|  |  |
| --- | --- |
| #include <fstream>  maze: main.o maze.o  g++ -Wall -g main.o maze.o -o maze  %.o: %.cpp g++ -Wall -g -c $<  main.o: maze.h maze.o: maze.h  clean: rm -f \*.o maze  **If/else:** x<3?(true ):(false)  **int to char:** char x = (char)(a+’0’);  **str[n]** = ‘A’; str[n+1] = ‘\0’;  **const char\*c=strchr**(s1,'e');  **Pl**ease split, this, sentence | **<cctype>**isalnum/isalpha/isdigit/isspace/ispunct/islower/toupper/isspace  **<cstring>**strlen(e\0)/strcmp(s1,””/0 if same)/strcpy/strcat/strncmp/strncat  **IO:** ifstream in("maze.txt");decode(input, cout);input.close() | ofstream out(“output.txt”); out.close();in.ignore(256,’ ’); strings skipping ws: char word[]; while(in>>word)  in.get(ch); out.put(ch); in.get(charArray, size); in.getline(charArray, size) (c)out.put(letter); (c)in.putback(ch);  **ASCII:**32:‘ ’|48:0|49:1|57:9|65:A|90:Z|97:a|122:z  **switch**(path[i]){case'N':row--;break;default:return false;}  **Init:** char str[512]=“ab”; char\*str=new char[64];return ptr;  **cstring to int** (#include <cstdlib>) int num = atoi(const char\* str)  **int \*p** = new int; \*p = &num; \*p = number; delete p; |**int \*p[size]** = {“ab” , “cd”}; int \*p[size]; for (…) p[n]=sth;  **char\*cr=strrchr**(s1,'e'); Please split, this, sentenc**e ; const char\*sr=strstr**(s1,"this"); **Please split,** this, sentence  int c = cin.peek(); // peek character if ( c == EOF ) return 1; |

**Mines**

**void load\_board**(\*filename,board[9][9]) {ifstream input(filename);char line[512];for (int r = 0; r<9; r++) {input.getline(line, 512);strcpy(board[r], line);}input.close();}

**void display\_board**(char board[9][9]) {cout << setw(4) << " " << " ";for (int c=0; c<9; c++)cout << (c +1 % 9);cout << endl;for (int r=0; r<9; r++) {cout << setw(4) << (char) (r + 'A') << " |";for (int c=0; c<9; c++) cout << board[r][c];cout << endl;}}

**void initialise\_board**(char board[9][9]) {for(int i = 0; i < 9; ++i)for(int j = 0; j < 9; ++j)board[i][j] = '?';}

**bool is\_complete**(mines[9][9], revealed[9][9]){for(int r = 0; r < 9; ++r) for(int c = 0; c < 9; ++c) if(revealed[r][c] == '?') {if(mines[r][c] != '\*') returnfalse;}return true;}

**bool validRC**(int r, int c) {return r>=0 && r<9 && c>=0 && c<9;}

**bool getCoordinates**(const char\* position, int& r, int& c){if(strlen(position) < 2) return false;int tempr = position[0] - 'A';int tempc = position[1] - '1'; if(!validRC(tempr,tempc)) return false;r = tempr;c = tempc;return true;}

**int count\_mines**(const char position[3], const char mines[9][9]){int r, c, adjR, adjC, numMines = 0;if(!getCoordinates(position, r, c))return 0;for(int rowDisp = -1; rowDisp <= 1; ++rowDisp){for(int colDisp = -1; colDisp <= 1; ++colDisp) {if(colDisp == 0 && rowDisp == 0)continue;adjR = r + rowDisp;adjC = c + colDisp; if(validRC(adjR, adjC))if(mines[adjR][adjC] == '\*')++numMines;}}return numMines;}

