**2019 UNIX 프로그래밍 과제 1**

201411151 전현태

목차

1. **Mycat**
   1. 아이디어
   2. 구현범위
   3. 동작에 대한 설명
      1. 기본동작
      2. 옵션의 동작
      3. 옵션
   4. 수행 예제
2. **Mycp**
   1. 아이디어
   2. 구현범위
   3. 동작에 대한 설명
      1. 기본동작
      2. 옵션의 동작
      3. 옵션
   4. 수행 예제
3. **Myls**
   1. 아이디어
   2. 구현범위
   3. 동작에 대한 설명
      1. 기본동작
      2. 옵션의 동작
      3. 옵션
   4. 수행 예제
4. **mycat [option] … [FILE]**

* **아이디어**

1. fopen() 함수를 이용하여 대상 파일의 파일포인터를 받아와 fgets()함수를 이용하여 buffer에 한 line씩 받는다. 표준출력을 이용하여 buffer를 출력하여 구현하였다.
2. 옵션의 구현은 getopt()함수와 옵션의 상태를 담아놓는 구조체를 정의 및 이용하여 구현하였다.

(struct opt { int a; int b; … } 의 형태)

* **구현범위**

1. 원래의 cat과는 다르게 인자로 file이 들어오지 않으면 return하고 함수를 종료한다.
2. 구현한 옵션은 총 7개이다.
   1. -A : 옵션 -vET와 같다.
   2. -b : 비지 않은 라인에 번호를 붙힌다.
   3. -E : 라인의 맨 끝에 $사인을 붙힌다.
   4. -e : 옵션 -vE와 같다.
   5. -n : 모든 라인에 번호를 붙힌다.
   6. -T : 모든 탭을 ^I문자로 만든다.
   7. -t : 옵션-vT와 같다.

옵션 -v를 구현하지 못했다. 따라서 –A, -e, -t의 구현은 완전하지는 않다.

* **동작에 대한 설명**

1. 기본동작

|  |
| --- |
| int main(int argc, char\* argv[]) {  FILE\* fp;  char buf[256];  char\* line = (char\*)calloc(256, sizeof(char));  int c;  int i = 1, lineNum;  struct opt opt = {  .b = 0, .E = 0,.n = 0, .v = 0 , .T = 0  };  extern char\* optarg;  extern int optind;    …(중략)…    // 인자가 없을때 함수를 종료한다.  if (argc == optind) {  return 0;  }  // 모든 인자에 대해 수행한다.  for (i = optind; i < argc; i++) {    // argv가 디렉토리라면 다음 인자로 넘어간다.  if (isDir(argv[i])) {  printf("mycat: %s: Is a directory\n", argv[i]);  continue;  }  if ((fp = fopen(argv[i], "r")) == NULL) {  perror(argv[i]);  exit(1);  }  lineNum = 1;  while (fgets(buf, sizeof(buf), fp) != NULL) {  strcpy(line, buf);  … (중략) …  printf("%s", line);  }  }  fclose(fp);  return 0;  } |

기본 동작의 구현은 while문과 fgets(), printf() 를 이용하여 구현하였다.

1. 옵션의 동작

|  |
| --- |
| struct opt {  int b;  int n;  int E;  int v;  int T;  };  int main(int argc, char\* argv[]) {  …(중략)…  // v는 구현 못함  while ((c = getopt(argc, argv, "AbEnetTv")) != -1) {  switch (c) {  case 'A': // equivalent to -vET  opt.v = 1;  opt.E = 1;  opt.T = 1;  break;  case 'b': // number nonempty output lines  opt.b = 1;  if (opt.n) { opt.n = 0; }  break;  case 'E': // display $ at end of each line  opt.E = 1;  break;  case 'e': // equivalent to -vE  opt.v = 1;  opt.E = 1;  break;  case 'n': // number all output lines  opt.n = 1;  if (opt.b) { opt.b = 0; }  break;  case 't': // equivalent to -vT  opt.v = 1;  opt.T = 1;  break;  case 'T': // display TAB characters as ^I  opt.T = 1;  break;  case 'v': // use ^ and M- notation, except for LFD a TAB  opt.v = 1;  break;  default:  break;  }  }  } |

옵션은 flag의 형태로 opt라는 구조체를 만들어서 해당 옵션이 들어오면 set, 들어오지 않는다면 clear되게 만들었다.

|  |
| --- |
| …  lineNum = 1;  while (fgets(buf, sizeof(buf), fp) != NULL) {  strcpy(line, buf);  // 옵션 n과 b는 양립할 수 없기에 if - else if 로 묶어주었다.  // 옵션 n이 set되어있다면 각 줄마다 lineNum을 증가시켜주고 각 line앞에 lineNum을 추가하여 출력한다.  if (opt.n) {  sprintf(line, " %d %s", lineNum, buf);  lineNum++;  }  // 옵션 b가 set되어있다면 공백이 아닌 각 줄마다 lineNum을 증가시켜주고 각 line앞에 lineNum을 추가하여 출력한다.  else if (opt.b) {  if (strcmp(buf, "\n")) {  sprintf(line, " %d %s", lineNum, buf);  lineNum++;  }  }  // 옵션 E가 set되어있다면 line변수에 OptionE()를 사용하여 새로운 문자열을 담는다.  if (opt.E) {  line = OptionE(line);  }  // 옵션 T가 set되어있다면 line변수에 OptionT()를 사용하여 새로운 문자열을 담는다.  if (opt.T) {  line = OptionT(line);  }  printf("%s", line);  }  … |

옵션이 들어올 경우 해당옵션을 수행하고 최종 결과물을 만든다.

1. 옵션
   1. -b & -n

|  |
| --- |
| // 옵션 n과 b는 양립할 수 없기에 if - else if 로 묶어주었다.  // 옵션 n이 set되어있다면 각 줄마다 lineNum을 증가시켜주고 각 line앞에 lineNum을 추가하여 출력한다.  if (opt.n) {  sprintf(line, " %d %s", lineNum, buf);  lineNum++;  }  // 옵션 b가 set되어있다면 공백이 아닌 각 줄마다 lineNum을 증가시켜주고 각 line앞에 lineNum을 추가하여 출력한다.  else if (opt.b) {  if (strcmp(buf, "\n")) {  sprintf(line, " %d %s", lineNum, buf);  lineNum++;  }  } |

둘 다 번호를 붙혀주지만 -b는 공백인 라인은 번호를 붙히지 않는다. 아무것도 없는 공백인 line에도 개행문자는 있으므로 buf가 \n과 같으면 번호를 붙히지 않는 것으로 한다.

