int \*\*\*array;

and we want the dimensions to be 10\*20\*30 (all of the stuff below could be done for an arbitrary i,j,k, which is closer to what you need).

First, we need an array of 10 int \*\*'s, so we use the following:

array = (int \*\*\*) malloc( 10 \* sizeof(int \*\*) );

The *sizeof* function returns an integer telling how many bytes are needed by something of type "int \*\*", and we need 10 of them.  The "(int \*\*\*)" is a cast which changes the pointer type from "char \*" to "int \*\*\*", to keep the types correct.  Don't forget that, after this call to *malloc*, you should check to see if array==NULL.

Note:  *malloc* is asked for 10 int \*\*'s but its return is a pointer to them, so the result is an int \*\*\*.

Now that we have the 10 pointers, we can get the next level of pointers:

for ( i = 0 ; i < 10 ; ++i ) {

array[i] = (int \*\*) malloc( 20 \* sizeof(int \*) );

}

And finally, we can fill in each of these pointers with an array of 30 integers:

for ( i = 0 ; i < 10 ; ++i ) {

for ( j = 0 ; j < 20 ; ++j ) {

array[i][j] = (int \*) malloc( 30 \* sizeof(int) );

}

}

Again, remember that each call to malloc must check the result.  Also note that we could have put the two steps above together, filling each set of 20 pointers as we get them.

It is *much* more efficient to combine all similar allocations and divide up the memory after getting it.  (It is also *much* easier to make mistakes.)  When you've convinced yourself the the following works, you will understand C pointers fairly well.

array = (int \*\*\*) malloc( 10 \* sizeof(int \*\*) );

array[0] = (int \*\*) malloc( 10 \* 20 \* sizeof( int \*) );

array[0][0] = (int \*) malloc( 10 \* 20 \* 30 \* sizeof(int) );

for ( j = 1 ; j < 20 ; ++j ) {

array[0][j] = array[0][j-1] + 30;

}

for ( i = 1 ; i < 10 ; ++i ) {

array[i] = array[i-1] + 20;

array[i][0] = array[i-1][20-1] + 30;

for ( j = 1 ; j < 20 ; ++j ) {

array[i][j] = array[i][j-1] + 30;

}

}

This is the method you should use, but you will probably only need two dimensional arrays, which are far easier once you understand this example.

Space is returned to the system with the command

free( pointer );

For returning the space to the system, you *always* return exactly what you were given, *i.e.*, the exact pointer that *malloc* gave you.  You cannot return a portion of an allocation.  Thus there should be a perfect one-to-one correspondence between calls to *malloc* and calls to *free*.