**MoveResult make\_move**(const char\* position, const char mines[9][9], char revealed[9][9]){int r, c, adjR, adjC, n;bool flag = false;char adj[3];if(!getCoordinates(position, r, c)) return INVALID\_MOVE;if(strlen(position) == 3) {if(position[2] != '\*')return INVALID\_MOVE;flag = true;}if(flag) {if(revealed[r][c] == '\*')return REDUNDANT\_MOVE;revealed[r][c] = '\*';return VALID\_MOVE;}if(revealed[r][c] != '?' && revealed[r][c] != '\*')return REDUNDANT\_MOVE;if(mines[r][c] == '\*')return BLOWN\_UP;if(!flag) {n = count\_mines(position, mines);revealed[r][c] = (n==0) ? ' ' : ('0' + n);if(n == 0) {for(int rowDisp = -1; rowDisp <= 1; ++rowDisp){ for(int colDisp = -1; colDisp <= 1; ++colDisp) {if(colDisp == 0 && rowDisp == 0)continue;adjR = r + rowDisp;adjC = c + colDisp;if(validRC(adjR, adjC)) {adj[0] = 'A' + adjR;adj[1] = '1' + adjC;adj[2] = '\0';make\_move(adj, mines, revealed);}}}}}if(is\_complete(mines, revealed))return SOLVED\_BOARD;return VALID\_MOVE;}

**bool find\_safe\_move**(const char revealed[9][9], char\* move){char current[3] = "A1", next[3];int r, c, adjR, adjC;while(true) {getCoordinates(current, r, c);if(revealed[r][c] == '?') {for(int rowDisp = -1; rowDisp <= 1; ++rowDisp){for(int colDisp = -1; colDisp <= 1; ++colDisp) {if(colDisp == 0 && rowDisp == 0)continue;

adjR = r + rowDisp;adjC = c + colDisp;if(validRC(adjR, adjC)) {Opinion op = squaresOpinion(adjR, adjC, r, c, revealed);if(op != UNKNOWN) {strcpy(move, current);

if(op == MINE) {strcat(move, "\*");}return true;}}}}}if(!getNextSquare(current, next))break;strcpy(current, next);}return false;}

**Opinion squaresOpinion**(int r, int c, int onR, int onC, const char revealed[9][9]){char n = revealed[r][c];int num = n - '0';int unfound = num;int pos = num;int adjR, adjC;if(num < 0 && num > 8)return UNKNOWN;for(int rowDisp = -1; rowDisp <= 1; ++rowDisp){for(int colDisp = -1; colDisp <= 1; ++colDisp) {adjR = r + rowDisp; adjC = c + colDisp;if((colDisp == 0 && rowDisp == 0) || (adjR == onR && adjC == onC))continue;if(validRC(adjR, adjC)) {if(revealed[adjR][adjC] == '\*') {unfound--;

pos--;} if(revealed[adjR][adjC] == '?') {pos--;}}}}if(unfound == 0)return NO\_MINE;if(unfound > 0 && pos == 1)return MINE;return UNKNOWN;}

**bool getNextSquare**(char prev[3], char next[3]){int r, c;if(!getCoordinates(prev, r, c)) {return false;}if(c < 8) {++c;} else {++r;c = 0;}if(!validRC(r,c)) {return false;}

next[0] = 'A' + r;next[1] = '1' + c;next[2] = '\0';return true;}

**enum** MoveResult {INVALID\_MOVE = -3, REDUNDANT\_MOVE, BLOWN\_UP, SOLVED\_BOARD, VALID\_MOVE}; **enum** Opinion{NO\_MINE, MINE, UNKNOWN};

**Playfair (print grids/swap locations/pass int/char/stream by ref)**

**bool occurs\_before**(const char str[], char letter,int pos){ for(int i=0;i<pos;i++) if(str[i]==letter) return true; return false;}

**void grid**(const char codeword[], char square[6][6]){ int i,len=strlen(codeword); char buf[512]; buf[0]='\0'; int count=0; for(int j=0;j<len;j++) if(!occurs\_before(codeword,codeword[j],j)){ buf[count]=codeword[j]; count++; } for(i=0;i<strlen(buf);i++){ square[i/6][i%6]=buf[i]; } for(char c='A';c<='Z';c++){ if(!occurs\_before(codeword,c,len)){ square[i/6][i%6]=c; i++; } } for(char n='0';n<='9';n++){ if(!occurs\_before(codeword,n,len)){ square[i/6][i%6]=n; i++; } } }