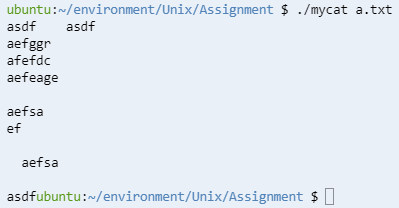
* 1. -E

|  |
| --- |
| /\*  옵션 E를 구현한 함수  문자열 마지막에 $를 붙히고 리턴한다..  \*/  char\* OptionE(char\* line) {  int n, i, j = 0;  n = strlen(line);  char\* buf = (char\*)calloc(n + 1, sizeof(char\*)); // 새로운 문자열 변수를 할당한다. 문자열의 크기는 기존의 문자열보다 1개 더 큰 n+1만큼 증가시켜준다.  for (i = 0; i < n; i++) {  if (line[i] == '\n') {  j = i;  }  }  if (j == 0) { return line; }  strcpy(buf, line);  for (i = n + 1; i > j; i--) {  buf[i] = buf[i - 1];  }  buf[n - 1] = '$'; // 문자열 마지막 부분에 '$'를 붙혀준다.  return buf;  }    …  // 옵션 E가 set되어있다면 line변수에 OptionE()를 사용하여 새로운 문자열을 담는다.  if (opt.E) {  line = OptionE(line);  }  … |

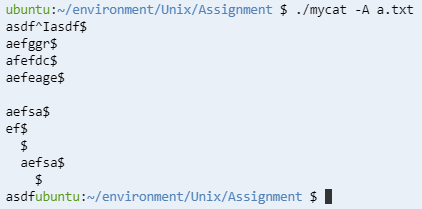
* 1. -T

|  |
| --- |
| /\*  옵션 T를 구현한 함수.  문자열을 받아 모든 탭을 ^I문자로 바꾸어준 후 문자열을 리턴한다.  \*/  char\* OptionT(char\* line) {  int n = strlen(line);  int i, j;  char\* buf;  for (i = 0; i < n; i++) {  if (line[i] == '\t') {  buf = (char\*)calloc(n + 1, sizeof(char)); // 새로운 문자열 변수를 할당한다. 문자열의 크기는 기존의 문자열보다 1개 더 큰 n+1만큼 증가시켜준다.  strcpy(buf, line);  for (j = n; j > i + 1; j--) {  buf[j] = buf[j - 1];  }  buf[i] = '^';  buf[i + 1] = 'I';  n++;  line = buf;  }  }  return line;  }  // 옵션 T가 set되어있다면 line변수에 OptionT()를 사용하여 새로운 문자열을 담는다.  if (opt.T) {  line = OptionT(line);  } |

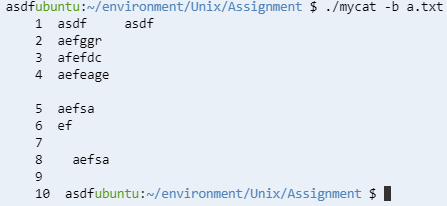
* **수행 예제**
  1. 기본기능



* 1. -A



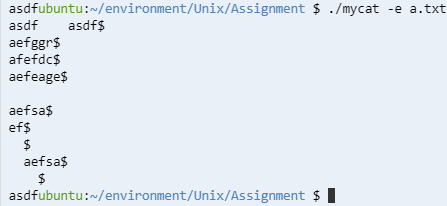
* 1. -b



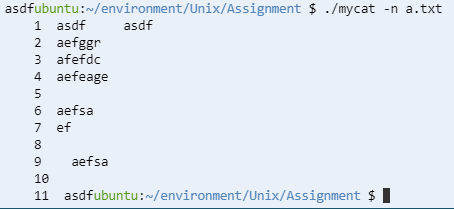
* 1. -E



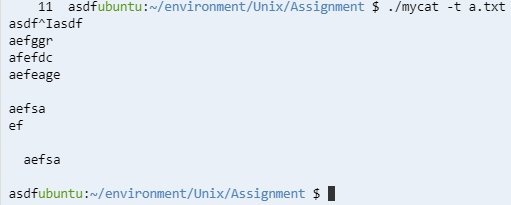
* 1. -e



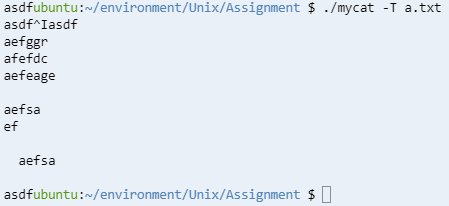
* 1. -n



* 1. -t



* 1. -T



1. **mycp source dest / source directory**

* **아이디어**

1. 3가지로 나누어 구현하였다
   1. mycp src dest : 2번째 인자가 파일일경우
   2. mycp src directory : 2번째 인자가 디렉토리일 경우
   3. mycp src1, src2, … srcN directory : 인자가 3개이상이고 마지막 인자가 디렉토리인 경우
2. copy동작은 2개의 fopen()함수를 “r”과 “w”로 열어 read로 받은 파일포인터를 이용하여 src의 내용을 fgets로 받아 fprintf()로 dest로 복사하였다.
3. 옵션과 관련된 아이디어는 mycat과 같다.

* **구현범위**

1. 기존의 cp는 src가 디렉토리인 경우 -r 옵션을 이용하여 디렉토리까지 복사가 가능했지만 -r옵션을 구현하지 못하여 src가 디렉토리인 경우는 구현하지 못했다.
2. 구현한 옵션은 총 6개이다.
   1. -n : 파일이 이미 존재하면 덮어쓰지 않는다.
   2. -I : 덮어쓰기 전에 물어본다.
   3. -l : copy하는 대신 하드링크를 만든다.
   4. -s : copy하는 대신 심볼릭 링크를 만든다.
   5. -v : 어디서 어디로 복사했는지 설명해 준다.
   6. -b : 백업파일을 만든다.

* **동작에 대한 설명**

1. 기본동작

|  |
| --- |
| int main(int argc, char\* argv[]) {  char buf[256];  DIR\* dir;  char path[256];  int n, i, c;  extern char\* optarg;  extern int optind;  **…**  **…**  if (argc - optind == 2) { // 1.인자가 2개일 때  // src가 디렉토리라면 종료한다.  if (isDir(argv[optind])) {  printf("mycp: -r not specified; omitting directory '%s'\n", argv[optind]);  exit(1);  }  // case 1  if ((dir = opendir(argv[optind + 1])) == NULL) { // 2번째 인자가 디렉토리가 아닐 경우  CopyFileWithOption(argv[optind], argv[optind + 1]);  }  // case 2  else {  sprintf(path, "%s/%s", argv[optind + 1], argv[optind]);  CopyFileWithOption(argv[optind], path);  }  closedir(dir);  }  // case 3  else if (argc - optind > 2) {  n = argc - 1;  if ((dir = opendir(argv[n])) == NULL) {  perror(argv[n]);  exit(1);  }  // argv[1]은 -n  for (i = optind; i < n; i++) {  sprintf(path, "%s/%s", argv[n], argv[i]);  // src가 디렉토리라면 종료한다.  if (isDir(argv[i])) {  printf("mycp: -r not specified; omitting directory '%s'\n", argv[i]);  exit(1);  }  CopyFileWithOption(argv[optind], path);  }  closedir(dir);  }  else if (argc - optind == 1) { // 인자가 1개일 때  printf("mycp: missing destination file operand after '%s'\n", argv[optind]);  }  else {  printf("mycp: missing file operand\n");  }  return 0;  }  /\*  기본기능인 파일을 복사하는 함수이다.  src를 dest로 복사한다.  \*/  void copyFile(char\* src, char\* dest) {  FILE\* rfp, \* wfp , \*bOptionfp;  DIR \* dir ;  char buf[256], \*newDirectory, \*oldDirectory;  struct dirent \*dent;  **…**  /\*  기본기능.  2개의 파일 포인터로 각각 src와 dest의 파일을 열고  src에서 fgets()로 읽은 값을 buf에 담아 dest에 쓴다.  \*/  else {  if ((rfp = fopen(src, "r")) == NULL) {  perror(src);  exit(1);  }  if ((wfp = fopen(dest, "w")) == NULL) {  perror(dest);  exit(1);  }  // 옵션 b가 set되어있다면 백업파일을 위한 파일포인터를 받는다.  if (opt.b && ((bOptionfp = fopen(addWaveString(dest), "w")) == NULL)) {  perror(addWaveString(dest));  exit(1);  }  while (fgets(buf, sizeof(buf), rfp) != NULL) {  fprintf(wfp, "%s", buf);  if (opt.b) {  //옵션 b가 set 되어있다면 백업파일에도 write를 한다.  fprintf(bOptionfp, "%s", buf);  }  }  fclose(rfp);  fclose(wfp);  if (opt.b) { fclose(bOptionfp); }  } |