HOMEWORK 1

|  |  |
| --- | --- |
| 1.  man -K <str>  Searches all man pages for a specific string, according to the man man page.  2.  which <program>  Prints path to program.  $which cp  /bin/cp  $which wc  /usr/bin/wc  3.  Programs are located in /usr/bin, so we can find programs with one letter names using:  find /usr/bin -name "?"  since the "?" character is a wildcard that stands for any character.  /usr/bin/w  /usr/bin/[  /usr/bin/X  To determine what each of these programs does, we may use man or help commands:  man <program>  help <program>  w  Show who is logged on and what they are doing.  [  Execute conditional command.  X  a portable, network-transparent window system  4.  /etc/alternatives/emacs  the readlink command reads the value of a symbolic link  readlink /usr/bin/emacs  5.  g+s  Allows other users in the file's group to set user or group ID on execution.  o-x  Gives the same permissions granted to users that are neither the user who owns the file nor  the users who are members of the file's group (o) but not permission to execute (-x).  6.  find -mtime 21  from find's man page description of -mtime n:  File's data was last modified n\*24 hours ago. 21 days = 3 weeks.  7.  find -type d -mtime -21  -type d specifies directory.  .  ./Desktop  ./Desktop/CS35L  ./Desktop/CS35L/Lab 1  ./Desktop/CS 33  ./Desktop/CS 33/Lab 4  ./Documents  ./Downloads  ./.emacs.d/auto-save-list  12.  C-h k <command>  gives a description of each command.  C-M-a  move-beginning-of-line: move point to beginning of current line  C-M-b  backward-char: move point left N characters  C-M-c  exit-recursive-edit: Exit from the innermost recursive edit or minibuffer  C-M-d  delete-char  C-M-e  end-of-defun: Move forward to next end of defun  C-M-f  forward-sexp: Move forward across one balanced expression (sexp)  C-M-g  info on C-g shows up; C-M-g doesn't exist  C-M-h  mark-defun: Put mark at end of this defun, point at beginning  13.  from emacs (C-h k C-g):  C-g runs the command keyboard-quit, which is an interactive compiled  Lisp function.  C-g causes emacs to quit while executing Lisp code.  14.  C-h f yank  yank reinserts "the last stretch of killed text most recently killed OR yanked" and  places the "point at end" and sets "mark at beginning".  15.  To invoke yank function on /usr/bin:  C-x d .. (however many times necessary to get to /)  C-x d bin  To invoke ls -l on /usr/bin:  cd /usr/bin  ls -l  Differences:  1) dired identifies total used in directory 9504 available 8658372 while ls -l only identifies total 9496.  2) dired lists  dr-xr-xr-x 2 root root 4096 Oct 3 13:33 .  dr-xr-xr-x 29 root root 4096 Sep 15 17:22 ..  which are absent in ls -l  3) not a significant difference, but dired lists names in green while ls -l lists them in default color. | 8.  to locate the directory containing find:  which find  /bin/find  man find tells us that -type l will find all files that are symbolic links.  find -type l  /bin/ksh  /bin/nisdomainname  /bin/gtar  /bin/ex  /bin/dnsdomainname  /bin/sh  /bin/csh  /bin/domainname  /bin/fgrep  /bin/awk  /bin/traceroute6  /bin/rvi  /bin/ypdomainname  /bin/egrep  /bin/mail  /bin/rview  /bin/iptables-xml  /bin/iptables-xml-1.4.7  /bin/view  /bin/rnano  for a total of 20 files.  9.  to list contents of a directory sorted by modification time (from ls man):  ls -t  for long-listing pattern needed to determine which contents are regular files:  ls -l  ls -l -t  -rwxr-xr-x 1 root root 28776 Mar 3 2010 libgamin-1.so.0.1.10  This is the last file listed and is a regular file, making it the oldest regular file.  10.  from man locale, we find that locale draws its information from the environment variables.  11.  C-h a sort  lists all commands with the keyword sort  rmail-sort-by-author M-x ... RET  Sort messages of current Rmail buffer by author.  rmail-sort-by-correspondent M-x ... RET  Sort messages of current Rmail buffer by other correspondent.  rmail-sort-by-date M-x ... RET  Sort messages of current Rmail buffer by "Date" header.  rmail-sort-by-labels M-x ... RET |