**void bigram**(char square[6][6], char inchar1, char inchar2, char& outchar1, char& outchar2){int r1,r2,c1,c2;for(int r=0;r<6;r++)for(int c=0;c<6;c++)if(square[r][c]== inchar1){r1=r;c1=c;break;}for(int r=0;r<6;r++)for(int c=0;c<6;c++)if(square[r][c]==inchar2){r2=r;c2=c;break;}outchar1=square[r1][c2];outchar2=square[r2][c1];}

**Multitap(STREAMS/casted looping)**

**char decode\_character**(const char multitap[]) { char output[512]; for(int i='a'; i<='z'; i++){ encode\_character((char)i,output); if(!strcmp(output,multitap)) return (char) i; } if(!strcmp(multitap,"0")) return ' '; if(!strcmp(multitap,"1")) return '.'; if(!strcmp(multitap,"11")) return ','; if(!strcmp(multitap,"111")) return '!'; return '?'; return '@'; }.

**void print\_decode**(const char\* input, ostream& out, bool caps) { char ch = decode\_character(input); if (isalpha(ch) && caps) ch = toupper(ch); out << ch; }

**void decode**(istream& input, ostream& output){ char multitap[512]; char multitap\_ch[8]; char outputchar; while(input.getline(multitap, 512)){ int i = 0, count = 0; bool upperflag = false; while(multitap[i]){ if(i > 0 && multitap[i] != multitap[i - 1] && multitap[i - 1] != '|' && multitap[i - 1] != '#' && multitap[i - 1] != '\*'){ multitap\_ch[count] = '\0'; if(upperflag){ outputchar = toupper(decode\_character(multitap\_ch)); upperflag = false; } else outputchar = decode\_character(multitap\_ch); output << outputchar; count = 0; } if(multitap[i] == '|'){ i++; continue; } if(multitap[i] == '#'){ i++; upperflag = true; continue; } multitap\_ch[count] = multitap[i]; count++; i++; } output << endl; } }

**Sonnet(dictionary)**

**bool get\_word**(const char \*input\_line, int word\_number, char \*output\_word) { char \*output\_start = output\_word; int words = 0; if (word\_number < 1) { \*output\_word = '\0'; return false; } do { while (\*input\_line && !isalnum(\*input\_line)) input\_line++; if (\*input\_line == '\0') break; output\_word = output\_start; while (\*input\_line && (isalnum(\*input\_line) || \*input\_line=='\'')) { \*output\_word = toupper(\*input\_line); output\_word++; input\_line++; } \*output\_word = '\0'; if (++words == word\_number) return true; } while (\*input\_line); \*output\_start = '\0'; return false; }

**River(add/create scene)**

**char** \*\***create\_scene**() { char \*\*scene = allocate\_2D\_array(HEIGHT,WIDTH); for (int i=0; i<HEIGHT; i++) for (int j=0; j<WIDTH; j++) scene[i][j] = ' '; return scene; }

**void** **filter(**char \*line) { while (\*line) { if (\*line >= ' ') line++; else \*line = '\0'; } }

**bool add\_to\_scene**(char \*\*scene, int row, int col, const char \*filename) { ifstream in(filename); if (!in) return false; int start\_Col = col; char line[512]; in.getline(line,512); filter(line); while (!in.fail()) { for (int n=0; n<strlen(line); n++) { if (row >= SCENE\_HEIGHT) return false; if (col >= SCENE\_WIDTH) break; scene[row][col++] = line[n]; } row++; col = start\_Col; in.getline(line,512); filter(line); } return true; }