copyFile()이라는 함수를 만들었다. copyFile()에는 rfp, wfp, fgets(), fprintf() 를 이용하여 파일을 읽고 썼다.

1. 옵션의 동작 : 기본적인 옵션의 동작은 mycat과 마찬가지로 getopt()와 opt구조체를 활용한다.
2. 옵션
   1. -n & -i

|  |
| --- |
| while ((c = getopt(argc, argv, "nilsvb")) != -1) {  switch (c) {  case 'n': // do not overwrite an existing file  opt.n = 1;  if (opt.i == 1) { opt.i = 0; } // i가 set되어있다면 clear한다.  break;  case 'i':// prompt before overwrite  opt.i = 1;  if (opt.n == 1) { opt.n = 0; } // n이 set되어있다면 clear한다.  break;  **…**  void CopyFileWithOption(char\* src, char\* dest) {  if (!isExistFile(dest)) {  copyFile(src, dest);  }  else {  if (opt.i) {  if (tellMeYesOrNo(dest)) {  copyFile(src, dest);  }  }  else if (opt.n) {  // do noting  }  else {  copyFile(src, dest);  }  }  }  /\*  파일이 존재하는 지 확인하는 함수이다.  존재하면 1, 아니면 0을 리턴한다.  \*/  int isExistFile(char\* filename) {  FILE\* fp;  if ((fp = fopen(filename, "r")) != NULL) {  return 1;  }  else {  return 0;  }  return 0;  }  /\*  옵션 i를 쓰면 overwrite를 할 지 말지를 결정하게 된다.  y로 시작하는 단어를 쓰면 1, n로 시작하는 단어를 쓰면 0을 리턴한다.  \*/  int tellMeYesOrNo(char\* filename) {  char\* s;  printf("mycp: overwrite '%s'?", filename);  scanf("%s", s);  if (s[0] == 'Y' || s[0] == 'y') {  return 1;  }  else if (s[0] == 'N' || s[0] == 'n') {  return 0;  }  return -1;  } |

-n과 -i 는 같이 사용하면 한쪽이 덮어써진다. 따라서 옵션 n을 선택했을 때 i가 set되어있다면 clear시켜주고 i를 선택했을 때도 n을 clear시켜주었다.

-n과 -I 그리고 기본 copy는 서로 양립할 수 없다. 따라서 위의 3가지를 if … else if … else로 묶어주었다.

* 1. -l & -s

|  |
| --- |
| void copyFile(char\* src, char\* dest) {  FILE\* rfp, \* wfp , \*bOptionfp;  DIR \* dir ;  char buf[256], \*newDirectory, \*oldDirectory;  struct dirent \*dent;  /\*  옵션 l과 옵션 s는 서로 양립할 수없다.  \*/  if (opt.l && opt.s) {  printf("mycp: cannot make both hard and symbolic links");  exit(1);  }  /\*  옵션 l이 set되어 있다면 copy 하는 대신 hard link를 만든다.  \*/  else if (opt.l) {  if (link(src, dest) == -1) {  perror(dest);  exit(1);  }  }  /\*  옵션 s가 set되어 있다면 copy 하는 대신 symbolic link를 만든다.  \*/  else if (opt.s) {  if (symlink(src, dest) == -1) {  perror(dest);  exit(1);  }  }  /\*  기본기능.  2개의 파일 포인터로 각각 src와 dest의 파일을 열고  src에서 fgets()로 읽은 값을 buf에 담아 dest에 쓴다.  \*/  else {  … |

copyFile()함수에서 기본으로 복사를 하는 함수에서 -s 또는 -l 이 되어있다면 파일포인터를 통한 copy대신 하드링크, 심볼릭 링크를 만든다. 둘은 서로 같이 사용하면 같이 사용할 수 없다는 메시지를 출력한다.

link(), symlink()함수를 이용하여 구현하였다.

* 1. -v

|  |
| --- |
| /\*  기본기능인 파일을 복사하는 함수이다.  src를 dest로 복사한다.  \*/  void copyFile(char\* src, char\* dest) {  FILE\* rfp, \* wfp , \*bOptionfp;  DIR \* dir ;  char buf[256], \*newDirectory, \*oldDirectory;  struct dirent \*dent;  **…**  /\*  옵션 v가 set 되어있다면 src -> dest인 과정을 출력한다.  \*/  if (opt.v) {  printf("'%s' -> '%s' ", src, dest);  if (opt.b) { printf("(backup: '%s')", addWaveString(dest)); } // 옵션 b가 set되어있다면 backup파일도 표시해준다.  printf("\n");  }  } |

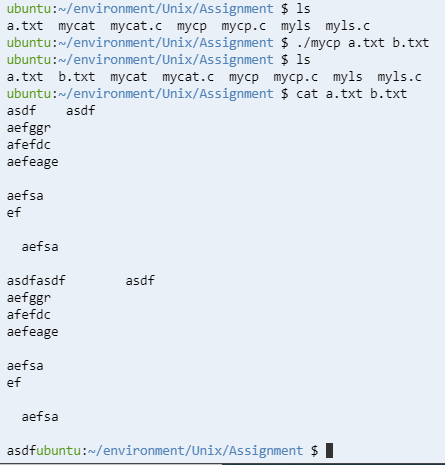
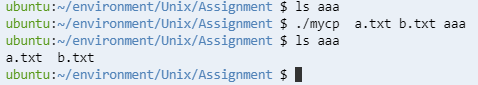
copyFile() 함수에서 복사 완료후 src -> dest를 출력해 준다.