ASSIGNMENT 3 PYTHON

|  |  |
| --- | --- |
| #!/usr/bin/python  import locale, string, random, sys  from optparse import OptionParser  class comm:  def \_\_init\_\_(self, file1, file2):  try:  if file1 == "-" and file2 == "-":  print("Pass at least one file!")  exit()  elif file1 == "-":  fin1 = sys.stdin  fin2 = open(file2, 'r')  elif file2 == "-":  fin2 = sys.stdin  fin1 = open(file1, 'r')  else:  fin1 = open(file1, 'r')  fin2 = open(file2, 'r')  self.lines1 = fin1.read().split('\n')  self.lines2 = fin2.read().split('\n')  del self.lines1[len(self.lines1)-1]  del self.lines2[len(self.lines2)-1]  self.masterList = []  self.contents = []  self.tracker1 = []  self.tracker2 = []  fin1.close()  fin2.close()  except IOError as e:  errno = e.errno  strerror = e.strerror  parser.error("I/O error({0}): {1}".  format(errno, strerror))  def isSorted(self, file):  for k in range(len(file) - 1):  if locale.strcoll(file[k],file[k+1]) > 0:  print ("File %s is unsorted. Use -u to run on unsorted files.") % file  exit()  return True  def compareSorted(self):  max1 = len(self.lines1) #DOES THIS WORK?  max2 = len(self.lines2)  i = 0  j = 0  if self.isSorted(self.lines1) and self.isSorted(self.lines2):  self.masterList.append(self.lines1[i] + "\n")  if self.tracker1[i]:  self.contents.append(3)  else:  self.contents.append(1)  i += 1  while j < len(self.lines2): #ADD LINES1 ELEMENTS TO MASTERLIST  if self.tracker2[j]:  pass  else:  self.masterList.append(self.lines2[j] + "\n")  self.contents.append(2)  j += 1  def printout(self, op1, op2, both):  if not op2: #DETERMINE COLUMN 2 INDENT  if op1:  indent1 = ""  else:  indent1 = " "  else:  indent1=""  if not both: #DETERMINE COLUMN 3 INDENT  if op1 and op2:  indent2 = ""  elif not op1 and not op2:  indent2 = " "  else:  indent2 = " "  else:  indent2 = ""  for i in range(len( self.masterList)): #ADD INDENTS  if self.contents[i] == 1:  pass  elif self.contents[i] == 2:  self.masterList[i] = indent1 + self.masterList[i]  elif self.contents[i] == 3:  self.masterList[i] = indent2 + self.masterList[i]  else:  pass  for i in range(len( self.masterList)): #PRINT THEM OUT  if not op1 and self.contents[i] == 1:  sys.stdout.write(self.masterList[i])  elif not op2 and self.contents[i] == 2:  sys.stdout.write(self.masterList[i])  elif not both and self.contents[i] == 3:  sys.stdout.write(self.masterList[i])  else:  pass    def main():  version\_msg = "%prog2.0"  usage\_msg = """%prog [OPTION]... FILE1 FILE2  Compare files line by line, like linux `comm` command"""  parser = OptionParser(version=version\_msg, usage=usage\_msg) | while True:  if i == max1 and j == max2: #HIT THE END  break  elif i == max1: #FIRST LIST AT END, FINISH UP SECOND  self.masterList.append(self.lines2[j] + "\n")  j += 1  self.contents.append(2)  elif j == max2: #SECOND LIST AT END, FINISH UP FIRST  self.masterList.append(self.lines1[i] + "\n")  i += 1  self.contents.append(1)  elif self.lines1[i] == self.lines2[j]: #BOTH EQUAL, ADD TO THIRD COLUMN  self.masterList.append(self.lines1[i] + "\n")  i += 1  j += 1  self.contents.append(3)  elif self.lines1[i] < self.lines2[j]: #FIRST IS LESS, ADD TO FIRST COLUMN  self.masterList.append(self.lines1[i] + "\n")  i += 1  self.contents.append(1)  elif self.lines1[i] > self.lines2[j]: #SECOND IS LESS, ADD TO SECOND COLUMN  self.masterList.append(self.lines2[j] + "\n")  j += 1  self.contents.append(2)  else:  pass  def compareUnsorted(self):  for i in range(len(self.lines1)): #FILL WITH VALUES  self.tracker1.append(False)  for j in range(len(self.lines2)): #FILL WITH VALUES  self.tracker2.append(False)  for i in range(len(self.lines1)):  for j in range(len(self.lines2)):  if self.lines1[i] == self.lines2[j] and not self.tracker1[i] and not self.tracker2[j]:  self.tracker1[i] = True  self.tracker2[j] = True  break  i = 0  j = 0  while i < len(self.lines1): #ADD LINES1 ELEMENTS TO MASTERLIST  parser.add\_option("-1", action="store\_true", dest="op1",  default=False,  help="suppress column 1 (lines unique to FILE1)")  parser.add\_option("-2", action="store\_true", dest="op2",  default=False,  help="suppress column 2 (lines unique to FILE2)")  parser.add\_option("-3", action="store\_true", dest="both",  default=False,  help="suppress column 3 (lines that appear in both files)")  parser.add\_option("-u", action="store\_true", dest="unsorted",  default=False,  help="compare files even if unsorted")  options, args = parser.parse\_args(sys.argv[1:])  try:  op1 = bool(options.op1)  op2 = bool(options.op2)  both = bool(options.both)  unsorted = bool(options.unsorted)  except:  parser.error("invalid option type: {0}".  format(options.op1))    if len(args) != 2:  parser.error("indicate 2 files or '-' for stdin")  infile1 = args[0]  infile2 = args[1]  try:  compObject = comm(infile1, infile2)  if unsorted:  compObject.compareUnsorted()  else:  compObject.compareSorted()  compObject.printout(op1, op2, both)  except IOError as e:  errno = e.errno  strerror = e.strerror  parser.error("I/O error({0}): {1}".  format(errno, strerror))  if \_\_name\_\_ == "\_\_main\_\_":  main() |

LAB 4 DIFF FILE

-

2017-10-24 Jonathan Wei-Hong Chu <weihong@lnxsrv05.seas.ucla.edu>

\* timespec.h (timespec\_cmp): Comparison done by if statements to

imitate output of ? : subtraction

--- timespec.h 2005-09-21 23:05:39.000000000 -0700

+++ timespec1.h 2017-10-24 17:17:20.409581000 -0700

@@ -45,10 +45,13 @@

static inline int

timespec\_cmp (struct timespec a, struct timespec b)