**char\*\* make\_river\_scene**(const char\* left, const char\* boat){ char \*\*scene = create\_scene(); add\_to\_scene(scene, 0, 0, "bank.txt"); int i=0,j=0,l\_M=0,l\_C=0,boatM=0,boatC=0,b=1; while(left[i]){ if(left[i]=='M') l\_M++; if(left[i]=='C') l\_C++; if(left[i]=='B') b--; i++; } while(boat[j]){ if(boat[j]=='M') boatM++; if(boat[j]=='C') boatC++; j++; } add\_to\_scene(scene, 17, 19+17\*b, "boat.txt"); for(int k=0; k<l\_M; k++) add\_to\_scene(scene, 2, 1+6\*k, "missionary.txt"); for(int k=0; k<l\_C; k++) add\_to\_scene(scene, 11, 1+6\*k, "cannibal.txt"); for(int k=0; k<3-l\_M-boatM; k++) add\_to\_scene(scene, 2, 54+6\*k, "missionary.txt"); for(int k=0; k<3-l\_C-boatC; k++) if(b==0){ for(int k=0; k<boatM; k++) add\_to\_scene(scene, 11, 22+6\*k, "missionary.txt"); for(int k=0; k<boatC; k++) if(b==1){ for(int k=0; k<boatM; k++) add\_to\_scene(scene, 11, 39+6\*k, "missionary.txt"); for(int k=0; k<boatC; k++) add\_to\_scene(scene, 11, 39+6\*k, "cannibal.txt"); } return scene; }

**int play\_game**(){ char left[8] = "BCCCMMM"; char targets[3]; char\*\* scene = make\_river\_scene(left,""); print\_scene(scene); int result; while(true){ cin >> targets; result = perform\_crossing(left, targets); if(result != VALID\_NONGOAL\_STATE) return result; } }

**Maze(valid solution)**

bool **get\_maze\_dimensions**(const char \*filename, int &height, int &width) { char line[512]; ifstream input(filename); height = width = 0; input.getline(line,512); while (input) { if ( (int) strlen(line) > width) width = strlen(line); height++; input.getline(line,512); } if (height > 0) return true; return false; }

bool **valid\_solution**(const char \*path, char \*\*m, int height, int width){ int row,column; if(find\_marker('>', m, height, width, row, column)){ for (int i=0; i<strlen(path); i++){ switch(path[i]){ case 'N': row--; break; case 'S': row++; break; case 'E': column++; break; case 'W': column--; break; default: return false; } if (row>height || row<0 || column>width || column<0) return false; if (m[row][column]=='-' || m[row][column]=='+' || m[row][column]=='|') return false; } if (m[row][column]=='X') return true; return false; } return false; }

**Shrink** (declaring string array)

int **lookup**(const char\* word, const char\* dictionary[]){int i = 0;char word\_buf[100];do {strcpy(word\_buf, dictionary[i]);if (!strcmp(word,word\_buf)) return i;i++;} while (strcmp(word\_buf,""));return -1;}

void **encode**(const char\* word, char compressed[100], const char\* dictionary[]){int pos;if (strlen(word) == 2){if (isalnum(word[0]) && isalnum(word[1])){compressed[0] = '!';compressed[1] = word[0];compressed[2] = word[1];compressed[3] = '\0';return;}}pos = lookup(word,dictionary);if (pos > -1){if (pos < 10){compressed[0] = '0'; compressed[1] = (char)(((int)'0')+pos);}else if (pos < 100){compressed[0] = (char)(((int)'0')+pos/10);compressed[1] = (char)(((int)'0')+pos%10);} else{cerr << "pos out of 00 - 99 range with value: " << pos << endl;}compressed[2] = '\0';return;}strcpy(compressed,word);return;}

**print\_braille**(text, ostream& cout){char braille[512];encode(text,braille);for (int row=0; row<3; row++){size\_t i=row;while(i<strlen(braille)){cout << braille[i]; i+=3;cout << braille[i] << " ";i+=3;}cout << endl;}for(size\_t i=0; i<strlen(text); i++){if(isupper(text[i]) || isdigit(text[i]))cout << " ";cout << text[i] << " ";}cout << endl;}