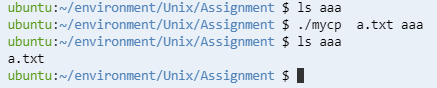
* 1. -b

|  |
| --- |
| /\*  옵션 b를 위한 함수  인자로 받은 문자열 뒤에 '~' 문자를 넣어준다.  \*/  char\* addWaveString(char\* line) {  int n, i, j = 0;  n = strlen(line);  char\* buf = (char\*)calloc(n + 1, sizeof(char\*)); // 새로운 문자열 변수를 할당한다. 문자열의 크기는 기존의 문자열보다 1개 더 큰 n+1만큼 증가시켜준다.  strcpy(buf, line);  for (i = n + 1; i > n-1; i--) {  buf[i] = buf[i - 1];  }  buf[n] = '~'; // 문자열 마지막 부분에 '~'를 붙혀준다.  return buf;  }  /\*  기본기능인 파일을 복사하는 함수이다.  src를 dest로 복사한다.  \*/  void copyFile(char\* src, char\* dest) {  /\*  기본기능.  2개의 파일 포인터로 각각 src와 dest의 파일을 열고  src에서 fgets()로 읽은 값을 buf에 담아 dest에 쓴다.  \*/  else {  if ((rfp = fopen(src, "r")) == NULL) {  perror(src);  exit(1);  }  if ((wfp = fopen(dest, "w")) == NULL) {  perror(dest);  exit(1);  }  // 옵션 b가 set되어있다면 백업파일을 위한 파일포인터를 받는다.  if (opt.b && ((bOptionfp = fopen(addWaveString(dest), "w")) == NULL)) {  perror(addWaveString(dest));  exit(1);  }  while (fgets(buf, sizeof(buf), rfp) != NULL) {  fprintf(wfp, "%s", buf);  if (opt.b) {  //옵션 b가 set 되어있다면 백업파일에도 write를 한다.  fprintf(bOptionfp, "%s", buf);  }  }  fclose(rfp);  fclose(wfp);  if (opt.b) { fclose(bOptionfp); }  }  **}** |

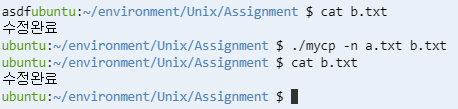
옵션 b가 set되어있다면 파일포인터를 하나 더 선언하여 dest에 ~문자를 추가한 파일을 하나 더 생성한다.

* 수행 예제
  1. 기본기능

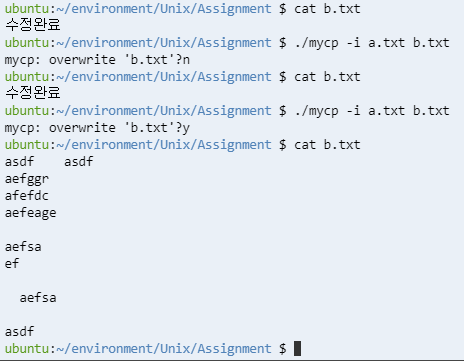
 



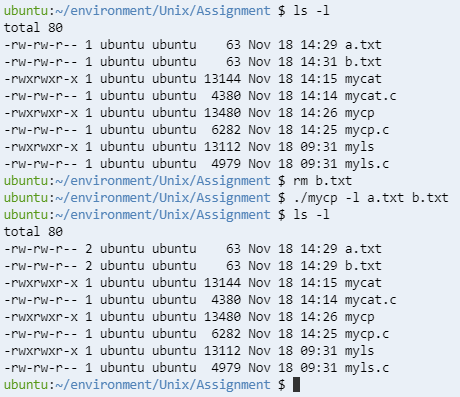
* 1. -n



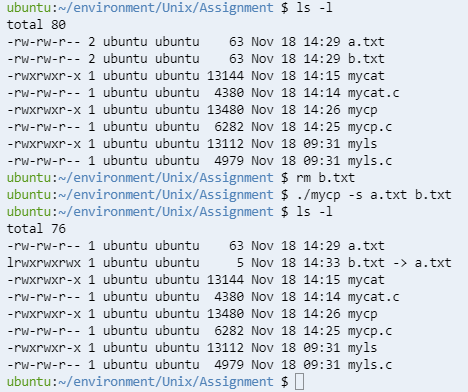
* 1. -i



* 1. -l



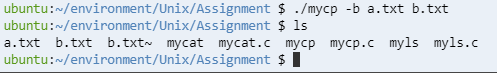
* 1. -s



* 1. -v



* 1. -b



* 1. -vb



1. **myls [option] [file]**

* 아이디어

1. opendir(), scandir(), readdir(), dirent구조체를 이용하여 해당 디렉토리의 파일들의 이름을 구한다.
2. 인자로 들어 온 것이 파일일때와 디렉토리일 때 둘 다 아닐 때 동작하는 방식이 다르다.
   1. 디렉토리일 경우 : opendir()로 해당 디렉토리를 열고 scandir()사용하여 dirent구조체에 해당 디렉토리의 파일들의 배열이 들어간다. 차례로 dirent구조체에 있는 파일의 이름을 출력한다.
   2. 디렉토리가 아닐 경우 : opendir()로 현재 디렉토리를 연다. Scandir()을 사용하여 현재 디렉토리에 인자와 같은 이름의 파일이 있는 지 검사한다. 있다면 출력한다.

* 구현범위

1. -m 옵션 구현에 있어 출력되는 파일 및 디렉토리 사이에만 ‘,’가 출력되야 하는데 마지막 파일(또는 디렉토리)이후에 ,가 찍힌다.
2. 인자가 없을 경우 현재 디렉토리를 인자로 넣어준다.
3. 인자가 2개 이상이고 그 중 디렉토리가 하나이상 포함되어 있다면 디렉토리의 내용물을 출력하기 전 “디렉토리 명 : “을 출력한다.
4. 구현한 옵션은 6개이다.
   1. -a : 숨김파일(.로 시작)을 무시하고 출력한다.
   2. -d : 디렉토리도 파일로 취급하고 출력한다.
   3. -f : 정렬하지 않고 출력한다.
   4. -I : 파일 이름앞에 inode값도 출력한다.
   5. -m : 각각의 파일(또는 디렉토리)사이에 ‘,’로 구분해 준다.
   6. -r : 정렬방향을 거꾸로한다.