{

- int diff = a.tv\_sec - b.tv\_sec;

- return diff ? diff : a.tv\_nsec - b.tv\_nsec;

+ if (a.tv\_sec < b.tv\_sec)

+ return -1;

+ else if (b.tv\_sec < a.tv\_sec)

+ return 1;

+ else

+ return 0;

}

-

# if ! HAVE\_DECL\_NANOSLEEP

/\* Don't specify a prototype here. Some systems (e.g., OSF) declare

nanosleep with a conflicting one (const-less first parameter). \*/

LAB 4 SFROB.C

|  |  |
| --- | --- |
| #include<stdio.h>  #include<stdlib.h>  int frobcmp(char const \*a, char const \*b) {  while(1)  {  if(\*a == ' ' && \*b == ' ')  return 0;  else if (\*a == ' ' || ((\*a^42) < (\*b^42)))  return -1;  else if (\*b == ' ' || ((\*a^42) > (\*b^42)))  return 1;  a++; b++;  }  }  //Function to check for reading error  void readErr(){  if(ferror(stdin)){  fprintf(stderr, "Read Error");  exit(1);  }  }  int compare(const void\* ptr1, const void\* ptr2){  const char\* a = \*(const char\*\*)ptr1;  const char\* b = \*(const char\*\*)ptr2;  return frobcmp(a,b); //compare with frobcmp  }  while (cur != EOF && !ferror(stdin)) //Read file until EOF  {  word[cindex] = cur; //Add letters to the word  char\* mem = realloc(word, (cindex+2)\*sizeof(char));  if (allocateErr(mem)){  free(word);  exit(1);  }  word = mem; //point word to reallocated space  if (cur == ' '){ //End of word  arr[windex] = word; //Add word to array  char\*\* mem2 = realloc(arr, (windex+2)\*sizeof(char\*));  if (allocateErr(mem2)){  free(arr);  exit(1);  }  arr = mem2;  windex++;  word = NULL;  word = (char\*)malloc(sizeof(char));  cindex = -1;  }  //ENDCASES  if (next == EOF && cur == ' ')  break;  else if (cur == ' ' && next == ' '){ //extra spaces  while (cur == ' '){ //go past extra spaceS  cur = getchar();  readErr();  }  next = getchar();  readErr();  cindex++;  continue;  }  else if (next == EOF){  cur = ' ';  cindex++;  continue;  }  cur = next;  next = getchar();  readErr();  cindex++;  } | int allocateErr(void\* ptr){  if (ptr == NULL){  fprintf(stderr, "Error Allocating Memory");  return 1;  }  return 0;  }  int main(void){  char\* word; //Holds chars to form words  char\*\* arr; //Holds input words  word = (char\*)malloc(sizeof(char)); //allocate some memory to hold words  arr = (char\*\*)malloc(sizeof(char\*));  char cur = getchar();  readErr(); //check for read error each time reading a char  char next = getchar();  readErr();  int cindex = 0; //character index  int windex = 0; //word index (array)  //SORT  qsort(arr, windex, sizeof(char\*), compare);    //PRINT AND FREE  int i, j; //C doesn't allow declaration in for statement  for (i = 0; i < windex; i++){  j = 0;  while(1){  if (putchar(arr[i][j]) == EOF){  fprintf(stderr, "Error Writing");  exit(1);  }  if (arr[i][j] == ' '){ //reached end  break;  }  j++;  }  }  int k;  for (k = 0; k < windex; k++){ //free all words  free(arr[k]);  }  free(arr); //free word array  exit(0);  } |

