```
//my_shell.cpp /*
 * Course: CS 100 Fall 2013
 *
 * First Name: <Kevin>
 * Last Name: <Hsieh>
 * Username: <khsie003>
 * email address: <khsie003@ucr.edu>
 *
 *
 * Assignment: <Homework #5>
 *
 * I hereby certify that the contents of this file represent
 * my own original individual work. Nowhere herein is there
 * code from any outside resources such as another individual,
 * a website, or publishings unless specifically designated as
 * permissible by the instructor or TA.
 */

#include <stdio.h> #include <unistd.h> #include <errno.h> #include <stdlib.h> #include <signal.h> #include <string.h> #include <sys/types.h> #include <sys/wait.h> #include <sys/stat.h> #include <fcntl.h> #include <sstream> #include <iostream>

using namespace std;

bool iRedir; bool oRedir; bool ifPipe; bool background; void parse_arg(string & buf, char*args[]){

   stringstream ss(buf);
   char par_arg[BUFSIZ];
   while(ss >> par_arg){
      char * cur_par = new char[BUFSIZ];

      if(par_arg[0] == '<')
         iRedir = true;
      if(par_arg[0] == '>')
         oRedir = true;
      if(par_arg[0] == '|')
         ifPipe = true;
      if(par_arg[0] == '&'){
         background = true;
      }

      unsigned i = 0;
      for(i = 0; par_arg[i] != '  '; ++i){
         cur_par[i] = par_arg[i];
      }
      cur_par[i] = '  ';
      *args++ = cur_par;
   }
   *args = 0; }

void modArgs(char * args[]){
   //mod args to get only the first arg
   for(unsigned i = 1; args[i]; ++i){
         delete args[i];
   }
   args[1] = 0; }

void modTee(char * args[], char * newargs[], char * inFile){

   for(unsigned i = 0; args[i]; ++i){
      if(args[i][0] == '<'){
         unsigned x;
         for(x = 0; args[i+1][x] != '  '; ++x){
            inFile[x] = args[i+1][x];
         }
         inFile[x] = '  ';
```

```
        }
    }

    //take out every < and > and "<+1"
    unsigned k, j;
    for(k = 0, j = 0; args[k]; ++k){
        if(args[k][0] == '<' || args[k][0] == '>'){
            if(args[k][0] == '<')
                k++;
            continue;
        }

        char * temp = new char [BUFSIZ];

        unsigned x;
        for(x = 0; args[k][x] != '  ';++x){
            temp[x] = args[k][x];
        }
        temp[x] = '  ';
        newargs[j] = temp;
        ++j;
    }
    newargs[j] = 0;

    //delete args
    for(unsigned i = 0; args[i]; ++i){
        delete args[i];
    }

    //set args  = newargs
    unsigned l;
    for(l = 0; newargs[l]; ++l){
        args[l] = newargs[l];
    }
    args[l] = 0; }

void getFileNames(char * args[], char * inFile, char * outFile){

    if(iRedir && strcmp(args[0], "tee")){ //'<' infile: i + 1, outfile i - 1
        for(unsigned i = 0; args[i]; ++i){
            if(args[i][0] == '<'){ // get input filename
                //get infile name
                unsigned p;
                for(p = 0; args[i+1][p] != '  '; ++p){
                    inFile[p] = args[i + 1][p];
                }
                inFile[p] = '  ';

                //get outfile name
                unsigned q;
                for(q = 0; args[i -1][q] != '  '; ++q){
                    outFile[q] = args[i-1][q];
                }
                outFile[q] = '  ';
            }
        }
    }

    if(oRedir && strcmp(args[0], "tee")){ //'>' infile: i -1, outfile: i + 1
        for(unsigned i = 0; args[i]; ++i){
            if(args[i][0] == '>'){
                //get infile name
                unsigned p;
                for(p = 0; args[i-1][p] != '  '; ++p){
                    inFile[p] = args[i - 1][p];
                    cout << "in[p]: " << inFile[p] << endl;
```

```
            }
            inFile[p] = ’  ’;
            cout << "in[p]: " << inFile[p] << endl;

            //get outfile name
            unsigned q;
            for(q = 0; args[i+1][q] != ’  ’; ++q){
               outFile[q] = args[i+1][q];
               cout << "out[q]: " << outFile[q] << endl;
            }
            outFile[q] = ’  ’;
            cout << "out[q]: " << outFile[q] << endl;

         }
      }
   }

   if((iRedir || oRedir) && strcmp(args[0], "tee"))
      modArgs(args);