* 동작에 대한 설명

1. 기본동작

|  |
| --- |
| int main(int argc, char\* argv[]) {  int n, i, c;  extern char\* optarg;  extern int optind;  **…**  if (argc == optind) { argv[optind] = "."; } // 인자가 없다면 기본으로 현재 디렉토리를 argv에 넣어준다.  if (argc - optind <= 1) { // 인자가 1개일 때  printLists(argv[optind]);  }  /\*  인자가 2개 이상일 때 디렉토리가 있다면  디렉토리 안 파일들이 출력되기 전 '디렉토리명 : ' 이 출력된다.  \*/  else { //  for (i = optind; i < argc; i++) {  if (!isDir(argv[i])) {  printLists(argv[i]);  if (argc - optind > 1) { printf("\n"); }  }  }  for (i = optind; i < argc; i++) {  if (isDir(argv[i])) {  if (argc - optind > 1) printf("%s:\n", argv[i]); // "디렉토리명 :"  printLists(argv[i]);  if ((argc - optind > 1) && (i != argc - 1)) { printf("\n"); }  }  }  }  return 0;  }  /\*  ls의 기본기능 및 옵션을 구현한 함수  어떤 옵션이 set되어있는 지에 따라 출력하는 범위 및 동작이 달라진다.  기본 동작 순서  1. 인자로 들어온 filename이 디렉토리 인지 검사한다.  a. 디렉토리라면 opendir로 filename을 연다.  옵션에 따라 readdir, scandir등의 함수로 파일 내용을 검색하여 출력한다.  b. 디렉토리가 아니라면 opendir로 현재디렉토리 (".")를 연다.  readdir로 현재 디렉토리의 파일을 읽으며 filename과 같은 파일이 있다면 출력한다.  \*/  void printLists(char\* filename) {  struct stat st;  DIR\* dir;  struct dirent\* dent, \*\* namelist;  int n = 0, i = 0;  char\* path;  // if filename is a directory  // 옵션 d 가 set된다면 opendir로 열지 않고 파일이 들어왔을 때 처럼 취급한다.  if (((dir = opendir(filename)) != NULL) && !opt.d) {  **…**  n = scandir(filename, &namelist, NULL, alphasort); // 기본기능. 파일을 알파벳 순서로 출력    for (i = opt.r ? n-1 : 0 ; opt.r ? i > 0 : i < n; opt.r ? i-- :i++ ) { // 옵션 r이 set되었다면 내림차순, 아니라면 오름차순으로 출력  // 옵션 a 가 set된다면 .으로 시작하는(숨김파일)파일도 출력한다.  if (namelist[i]->d\_name[0] == '.' && !opt.a) { continue; } // 옵션 a가 set된다면 숨김파일을 출력  else {  sprintf(path, "%s/%s", filename, namelist[i]->d\_name);  stat(path, &st);  if (opt.i) { printf("%d ", (int)st.st\_ino); } // 옵션 i 가 set된다면 이름을 출력하기 전 inode값을 출력한다.  printf("%s", namelist[i]->d\_name);  if (opt.m) { printf(","); }  printf("\t");  }  }  for (i = 0; i < n; i++) {  free(namelist[i]);  }  free(namelist);  }  printf("\n");  closedir(dir);  }  // if filename is a file  else {  dir = opendir(".");  while ((dent = readdir(dir)) != NULL) {  if (!strcmp(dent->d\_name, filename)) {  sprintf(path, "%s/%s", filename, dent->d\_name);  stat(path, &st);  if (opt.i) { printf("%d ", (int)st.st\_ino); } // 옵션 i 가 set된다면 이름을 출력하기 전 inode값을 출력한다.  printf("%s", dent->d\_name);  if (opt.m) { printf(","); } // 옵션 m 이 set된다면 출력되는 파일의 이름 사이에 , 를 출력한다.  printf("\t");  }  }  printf("\n");  closedir(dir);  }  } |

인자가 없을 경우 현재 디렉토리를 인자로 넣어준다. 인자가 2개 이상이고 디렉토리가 하나이상 포함되어 있다면 디렉토리의 내용물을 출력하기 전 “디렉토리 명 : “을 출력한다.

각각의 경우에 printList()라는 함수를 만들어 출력하였다.

1. 옵션
   1. **-**a

|  |
| --- |
| void printLists(char\* filename) {  …  for (i = opt.r ? n-1 : 0 ; opt.r ? i > 0 : i < n; opt.r ? i-- :i++ ) { // 옵션 r이 set되었다면 내림차순, 아니라면 오름차순으로 출력  // 옵션 a 가 set된다면 .으로 시작하는(숨김파일)파일도 출력한다.  if (namelist[i]->d\_name[0] == '.' && !opt.a) { continue; } // 옵션 a가 set된다면 숨김파일을 출력  else {  sprintf(path, "%s/%s", filename, namelist[i]->d\_name);  stat(path, &st);  if (opt.i) { printf("%d ", (int)st.st\_ino); } // 옵션 i 가 set된다면 이름을 출력하기 전 inode값을 출력한다.  printf("%s", namelist[i]->d\_name);  if (opt.m) { printf(","); }  printf("\t");  }  } |

if (namelist[i]->d\_name[0] == '.' && !opt.a) { continue; } // 옵션 a가 set된다면 숨김파일을 출력 를 이용하여 -a이 set된다면 전체 if문은 항상 false가 되어 .으로 시작하는 파일도 출력한다.

* 1. **-**d

|  |
| --- |
| void printLists(char\* filename) {  // if filename is a directory  // 옵션 d 가 set된다면 opendir로 열지 않고 파일이 들어왔을 때 처럼 취급한다.  if (((dir = opendir(filename)) != NULL) && !opt.d) { … }  else {  dir = opendir(".");  while ((dent = readdir(dir)) != NULL) {  if (!strcmp(dent->d\_name, filename)) {  sprintf(path, "%s/%s", filename, dent->d\_name);  stat(path, &st);  if (opt.i) { printf("%d ", (int)st.st\_ino); } // 옵션 i 가 set된다면 이름을 출력하기 전 inode값을 출력한다.  printf("%s", dent->d\_name);  if (opt.m) { printf(","); } // 옵션 m 이 set된다면 출력되는 파일의 이름 사이에 , 를 출력한다.  printf("\t");  }  }  printf("\n");  closedir(dir);  }  **}** |

-d가 set된다면 if (((dir = opendir(filename)) != NULL) && !opt.d) 이 if문은 항상 false가 되어 else문을 수행한다. 그렇다면 현재 디렉토리를 열어 디렉토리와 같은 이름이 있는 지 확인하고 출력하게 된다.

* 1. -f

|  |
| --- |
| // 옵션 f 가 set된다면 scandir로 정렬하며 파일을 읽지않고 그냥 파일을 읽는다.  if (opt.f) {  while ((dent = readdir(dir)) != NULL) {  sprintf(path, "%s/%s", filename, dent->d\_name);  stat(path, &st);  if (opt.i) { printf("%d ", (int)st.st\_ino); } // 옵션 i 가 set된다면 이름을 출력하기 전 inode값을 출력한다.  printf("%s", dent->d\_name); // 기본기능. 파일의 이름을 출력  if (opt.m) { printf(","); } // 옵션 m 이 set된다면 출력되는 파일의 이름 사이에 , 를 출력한다.  printf("\t"); // 기본기능. 파일 이름 사이에 탭 출력  }  } |

Scandir()로 디렉토리를 읽지않고 readdir()로 디렉토리를 읽는다.