**Correct**

void **ascii\_to\_binary**(char ch,char\*binary){for(int n=128;n;n>>=1)\*binary++ =(ch&n)?'1':'0';\*binary='\0';}

char **binary\_to\_ascii**(char\*binary){int ch=0;for(int n=0,slide=128;n<8;n++,slide>>=1){if(binary[n]=='1')ch=ch|slide;}return ch;}

void **text\_to\_binary**(const char\* str, char\* encoded){char binary[512];for(size\_t i=0;i<strlen(str);i++){ascii\_to\_binary(str[i],binary);strcat(encoded,binary);}}

void **recursive**(text, binary){char bin\_buf[512];ascii\_to\_binary(text[0],bin\_buf);strcat(binary,bin\_buf);if(strlen(text)>1)text\_to\_binary\_recursive(&text[1],binary);}

void **text\_to\_binary**(const char\* text, char\* binary){binary[0]='\0';text\_to\_binary\_recursive(text,binary);}

**bin2text**(encoded,text){char buf[512];for(size\_t i=0;i<(strlen(encoded)/8);i++){strncpy(buf,encoded+(i\*8),8);char ch=binary\_to\_ascii(buf);text[i]=ch;text[i+1]='\0';}}

**correction**(data, char\* correct){char c1,c2,c3;char data\_buf[512],cor\_buf[512];correct[0]='\0';for(int i=0;i<(strlen(data)/4);i++){strncpy(data\_buf,data+(i\*4),4); ((data\_buf[0]+data\_buf[1]+data\_buf[3])%2==0)?c1='0':c1='1';cor\_buf[0]=c1;cor\_buf[1]=c2;cor\_buf[2]=data\_buf[0];cor\_buf[7]='\0';strcat(correct,cor\_buf);}}

**int decode**(received,decoded){char rec\_buf[512],dec\_buf[512];char p1,p2,p3;int pos,count=0;decoded[0]='\0';for(int i=0;i<(strlen(received)/7);i++){strncpy(rec\_buf, received+(i\*7),7);**rec\_buf[7]='\0';**(rec\_buf[3]+rec\_buf[4]+rec\_buf[5]+rec\_buf[6])%2==0?p1=0:p1=1;if(!(p1==0 && p2==0 && p3==0)){count++;pos=p1\*4+p2\*2+p3 ;rec\_buf[pos-1]=='0'?rec\_buf[pos-1]='1':rec\_buf[pos1]='0';}dec\_buf[0]=rec\_buf[2];rec\_buf[4];dec\_buf[4]='\0';strcat(decoded,dec\_buf);}return count;}

**Palindrome and Anagram**

void **reverse**(const char\* str, char\* rev){ size\_t len=strlen(str); for(size\_t i=0; i<=len; i++) rev[i]=str[len-1-i]; rev[len]='\0'; }

**Soundex(NULL STRTOK)**

**encode**(surname,soundex){code[]={'0'};size\_t n=1;soundex[0]=surname[0];for(int x=1;x<4;x++){while(n<strlen(surname)&&(code[surname[n]-'a']=='0'|| code[surname[n]-'a']==soundex[x-1])) n++;if(n<strlen(surname)) soundex[x]=code[surname[n++]-'a'];else soundex[x]='0';}soundex[4]='\0';}

**compare**(one,two){if(strlen(one)==0&&strlen(two)==0) return true;if(one[0]==two[0]){onecut[strlen(one)];for(size\_t i=1;i<=strlen(one);i++) onecut[i-1]=one[i];

twocut[strlen(two)];for(size\_t i=1;i<=strlen(two);i++) twocut[i-1]=two[i];return compare(onecut,twocut);}return false;}

**count**(surname,sentence){sen\_cpy[512],\*sen\_tok,soundex[5],soundex\_tok[5];count=0;strcpy(sen\_cpy,sentence);encode(surname,soundex);**for(sen\_tok=strtok(sen\_cpy,", .!");sen\_tok!=NULL;sen\_tok=strtok(NULL,", .!"))**{encode(sen\_tok,soundex\_tok);if(compare(soundex,soundex\_tok)) count++;}return count;}

