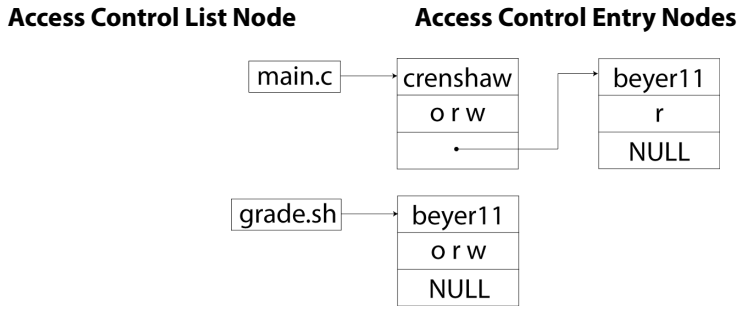
CS303 Programming Assignment #3: "Lovely Linked Lists"

Out:	Feb 20, 2013. **Due**: Mar. 1, 2013 by 7:00 pm on the CS303 Moodle site.
Total points:	100. approximately 20% of the total homework grade.
Lectures:	Lectures 16, 17 and 19 include linked-list topics including "printing a linked list" "bitwise operations" and "strategies for deleting from a linked list".

Background:

An Access Control List, or ACL, is the structure that the Windows Operating System uses to manage who may and may not access the set of files on a machine or network. An ACL may be implemented as a linked list in an operating system. For example, in the figure below are two files, main.c and grade.sh. Each file has its own linked list. The linked list for main.c shows that user 'crenshaw' has the rights 'o r w'. This means that she owns the file, may read the file, and can write to the file. The next entry in the list shows that beyer11 may read the file.



You will be implementing a small Access Control List data structure for a single file in C. It will consist of two parts. The first part is the Access Control List node which contains the file's name and a pointer to all the Access Control Entries for that file. The second part is the linked list of Access Control Entries. Each Access Control Entry stores the username and the rights that the user may invoke on the filename. It also stores a pointer to the next entry in the list.

In the Access Control Entry, users may have the following rights:

Own: The user owns the file.
Read: The user may read from the file.
Write: The user may write to the file.
Execute: The user may execute the file.

These rights are represented by numerical values, as defined in **aclist.h**.

```
// Define the rights for files.
#define R_OWN 8     // Binary = 0b1000
#define R_READ 4    // Binary = 0b0100
#define R_WRITE 2   // Binary = 0b0010
#define R_EXECUTE 1 // Binary = 0b0001
```

And these rights are stored in a 4-bit unsigned integer field called rights in the **`AccessControlEntry`** struct defined in **aclist.h**. For example, a user who has both read and write access to a file will have the value 0b0110, or 6, in the rights field.

You must extend an existing C program to fully implement a linked-list data structure to store and manipulate access control list (ACL) information.

You are provided with an already-working C program:

Download prog3.zip from the course website. The zipfile contains:

- **aclist.c**: A source file containing a partial implementation of the Access Control List data structure. It provides the functions initializeACL, addEntry() and printACL().
- **aclist.h**: A header file containing the function prototypes and constant declarations for aclist.c
- **main.c**: The main entry point of the program.
- **makefile**: A makefile to simplify compilation of prog3.
- **acl0.txt, acl1.txt, acl2.txt, acl3.txt**: Text files containing one Access Control List to be created.
- **commands1.txt, commands2.txt, commandsDel.txt**: Text files containing commands to be executed on the Access Control Lists in acl*.txt.

The working program has the following functionality:

1. It is invoked using: **`$ prog3 aclFile commandFile.`** If the file **`aclFile`** does not exist, it prints an error message and exits gracefully. Otherwise, it opens **`aclFile`** and attempts to parse it in order to create a new Access Control List. It prints the Access Control List created.
2. If **`commandFile`** does not exist, it prints an error message and exits gracefully. Otherwise, it opens **`commandFile`** and invokes a series of Access Control List functions on the Access Control List created in step 1. Finally, it prints the resulting Access Control List. Note that no changes will take place as the Access Control List functions have not yet been implemented.

Example compilation

```
$ make
gcc -c main.c
gcc -c aclist.c
gcc -o prog3 main.o aclist.o
```

or, if the make tool is not installed,

```
$ gcc -o prog3 main.c aclist.c
```

Example execution for starter program:

```
$ ./prog3
./prog3 error: incorrect number of parameters
usage: ./prog3 <aclFile> <commandFile>

./prog3 acl1.txt commands1.txt
acl1.txt was successfully opened.
Parsing access control entries from file.
Creating a new access control list for file: main.c.
printList: (File: main.c. jwhite04 (rights), ramon13 (rights), crenshaw
(rights), vegdahl (rights))

commands1.txt was successfully opened.
```

```
Parsing commands from file.
Delete right = 4 from user vegdahl
Delete right = 8 from user crenshaw
Add right = 8 to user jwhite04
Add right = 1 to user jwhite04
Add right = 2 to user jwhite04
Delete user ramon12
printList: (File: main.c. jwhite04 (rights), ramon13 (rights), crenshaw
(rights), vegdahl (rights))

Thanks for playing.
```

You must write a C program which extends prog3 with the following features (70 points):

1. Extend printACL() so that it appropriately prints the rights given to each user.
   Currently, when printACL() is invoked, it prints the original Access Control List like so:

   ```
   printList: (File: main.c. jwhite04 (rights), ramon13 (rights), crenshaw
   (rights), vegdahl (rights))
   ```

   Extend printEntry() so that it *actually* prints the rights for each user. Like so:

   ```
   printList: (File: main.c. jwhite04 (rw), ramon13 (x), crenshaw (orxw),
   vegdahl (r))
   ```

   where o represents own, r represents read, w represents write, and x represents execute.