* 1. -i

|  |
| --- |
| sprintf(path, "%s/%s", filename, dent->d\_name);  stat(path, &st);  if (opt.i) { printf("%d ", (int)st.st\_ino); } // 옵션 i 가 set된다면 이름을 출력하기 전 inode값을 출력한다.  printf("%s", dent->d\_name); // 기본기능. 파일의 이름을 출력 |

파일 및 디렉토리의 이름을 출력하기 전 inode값을 출력한다.

* 1. -m

|  |
| --- |
| printf("%s", dent->d\_name); // 기본기능. 파일의 이름을 출력  if (opt.m) { printf(","); } // 옵션 m 이 set된다면 출력되는 파일의 이름 사이에 , 를 출력한다.  printf("\t"); // 기본기능. 파일 이름 사이에 탭 출력 |

기본출력의 파일의 이름과 탭 사이 ,를 출력한다.

* 1. -r

|  |
| --- |
| n = scandir(filename, &namelist, NULL, alphasort); // 기본기능. 파일을 알파벳 순서로 출력    for (i = opt.r ? n-1 : 0 ; opt.r ? i > 0 : i < n; opt.r ? i-- :i++ ) { // 옵션 r이 set되었다면 내림차순, 아니라면 오름차순으로 출력  // 옵션 a 가 set된다면 .으로 시작하는(숨김파일)파일도 출력한다.  if (namelist[i]->d\_name[0] == '.' && !opt.a) { continue; } // 옵션 a가 set된다면 숨김파일을 출력  else {  sprintf(path, "%s/%s", filename, namelist[i]->d\_name);  stat(path, &st);  if (opt.i) { printf("%d ", (int)st.st\_ino); } // 옵션 i 가 set된다면 이름을 출력하기 전 inode값을 출력한다.  printf("%s", namelist[i]->d\_name);  if (opt.m) { printf(","); }  printf("\t");  }  } |

for (i = opt.r ? n-1 : 0 ; opt.r ? i > 0 : i < n; opt.r ? i-- :i++ )

삼항연산자를 이용하여

-r이 clear될 경우 : for (i = 0 ; i < n; i++ )

-r이 set될 경우 : for (i = n-1; i > 0; i--)

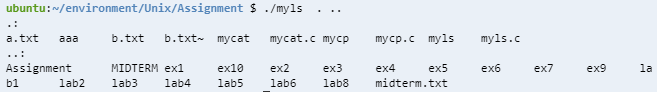
를 하나로 합쳤다.

-r이 set될 경우 내림차순으로 출력한다.

* 수행 예제

1. 기본기능





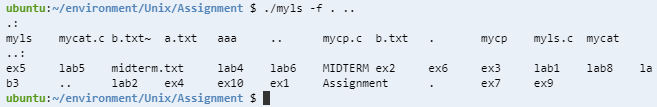
1. -a



1. -d



1. -f



1. -I



1. -m



1. -r



**mycp.c**

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

#include <unistd.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <string.h>

/\*

어떤 옵션을 넣어주었는 지 확인하는 구조체

옵션을 넣어주면 set된다.

서로 양립할 수 없는 옵션(서로 override되는 옵션)은 하나가 set되면 하나는 clear된다.

\*/

struct opt {

int n;

int i;

int l;

int s;

int v;

int b;

} opt;

/\*

옵션 b를 위한 함수

인자로 받은 문자열 뒤에 '~' 문자를 넣어준다.

\*/

char\* addWaveString(char\* line) {

int n, i, j = 0;

n = strlen(line);

char\* buf = (char\*)calloc(n + 1, sizeof(char\*)); // 새로운 문자열 변수를 할당한다. 문자열의 크기는 기존의 문자열보다 1개 더 큰 n+1만큼 증가시켜준다.

strcpy(buf, line);

for (i = n + 1; i > n-1; i--) {

buf[i] = buf[i - 1];

}

buf[n] = '~'; // 문자열 마지막 부분에 '~'를 붙혀준다.

return buf;

}

/\*

인자로 들어 온 filename이 디렉토리인지 확인하는 함수

디렉토리라면 1을 리턴한다.

\*/

int isDir(char\* filename) {

struct stat statbuf;

if (stat(filename, &statbuf) != 0)

return 0;

return S\_ISDIR(statbuf.st\_mode);

}

/\*

옵션 i를 쓰면 overwrite를 할 지 말지를 결정하게 된다.

y로 시작하는 단어를 쓰면 1, n로 시작하는 단어를 쓰면 0을 리턴한다.

\*/

int tellMeYesOrNo(char\* filename) {

char\* s;

printf("mycp: overwrite '%s'?", filename);

scanf("%s", s);

if (s[0] == 'Y' || s[0] == 'y') {

return 1;

}

else if (s[0] == 'N' || s[0] == 'n') {

return 0;

}

return -1;

}

/\*

파일이 존재하는 지 확인하는 함수이다.

존재하면 1, 아니면 0을 리턴한다.

\*/

int isExistFile(char\* filename) {

FILE\* fp;

if ((fp = fopen(filename, "r")) != NULL) {

return 1;

}

else {

return 0;

}

return 0;

}

/\*

기본기능인 파일을 복사하는 함수이다.

src를 dest로 복사한다.

\*/

void copyFile(char\* src, char\* dest) {

FILE\* rfp, \* wfp , \*bOptionfp;

DIR \* dir ;

char buf[256], \*newDirectory, \*oldDirectory;

struct dirent \*dent;

/\*

옵션 l과 옵션 s는 서로 양립할 수없다.

\*/

if (opt.l && opt.s) {

printf("mycp: cannot make both hard and symbolic links");

exit(1);

}

/\*

옵션 l이 set되어 있다면 copy 하는 대신 hard link를 만든다.

\*/

else if (opt.l) {

if (link(src, dest) == -1) {

perror(dest);

exit(1);

}

}

/\*

옵션 s가 set되어 있다면 copy 하는 대신 symbolic link를 만든다.

\*/

else if (opt.s) {

if (symlink(src, dest) == -1) {

perror(dest);

exit(1);

}

}

/\*

기본기능.

2개의 파일 포인터로 각각 src와 dest의 파일을 열고

src에서 fgets()로 읽은 값을 buf에 담아 dest에 쓴다.

\*/

else {

if ((rfp = fopen(src, "r")) == NULL) {

perror(src);

exit(1);

}

if ((wfp = fopen(dest, "w")) == NULL) {

perror(dest);

exit(1);

}

// 옵션 b가 set되어있다면 백업파일을 위한 파일포인터를 받는다.

if (opt.b && ((bOptionfp = fopen(addWaveString(dest), "w")) == NULL)) {

perror(addWaveString(dest));

exit(1);

}

while (fgets(buf, sizeof(buf), rfp) != NULL) {

fprintf(wfp, "%s", buf);

if (opt.b) {

//옵션 b가 set 되어있다면 백업파일에도 write를 한다.

fprintf(bOptionfp, "%s", buf);

}

}

fclose(rfp);

fclose(wfp);

if (opt.b) { fclose(bOptionfp); }

}

/\*

옵션 v가 set 되어있다면 src -> dest인 과정을 출력한다.