   if((iRedir||oRedir) && !strcmp(args[0], "tee")){
      char *newargs[64];
      modTee(args, newargs, inFile);
   }

}

void setFileDescriptors(char * args[], const char *inFile, const char *outFile){
   int inFD, outFD;
   if(iRedir && strcmp(args[0], "tee")){// ’<’ redirect
      inFD = open(inFile, O_RDWR);
      if(inFD < 0)
         cerr << "Error: opening input file0;
      dup2(inFD,0);
      close(inFD);

      if(strcmp(outFile,args[0])){
         outFD = open(outFile, O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
         if(outFD < 0)
            cerr << "Error: opening output file0;
         dup2(outFD,1);
         close(outFD);
      }

   }

   if(oRedir && strcmp(args[0], "tee")){// ’>’ redirect
      if(strcmp(inFile,args[0])){
         inFD = open(inFile, O_RDWR);
         if(inFD < 0)
            cerr << "Error: opening input file0;
         dup2(inFD, 0);
         close(inFD);
      }

      outFD = open(outFile, O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
      if(outFD < 0)
         cerr << "Error: opening output file0;
      dup2(outFD,1);
      close(outFD);

   }

   if( iRedir && !strcmp(args[0], "tee")){
      //dup2 the inFile
      inFD = open(inFile, O_RDWR);
```

```
        if(inFD < 0)
            cerr << "Error: opening input file0;
        dup2(inFD, 0);
        close(inFD);
    }

}

void par_Pipe(char *args[], char *args2[]){
    unsigned i;
    for(i = 0; args[i]; ++i){
        if(args[i][0] == '|')
            break;
    }
    cout << "i: " << i << endl;
    cout << "args[i]: " << args[i] << endl;
    i++;
    unsigned j;
    for( j = i; args[j]; ++j){
        args2[j-i] = args[j];
        //~ cout << "__args2[" << j-i -1 << "]: " << args2[j-i-1] << endl;
    }
    args2[j] = 0;
    //~ cout << "args[i-1]: " << args[i -1] << endl;
    delete args[i-1];
    args[i - 1] = 0;
    //~ cout << "args[i-1]: " << args[i -1] << endl; }

int exec_arg(char * args[]){
    int fd[2];
    char * inFile = new char [BUFSIZ];
    char * outFile = new char [BUFSIZ];
    //if pipe line exists, separate into two commands
    char * args2[64];
    if(ifPipe){
        par_Pipe(args,args2);
        pipe(fd);
        //~ for(unsigned i = 0; args[i]; ++i)
            //~ cout << "args[" << i << "]: " << args[i] << endl;
        //~ for(unsigned i = 0; args2[i]; ++i)
            //~ cout << "args2["<< i <<"]: " << args2[i] << endl;
        cout << "here?" << endl;
    }
    ///fork and execute
    int status;
    int pid = fork();
    switch(pid){
        case -1:
            cerr << errno << " fork failed!0;
            return -1;
        case 0: // child process
            ///check for if I/O redirect is needed
            if(!ifPipe){
                getFileNames(args,inFile,outFile);
                setFileDescriptors(args,inFile,outFile);
            }
            else{
                //piping for child
                getFileNames(args,inFile,outFile);
                dup2(fd[1],1);
                close(fd[0]);

            }
            //execute the command
            if(execvp(args[0],args)== -1)
                cerr << args[0] << " failed!0;
```

```
            _exit(1);
        default:// parent process
            iRedir = false;
            oRedir = false;

            if(ifPipe){
                //~ piping for parent
                getFileNames(args2, inFile, outFile);
                dup2(fd[0],0);
                close(fd[1]);
            }
            else{
                //delete inFile and outFile
                delete [] inFile;
                delete [] outFile;
                ifPipe = false;
                return waitpid(-1, &status, 0);
            }
    }

    return 0; }

int main(){

    string buf;
    char *args[64];
    while(1){
        cout.flush();
        cout << "~> " << flush;
        getline(cin,buf);
        parse_arg(buf, args);

        if(exec_arg(args) < 0)
            cerr << "Error: " << args[0] << endl;

        //destructor to prevent memory leak
        for(unsigned i = 0; args[i]; ++i){
            delete args[i];
        }
    }
    return 0; }
```