**Piglatin(Streams)**

**it stream**(istream&in,ostream&out){eng[65],pig[65];ch;while(in.get(ch)){if(isalnum(ch)){in.putback(ch);in>>eng;trans (eng,pig);out << pig;}else out<<ch;}}

**rc stream**(istream&in,ostream&out){eng[65],pig[65];ch;if(in.get(ch)){if(isalnum(ch)){in.putback(ch);in>>eng;trans(eng,pig);out << pig;}else out<<ch;steam(in,out);}}

**Substring(Prefix)**

**it** **subPos**(sub,str){size\_t count=0;if(is\_prefix(sub,str))return count;while(count!=strlen(str)){count++;char strcut[strlen(str)];for(size\_t i=0; i<=strlen(str); i++)strcut[i]=str[i+count];if(is\_prefix(sub,strcut))return count;}return -1;}

**rc** **subPos**(sub,str){if(is\_prefix(sub,str))return 0;if(strlen(sub)>strlen(str))return -1;return(substring\_position(sub,&str[1])==-1)?-1:substring\_position(sub,&str[1])+1;}

bool get\_maze\_dimensions(const char \*filename, int &height, int &width) { char line[512]; ifstream input(filename); height = width = 0; input.getline(line,512); while (input) { if ( (int) strlen(line) > width) width = strlen(line); height++; input.getline(line,512); } if (height > 0) return true; return false; }

**Sudoku**

**void load\_board**(const char\*filename,char board[9][9]){ifstream in(filename);if(!in)cout<<"Fail!"<<endl;char buf[512];int row=0;in.getline(buffer,512); while(in&&row<9){for(int n=0;n<9;n++){if(buffer[n]=='.'||isdigit(buffer[n]));board[row][n]=buffer[n];}row++;in.getline(buffer,512);}}

**void display\_board**(const char board[9][9]){for(int r=0;r<9;r++)cout<<(char)('1'+r)<<" ";cout<<endl;for(int r=0;r<9;r++){if(!(r%3))cout << "+"<<endl;else cout<<"+"

<<endl;cout<<(char)('A'+r)<<" ";for(int c=0;c<9;c++){cout<<((c%3)?':':'|')<<" ";cout<<((board[r][c]=='.')?' ':board[r][c])<<" ";}cout<<"|"<<endl;}cout<<" +"<<endl;}

**bool move**(const char pos[2],int digit,char board[9][9]){int r=pos[0]-'A',c=pos[1]-'0'-1;if(!isdigit(digit))return false;if(r<0||r>9||c<0||r>9)return false;for(int i=0;i<9;i++){

if(board[r][i]==digit)return false;}for(int j=0;j<9;j++){if(board[j][c]==digit)return false;}int sr=r-(r%3);int sc=c-(c%3);for(int x=sr,n=sr+3;x<n;x++)for(int y=sc,m=sc+3;y<m;y++)if(board[x][y]==digit)return false;board[r][c]=digit;return true;}

**bool save\_board**(const char\* filename, char board[9][9]){ofstream output(filename);if(!output){return false;}for(int r=0;r<9;r++){for(int c=0;c<9;c++){output<< board[r][c];}output<<endl;}output.close();return true;}bool solve\_board(char board[9][9]){static int counter=0;counter++;if(is\_complete(board))return true;else{char new\_board[9][9];copy(new\_board,board);char digit='0';char pos[]="A1";for(int row=0;row<9;row++)for(int col=0;col<9;col++){pos[0]='A'+row;pos[1]='1'+col; if(isdigit(new\_board[row][col]))continue;for(int n=1;n<=9;n++){digit='0'+n;if(make\_move(pos,digit,new\_board))if(solve\_board(new\_board)) {copy(board,new\_board);if(counter!=0)cout<<counter<<" recursions"<<endl;counter=0;return true;}}return false;}return false;}}

**void copy**(char to[9][9],char from[9][9]){ for(int row=0;row<9;row++) for (int col=0;col<9;col++) to[row][col]=from[row][col]; }