LAB 5 SFROB.U

|  |  |
| --- | --- |
| #include<stdio.h>  #include<stdlib.h>  #include <sys/stat.h>  #include <limits.h>  int frobcmp(char const \*a, char const \*b) {  while(1)  {  if(\*a == ' ' && \*b == ' ')  return 0;  else if (\*a == ' ' || ((\*a^42) < (\*b^42)))  return -1;  else if (\*b == ' ' || ((\*a^42) > (\*b^42)))  return 1;  a++; b++;  }  }  int ufrobcmp(char const\* in1, char const\* in2) {  char\* a = in1;  char\* b = in2;  for (;; a++, b++) {  char aa = (\*a) ^ 42;  char bb = (\*b) ^ 42;  if (aa <= UCHAR\_MAX && aa >= 0) {  aa = toupper(aa);  }  if (bb <= UCHAR\_MAX && bb >= 0) {  bb = toupper(bb);  }    if (\*a == ' ' && \*b == ' ')  return 0;  else if (\*a == ' ' || aa < bb)  return -1;  else if (\*b == ' ' || aa > bb)  return 1;  }  }  //Function to check for reading error  void readErr(ssize\_t state){  if (state < 0){  fprintf(stderr, "Error reading file!");  exit(1);  }  }  int compare(const void\* ptr1, const void\* ptr2){  const char\* a = \*(const char\*\*)ptr1;  const char\* b = \*(const char\*\*)ptr2;  return frobcmp(a, b); //compare with frobcmp  }  int ucompare(const void\* ptr1, const void\* ptr2)  {  const char\* a = \*(const char\*\*)ptr1;  const char\* b = \*(const char\*\*)ptr2;  return ufrobcmp(a, b);  }  char\* word;  word = (char\*)malloc(sizeof(char));  char cur[1];  ssize\_t curState = read(0, cur, 1);  readErr(curState);  char next[1];  ssize\_t nextState = read(0, next, 1);  readErr(nextState);  int cindex = 0;  while (curState > 0) {  word[cindex] = cur[0]; //Add letters to the word  char\* mem = realloc(word, (cindex+2)\*sizeof(char));  if (allocateErr(mem)){  free(word);  exit(1);  }  word = mem; //point word to reallocated space  if (cur[0] == ' '){ //End of word  arr[windex] = word; //Add word to array  char\*\* mem2 = realloc(arr, (windex+2)\*sizeof(char\*));  if (allocateErr(mem2)){  free(arr);  exit(1);  }  arr = mem2;  windex++;  word = NULL;  word = (char\*)malloc(sizeof(char));  cindex = -1;  }  //ENDCASES  if (nextState == 0 && cur[0] == ' ')  break;  else if (cur[0] == ' ' && next[0] == ' '){  while (cur[0] == ' '){ //go past extra spaces  nextState = read(0, next, 1);  readErr(nextState);  }  nextState = read(0,next,1);  readErr(nextState);  cindex++;  continue;  }  else if (nextState == 0){  cur[0] = ' ';  cindex++;  continue;  }  cur[0] = next[0];  nextState = read(0, next, 1);  readErr(nextState);  cindex++;  } | int allocateErr(void\* ptr){  if (ptr == NULL){  fprintf(stderr, "Error Allocating Memory");  return 1;  }  return 0;  }  int main(int argc, char\*\* argv){  struct stat fileStat;  if (fstat(0, &fileStat) < 0){ //error  fprintf(stderr, "fstat Error!");  exit(1);  }  char\*\* arr; //Holds input words  char\* fileArr;  int windex = 0; //word index (array)    if (S\_ISREG(fileStat.st\_mode)){  fileArr = (char\*)malloc(fileStat.st\_size\*sizeof(char)); //allocate mem for file input  int pct = 0;  ssize\_t state = read(0, fileArr, fileStat.st\_size);  int count = 0;  if (state > 0){  int m;  for (m = 0; m < fileStat.st\_size; m += count){  count = 1;  if (m == fileStat.st\_size - 1)  fileArr[m] = ' ';  if (fileArr[m] == ' '){  int n;  for (n = m; fileArr[n] != ' '; n++)  count++;  pct++;  }  }  }  arr = (char\*\*)malloc(pct \* sizeof(char\*));  int ind = 0;  int p;  for (p = 0; p < fileStat.st\_size; p++){  if (ind == 0 && fileArr[p] != ' '){  arr[windex] = &fileArr[p];  windex++;  ind = 1;  }  else if (ind == 1 && fileArr[p] == ' ')  ind = 0;  }  }  else  arr = (char\*\*)malloc(sizeof(char));  //SORT  if (argc == 2)  qsort(arr, windex, sizeof(char\*), ucompare);  else  qsort(arr, windex, sizeof(char\*), compare);    //PRINT AND FREE  int i, j; //C doesn't allow declaration in for statement  for (i = 0; i < windex; i++){  long wSize = 0;  j = 0;  while(1){  wSize++;  if (arr[i][j] == ' ') //reached end  break;  j++;  }  if (write(1, arr[i], wSize) == 0){  fprintf(stderr, "Error writing!");  exit(1);  }  }  free(arr); //free word array  exit(0);  } |