\*/

if (opt.v) {

printf("'%s' -> '%s' ", src, dest);

if (opt.b) { printf("(backup: '%s')", addWaveString(dest)); } // 옵션 b가 set되어있다면 backup파일도 표시해준다.

printf("\n");

}

}

void CopyFileWithOption(char\* src, char\* dest) {

if (!isExistFile(dest)) {

copyFile(src, dest);

}

else {

if (opt.i) {

if (tellMeYesOrNo(dest)) {

copyFile(src, dest);

}

}

else if (opt.n) {

// do noting

}

else {

copyFile(src, dest);

}

}

}

int main(int argc, char\* argv[]) {

char buf[256];

DIR\* dir;

char path[256];

int n, i, c;

extern char\* optarg;

extern int optind;

while ((c = getopt(argc, argv, "nilsvb")) != -1) {

switch (c) {

case 'n': // do not overwrite an existing file

opt.n = 1;

if (opt.i == 1) { opt.i = 0; } // i가 set되어있다면 clear한다.

break;

case 'i':// prompt before overwrite

opt.i = 1;

if (opt.n == 1) { opt.n = 0; } // n이 set되어있다면 clear한다.

break;

case 'l': // hard link files instead of copying

opt.l = 1;

break;

case 's': // make symbolic links instead of copying

opt.s = 1;

break;

case 'v': // explain what is being done

opt.v = 1;

break;

case 'b': // like --backup but does not accept an argument

opt.b = 1;

break;

default:

break;

}

}

/\*

3가지 경우로 나누어 코드를 작성하였다.

1.인자가 2개일때

case 1. mycp src dest : 2번째 인자가 파일일 때

case 2. mycp src Dir : 2번째 인자가 디렉토리일 때

2.인자가 3개 이상일 때

case 3. mycp src1 src2 ... srcN dir

\*/

if (argc - optind == 2) { // 1.인자가 2개일 때

// src가 디렉토리라면 종료한다.

if (isDir(argv[optind])) {

printf("mycp: -r not specified; omitting directory '%s'\n", argv[optind]);

exit(1);

}

// case 1

if ((dir = opendir(argv[optind + 1])) == NULL) { // 2번째 인자가 디렉토리가 아닐 경우

CopyFileWithOption(argv[optind], argv[optind + 1]);

}

// case 2

else {

sprintf(path, "%s/%s", argv[optind + 1], argv[optind]);

CopyFileWithOption(argv[optind], path);

}

closedir(dir);

}

// case 3

else if (argc - optind > 2) {

n = argc - 1;

if ((dir = opendir(argv[n])) == NULL) {

perror(argv[n]);

exit(1);

}

// argv[1]은 -n

for (i = optind; i < n; i++) {

sprintf(path, "%s/%s", argv[n], argv[i]);

// src가 디렉토리라면 종료한다.

if (isDir(argv[i])) {

printf("mycp: -r not specified; omitting directory '%s'\n", argv[i]);

exit(1);

}

CopyFileWithOption(argv[optind], path);

}

closedir(dir);

}

else if (argc - optind == 1) { // 인자가 1개일 때

printf("mycp: missing destination file operand after '%s'\n", argv[optind]);

}

else {

printf("mycp: missing file operand\n");

}

return 0;

}

**mycat.c**

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <stdlib.h>

#include <dirent.h>

/\*

어떤 옵션을 넣어주었는 지 확인하는 구조체

옵션을 넣어주면 set된다.

서로 양립할 수 없는 옵션(서로 override되는 옵션)은 하나가 set되면 하나는 clear된다.

\*/

struct opt {

int b;

int n;

int E;

int v;

int T;

};

/\*

인자로 들어 온 filename이 디렉토리인지 확인하는 함수

디렉토리라면 1을 리턴한다.

\*/

int isDir(char\* filename) {

struct stat statbuf;

if (stat(filename, &statbuf) != 0)

return 0;

return S\_ISDIR(statbuf.st\_mode);

}

/\*

옵션 T를 구현한 함수.

문자열을 받아 모든 탭을 ^I문자로 바꾸어준 후 문자열을 리턴한다.

\*/

char\* OptionT(char\* line) {

int n = strlen(line);

int i, j;

char\* buf;

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

if (line[i] == '\t') {

buf = (char\*)calloc(n + 1, sizeof(char)); // 새로운 문자열 변수를 할당한다. 문자열의 크기는 기존의 문자열보다 1개 더 큰 n+1만큼 증가시켜준다.

strcpy(buf, line);

for (j = n; j > i + 1; j--) {

buf[j] = buf[j - 1];

}

buf[i] = '^';

buf[i + 1] = 'I';

n++;

line = buf;

}

}

return line;

}

/\*

옵션 E를 구현한 함수

문자열 마지막에 $를 붙히고 리턴한다..

\*/

char\* OptionE(char\* line) {

int n, i, j = 0;

n = strlen(line);

char\* buf = (char\*)calloc(n + 1, sizeof(char\*)); // 새로운 문자열 변수를 할당한다. 문자열의 크기는 기존의 문자열보다 1개 더 큰 n+1만큼 증가시켜준다.

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

if (line[i] == '\n') {

j = i;

}

}

if (j == 0) { return line; }

strcpy(buf, line);

for (i = n + 1; i > j; i--) {

buf[i] = buf[i - 1];

}

buf[n - 1] = '$'; // 문자열 마지막 부분에 '$'를 붙혀준다.

return buf;

}

/\*

파일포인터로 파일을 열고 fgets로 버퍼에 한줄씩 담아 출력한다.

\*/

int main(int argc, char\* argv[]) {

FILE\* fp;

char buf[256];

char\* line = (char\*)calloc(256, sizeof(char));

int c;

int i = 1, lineNum;

struct opt opt = {

.b = 0, .E = 0,.n = 0, .v = 0 , .T = 0

};

extern char\* optarg;

extern int optind;

// v는 구현 못함

while ((c = getopt(argc, argv, "AbEnetTv")) != -1) {

switch (c) {

case 'A': // equivalent to -vET

opt.v = 1;

opt.E = 1;

opt.T = 1;

break;

case 'b': // number nonempty output lines

opt.b = 1;

if (opt.n) { opt.n = 0; }

break;

case 'E': // display $ at end of each line

opt.E = 1;

break;

case 'e': // equivalent to -vE

opt.v = 1;

opt.E = 1;

break;

case 'n': // number all output lines

opt.n = 1;

if (opt.b) { opt.b = 0; }

break;

case 't': // equivalent to -vT

opt.v = 1;

opt.T = 1;

break;

case 'T': // display TAB characters as ^I

opt.T = 1;

break;

case 'v': // use ^ and M- notation, except for LFD a TAB

opt.v = 1;

break;

default:

break;

}

}

// 인자가 없을때 함수를 종료한다.

if (argc == optind) {

return 0;

}

// 모든 인자에 대해 수행한다.

for (i = optind; i < argc; i++) {

// argv가 디렉토리라면 다음 인자로 넘어간다.

if (isDir(argv[i])) {

printf("mycat: %s: Is a directory\n", argv[i]);

continue;

}

if ((fp = fopen(argv[i], "r")) == NULL) {

perror(argv[i]);

exit(1);

}

lineNum = 1;

while (fgets(buf, sizeof(buf), fp) != NULL) {

strcpy(line, buf);

// 옵션 n과 b는 양립할 수 없기에 if - else if 로 묶어주었다.