**bool solve\_board**(char board[9][9]){ static int counter=0; counter++; if(is\_complete(board)) return true; else{ char new\_board[9][9]; copy(new\_board,board); char digit='0'; char pos[]="A1"; for(int row=0;row<9;row++) for(int col=0;col<9;col++){ pos[0]='A'+row; pos[1]='1'+col; if(isdigit(new\_board[row][col])) continue; for(int n=1;n<=9;n++){ digit='0'+n; if(make\_move(pos,digit,new\_board)) if(solve\_board(new\_board)){ copy(board,new\_board); if(counter!=0) cout<<counter<<" recursions"<<endl; counter=0; return true; } } return false; } return false; } }

**Tube**(find symbol/compare file lines with input)

**bool get\_symbol\_position**(char\*\* map, int height, int width, char target, int& r, int& c){ int i, j; for(i = 0; i < height; i++){ for(j = 0; j < width; j++){ if(map[i][j] == target){ r = i; c = j; return true; } } } r = -1; c = -1; return false; }

**char get\_symbol\_for\_station\_or\_line**(const char name[]){ ifstream input1("lines.txt"); char lines[32]; input1.getline(lines, 32); while(input1.good()){ if(!strcmp(name, &lines[2])){ input1.close(); return lines[0]; } input1.getline(lines, 32); } input1.close();

**Direction string\_to\_direction**(const char\* token) { const char\* strings[] = {"N", "S", "W", "E", "NE", "NW", "SE", "SW"}; for(int n=0; n<8; n++) { if(!strcmp(token, strings[n])) return (Direction) n; } return INVALID\_DIRECTION; }

**Other**

**selectionSort**(str){char temp;int i,j,min,len=strlen(str);for(i=0; i<len-1; i++){min=i;for(j=i+1; j<len; j++){if(str[j]<str[min])min=j;}temp=str[i];str[i]=str[min];str[min]=temp;}}

**void reverse\_words(char \*s)** { char b[100], \*t, \*z; int c = 0; t = s; while(\*t) { //processing complete string while(\*t != ' ' && \*t != '\0') { //extracting word from string b[c] = \*t; t++; c++; } b[c] = '\0'; c = 0; reverse\_string(b); // reverse the extracted word z = b; while (\*z) { //copying the reversed word into original string \*s = \*z; z++; s++; } while (\*s == ' ') { // skipping space(s) s++; } t = s; // pointing to next word } }

**void reverse\_string**(char \*t) { int l, c; char \*e, s; l = strlen(t); e = t + l - 1; for (c = 0; c < l/2; c++) { s = \*t; \*t = \*e; \*e = s; t++; e--; } }

**int main (int argc, char\*\* argv)** { int cmd\_input[4], empty\_or\_full = 0; // validate command line input for (int i = 1; i < argc; i++) { int int\_input = check\_arg(argv[i]); if (int\_input == -1) { cout << "parameters not integers\n"; return int\_input; } cmd\_input[i-1] = int\_input; }

**const char s1[512]={"Please split, this, sentence into tokens**"}; char s2[strlen(s1)]; strncpy(s2,s1+7,strlen(s1)-7); cout<<s2; char \*p = strtok(s2,", "); while(p!=NULL){ cout<<p<<endl; p=strtok(NULL,", "); } return 0; }

**void** **assign\_list**(Node\_ptr &a\_list) { Node\_ptr current\_node, last\_node; assign\_new\_node(a\_list); cout << "Enter first word (or '.' to end list): "; cin >> a\_list->word; if (!strcmp(".",a\_list->word)) { delete a\_list; a\_list = NULL; } current\_node = a\_list; /\* LINE 13 \*/ while (current\_node != NULL) { assign\_new\_node(last\_node); cout << "Enter next word (or '.' to end list): "; cin >> last\_node->word; if (!strcmp(".",last\_node->word)) { delete last\_node; last\_node = NULL; } current\_node->ptr\_to\_next\_node = last\_node; current\_node = last\_node; } }

**struct Node** { string content; Node\* ptr; }; typedef Node\* NodePtr; | current\_node->ptr= head; head = current\_node;