has the value 2
    E. x has the value 2

44. The Makefile


```
CC=gcc
CFLAGS=-lWarn -pedantic

tester: tester.o libmyifttt.a
	$(CC) tester.o -L. -lmyifttt -lcurl -o tester

libmyifttt.a: ifttt.o ifttt.h
	ar -rcs libmyifttt.a ifttt.o

ifttt.o:  ifttt.c ifttt.h
	$(CC) $(CFLAGS) -c -ansi $<

tester.o: tester.c ifttt.h
	$(CC) $(CFLAGS) -c -ansi $<
```

    Encodes that the target tester depends on
    A. The library libmyifttt.a and tester.o, which eventually depend on tester.c ifttt.c and ifttt.h
    B. ifttt.h only
    C. tester.c ifttt.c ittester.c only
    D. tester.c only
    E. tester.c ifttt.c only

45. If in file one.c there is the definition of a global variable float x, then to reference this global variable in another file two.c the definition of x there would be
    A. It is not possible to reference x from two.c
    B. extern float x;
    C. float x;
    D. None of the other answers are correct
    E. extern x;

46. What is the output of this C program fragment?

```
int x = 1;
do
  printf("%d ", x);
while(x++<=1);
```

    A. 1 2
    B. No output
    C. None of the other answers are correct
    D. 1
    E. Compile time error

46. What is the output of this C program fragment?

```
int x = 1;
do
  printf("%d ", x);
while(x++<=1);
```

A. 1 2
B. No output
C. None of the other answers are correct
D. 1
E. Compile time error

47. Consider the following C code fragment running on a machine that supports the ASCII character set

```
char ch; running on a machine
that supports the ASCII character set
int i;
ch = 'F';
i = ch - 'A';
printf("%d", i);
```


This snippet will output
A. 6
B. 5
C. 3
D. None of the other answers are correct
E. 4

48. Using GPIO pins on the Raspberry Pi for input, you used the code

```
pullUpDnControl(1, PUD_UP);
```

on pin 1 in order to
A. assign the pin 1 as output
B. None of the other answers are correct
C. ensure that the voltage on the input line is ground if it is not connected to +3.3V
D. ensure that the voltage on the input line is +3.3V if it is not connected to ground
E. assign the pin 1 as input

49. Consider the following C code fragment

```
int i = 10, j = 010;
printf("%d %d\n",i,j);
```

This code will output
A. 80 10
B. 10 160
C. 10 8
D. None of the other answers are correct
E. 10 80

50. To define a Makefile macro COMPILE that expands to gcc, one would define the macro as
A. COMPILE=g++
B. COMPILE=gcc
C. CC=g++
D. None of the other answers are correct
E. CC=gcc