// 옵션 n이 set되어있다면 각 줄마다 lineNum을 증가시켜주고 각 line앞에 lineNum을 추가하여 출력한다.

if (opt.n) {

sprintf(line, " %d %s", lineNum, buf);

lineNum++;

}

// 옵션 b가 set되어있다면 공백이 아닌 각 줄마다 lineNum을 증가시켜주고 각 line앞에 lineNum을 추가하여 출력한다.

else if (opt.b) {

if (strcmp(buf, "\n")) {

sprintf(line, " %d %s", lineNum, buf);

lineNum++;

}

}

// 옵션 E가 set되어있다면 line변수에 OptionE()를 사용하여 새로운 문자열을 담는다.

if (opt.E) {

line = OptionE(line);

}

// 옵션 T가 set되어있다면 line변수에 OptionT()를 사용하여 새로운 문자열을 담는다.

if (opt.T) {

line = OptionT(line);

}

printf("%s", line);

}

}

fclose(fp);

return 0;

}

**myls.c**

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

#include <string.h>

#include <unistd.h>

#include <sys/stat.h>

#include <sys/types.h>

/\*

어떤 옵션을 넣어주었는 지 확인하는 구조체

옵션을 넣어주면 set된다.

서로 양립할 수 없는 옵션(서로 override되는 옵션)은 하나가 set되면 하나는 clear된다.

\*/

struct opt {

int a;

int d;

int f;

int i;

int m;

int r;

}opt;

/\*

인자로 들어 온 filename이 디렉토리인지 확인하는 함수

디렉토리라면 1을 리턴한다.

\*/

int isDir(char\* filename) {

struct stat statbuf;

if (stat(filename, &statbuf) != 0)

return 0;

return S\_ISDIR(statbuf.st\_mode);

}

/\*

ls의 기본기능 및 옵션을 구현한 함수

어떤 옵션이 set되어있는 지에 따라 출력하는 범위 및 동작이 달라진다.

기본 동작 순서

1. 인자로 들어온 filename이 디렉토리 인지 검사한다.

a. 디렉토리라면 opendir로 filename을 연다.

옵션에 따라 readdir, scandir등의 함수로 파일 내용을 검색하여 출력한다.

b. 디렉토리가 아니라면 opendir로 현재디렉토리 (".")를 연다.

readdir로 현재 디렉토리의 파일을 읽으며 filename과 같은 파일이 있다면 출력한다.

\*/

void printLists(char\* filename) {

struct stat st;

DIR\* dir;

struct dirent\* dent, \*\* namelist;

int n = 0, i = 0;

char\* path;

// if filename is a directory

// 옵션 d 가 set된다면 opendir로 열지 않고 파일이 들어왔을 때 처럼 취급한다.

if (((dir = opendir(filename)) != NULL) && !opt.d) {

// 옵션 f 가 set된다면 scandir로 정렬하며 파일을 읽지않고 그냥 파일을 읽는다.

if (opt.f) {

while ((dent = readdir(dir)) != NULL) {

sprintf(path, "%s/%s", filename, dent->d\_name);

stat(path, &st);

if (opt.i) { printf("%d ", (int)st.st\_ino); } // 옵션 i 가 set된다면 이름을 출력하기 전 inode값을 출력한다.

printf("%s", dent->d\_name); // 기본기능. 파일의 이름을 출력

if (opt.m) { printf(","); } // 옵션 m 이 set된다면 출력되는 파일의 이름 사이에 , 를 출력한다.

printf("\t"); // 기본기능. 파일 이름 사이에 탭 출력

}

}

else {

n = scandir(filename, &namelist, NULL, alphasort); // 기본기능. 파일을 알파벳 순서로 출력

for (i = opt.r ? n - 1 : 0; opt.r ? i > 0 : i < n; opt.r ? i-- : i++) {

// 옵션 a 가 set된다면 .으로 시작하는(숨김파일)파일도 출력한다.

if (namelist[i]->d\_name[0] == '.' && !opt.a) { continue; }

else {

sprintf(path, "%s/%s", filename, namelist[i]->d\_name);

stat(path, &st);

if (opt.i) { printf("%d ", (int)st.st\_ino); } // 옵션 i 가 set된다면 이름을 출력하기 전 inode값을 출력한다.

printf("%s", namelist[i]->d\_name);

if (opt.m) { printf(","); }

printf("\t");

}

}

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

free(namelist[i]);

}

free(namelist);

}

// closedir(dir);

}

// if filename is a file

else {

dir = opendir(".");

while ((dent = readdir(dir)) != NULL) {

if (!strcmp(dent->d\_name, filename)) {

sprintf(path, "%s/%s", filename, dent->d\_name);

stat(path, &st);

if (opt.i) { printf("%d ", (int)st.st\_ino); } // 옵션 i 가 set된다면 이름을 출력하기 전 inode값을 출력한다.

printf("%s", dent->d\_name);

if (opt.m) { printf(","); } // 옵션 m 이 set된다면 출력되는 파일의 이름 사이에 , 를 출력한다.

printf("\t");

}

}

// closedir(dir);

}

// closedir(dir);

}

int main(int argc, char\* argv[]) {

int n, i, c;

extern char\* optarg;

extern int optind;

while ((c = getopt(argc, argv, "adfimr")) != -1) {

switch (c) {

case 'a': // do not ignore entries starting with .

opt.a = 1;

break;

case 'd': // list directories themselves, not their contents

opt.d = 1;

break;

case 'f': // do not sort

opt.f = 1;

break;

case 'i': // print the index number of each file

opt.i = 1;

break;

case 'm': // fill width with a comma separated list of entries

opt.m = 1;

break;

case 'r':

opt.r = 1;

break;

default:

break;

}

}

if (argc == optind) { // 인자가 없다면 기본으로 현재 디렉토리를 argv에 넣어준다.

argv[optind] = ".";

argc++;

}

if (argc - optind <= 1) { // 인자가 1개일 때

printLists(argv[optind]);

printf("\n");

}

/\*

인자가 2개 이상일 때 디렉토리가 있다면

디렉토리 안 파일들이 출력되기 전 '디렉토리명 : ' 이 출력된다.

\*/

else {

for (i = optind; i < argc; i++) {

if (!isDir(argv[i]) || opt.d) {

printLists(argv[i]);

if (argc - i == 1) { printf("\n"); }

}

}

for (i = optind; i < argc; i++) {

if (isDir(argv[i]) && !opt.d) {

if (argc - optind > 1) printf("%s:\n", argv[i]); // "디렉토리명 :"

printLists(argv[i]);

printf("\n");

}

}

}

return 0;

}