LAB 5 TR2B.C (GET/PUTCHAR) AND TR2U.C (READ/WRITE)

|  |  |
| --- | --- |
| #include<stdio.h>  #include<string.h>  #include<stdlib.h>  #include<errno.h>  int duplicates(const char\* from){  int len = strlen(from);  int k, j;  for (k = 0; k < len; k++)  for (j = k + 1; j < len; j++){  if (from[k] == from[j]){  fprintf(stderr, "can't map a character to multiple!");  exit(1);  }  }    return 0;  }  int main(int argc, char\* argv[]){  const char\* from, \* to;  if(argc != 3){  fprintf(stderr, "Need 3 arguments.");  exit(1);  }    from = argv[1];  to = argv[2];  if (strlen(from) != strlen(to)){ //from and to same length  fprintf(stderr, "byte strings are different lengths!");  exit(1);  }    duplicates(from);  char cur = getchar();  int found = 0; //acts as a bool indicating whether match found  while(cur != EOF){  int i;  for(i = 0; i < strlen(from); i++){  if(cur == from[i]){  putchar(to[i]);  found = 1;  break;  }  else{  found = 0;  }  }  if(!found)  putchar(cur);  cur = getchar();  }  return 0;  } | #include<stdio.h>  #include<string.h>  #include<stdlib.h>  #include<errno.h>  int duplicates(const char\* from){  int len = strlen(from);  int k, j;  for (k = 0; k < len; k++)  for (j = k + 1; j < len; j++){  if (from[k] == from[j]){  fprintf(stderr, "can't map a character to multiple!");  exit(1);  }  }    return 0;  }  int main(int argc, char\* argv[]){  const char\* from, \* to;  if(argc != 3){  fprintf(stderr, "Need 3 arguments.");  exit(1);  }  from = argv[1];  to = argv[2];  if (strlen(from) != strlen(to)){ //from and to same length  fprintf(stderr, "byte strings are different lengths!");  exit(1);  }    duplicates(from);  char cur[1];  int found;  ssize\_t state = read(0,cur, 1);  while(state > 0){  int i;  for(i = 0; i < strlen(from); i++)  {  if(cur[0] == from[i]){  cur[0] = to[i];  write(1,cur,1);  found = 1;  break;  }  else  found = 0;  }  if(!found){  write(1,cur,1);  }  state = read(0,cur,1);  }  return 0;  } |

LAB 6 MAKEFILE

CC = gcc

CFLAGS = -std=gnu11 -g -O2 -Wall -Wextra -Wno-unused-parameter

LDLIBS = -lm -lpthread

all: srt

srt\_objects = main.o raymath.o shaders.o

srt: $(srt\_objects)

$(CC) $(CFLAGS) -o $@ $(srt\_objects) $(LDLIBS)

main.o raymath.o: raymath.h

main.o shaders.o: shaders.h

tests = 1-test.ppm 2-test.ppm 4-test.ppm 8-test.ppm

check: baseline.ppm $(tests)

for file in $(tests); do \

diff -u baseline.ppm $$file || exit; \

done

$(tests): srt

time ./srt $@ >$@.tmp

mv $@.tmp $@

dist: srt.tgz

sources = COPYING Makefile main.c raymath.c raymath.h shaders.c shaders.h \

baseline.ppm

srt.tgz: $(sources)

tar cf - --transform='s|^|srt/|' $(sources) | gzip -9 >$@

clean:

rm -f \*.o \*.tmp $(tests) srt srt.tgz

LAB 7 QUESTIONS

1. Yes, the network is still secure because the private key was not sent

across the network. Therefore, other teams do not know the private key

regardless of what bytes were sent over the network, and they do not have

access to the complete public key system for both encryption and decryption.

(1) Yes, the network is still secure for the same reason as above. The other

teams still do not have the private key since it was not revealed as bytes

over the network or in keystrokes.

(2) No, the network is no longer certainly secure. Since other teams had

access to the bytes sent over the network, they had access to the public key.

Now that other teams have accessed USB, they could have found and copied the

private key on my system. In that case, the network is no longer secure.

2. From gpg's man page, --verify simply verifies a signed file or detached

signature that is specified as an argument. This means that anyone could

create another key and tarball and send those over to be verified as the

originals, when in reality, all that has been verified was that the detached

signature matches the key that was just falsely generated.

A solution would be for the original key and signature to be saved somewhere,

perhaps by a third party service or backed up somewhere on the web, so that

the originals can be verified with the receiver, and the receiver can be

certain that it has the correct key.