2. Enhance the implementation of the Access Control List by adding a function called deleteRight(). The function prototype is provided for you in aclist.h. A function stub in aclist.c has also been provided. The function, deleteRight(), should remove a right for a given user in the Access Control List.

   For example, if the current Access Control List, acl, looks like this:

   ```
   printList: (File: main.c. crenshaw (orwx))
   ```

   Then after invoking deleteRight(R_READ, “crenshaw”, acl) should result in the following. Notice that the user crenshaw loses the right to read the file.

   ```
   printList: (File: main.c. crenshaw (owx))
   ```

   The function should perform appropriate error checking. If a NULL acl pointer is provided, it should return an error code. If the right provided is not R_OWN, R_READ, R_WRITE, or R_EXECUTE, it should return a error code. If a user is not in the access control list, it should return a error code.

3. Enhance the implementation of the Access Control List by adding a function called addRight(). The function prototype is provided for you in aclist.h. A function stub in aclist.c has also been provided. The function, addRight(), should add a right for a given user in the Access Control List.

   For example, if the current Access Control List, acl, looks like this:

   ```
   printList: (File: main.c. ramon13 (x))
   ```

   Then after invoking addRight(R_READ, "ramon13", acl) should result in the following. Notice that the user ramon13 gains the right to read the file.

   ```
   printList: (File: main.c. ramon13 (rx))
   ```

   The function should perform appropriate error checking. If a NULL acl pointer is provided, it should return an error code. If the right provided is not R_OWN, R_READ, R_WRITE, or R_EXECUTE, it should return an error code. If a user is not in the access control list, it should return a error code. If a user already has the specified right, it should not alter the list and return a success code.

4. Enhance the implementation of the Access Control List by adding a function called deleteEntry(). The function prototype is provided for you in aclist.h. A function stub in aclist.c has also been provided. The function, deleteEntry(), should delete an Access Control Entry for the given user in the Access Control List.

   For example, if the current Access Control List, acl, looks like this:

   ```
   printList: (File: main.c. vegdahl (r), crenshaw (orwx))
   ```

   Then after invoking deleteEntry("vegdahl", acl) should result in the following. Notice that the user vegdahl is gone.

   ```
   printList: (File: main.c. crenshaw (owx))
   ```

   The function should perform appropriate error checking. If a NULL acl pointer is provided, it should return an error code. If a user is not in the access control list, it should not alter the list and return a success code. Take care that you can delete entries from the front, middle, and end of the list.

Example Execution of Test Cases

Test 1: Does printACL() correctly print the rights initially granted to each user? (15 points)

```
$ ./prog3 acl1.txt commands1.txt
acl1.txt was successfully opened.
Parsing access control entries from file.
Creating a new access control list for file:
main.c.
printList: (File: main.c. , jwhite04 (rw),
ramon13 (x), crenshaw (orwx), vegdahl (r))
...
```

Test 2: Does the program correctly delete rights from a user? (15 points)

```
$ ./prog3 acl1.txt commandsDel.txt
acl1.txt was successfully opened.
Parsing access control entries from file.
Creating a new access control list for file:
main.c.
printList: (File: main.c. , jwhite04 (rw),
ramon13 (x), crenshaw (orwx), vegdahl (r))

commandsDel.txt was successfully opened.
Parsing commands from file.
Delete right = 4 from user vegdahl
Delete right = 8 from user crenshaw
Delete right = 1 from user jwhite04
Delete right = 2 from user ramon12
Delete right = 16 from user ramon12
Delete right = 2 from user betty

printList: (File: main.c. , jwhite04 (rw),
ramon13 (x), crenshaw (rwx), vegdahl ( ))
```

Test 3: Does the program correctly add rights to a user? (10 points)

```
$ ./prog3 acl1.txt commandsAdd.txt
acl1.txt was successfully opened.
Parsing access control entries from file.
Creating a new access control list for file:
main.c.
printList: (File: main.c. , jwhite04 (rw),
ramon13 (x), crenshaw (orwx), vegdahl (r))

commandsAdd.txt was successfully opened.
Parsing commands from file.
Add right = 1 to user vegdahl
Add right = 2 to user vegdahl
Add right = 8 to user jwhite04
Add right = 8 to user crenshaw
Add right = 4 to user ramon14
Add right = 8 to user satish08
printList: (File: main.c. , jwhite04 (orw),
ramon13 (x), crenshaw (orwx), vegdahl (rwx))

Thanks for playing.
```

Test 4: Does the program correctly delete entries from the Access Control List?

```
./prog3 acl2.txt commandsDe2.txt
acl2.txt was successfully opened.
Parsing access control entries from file.
Creating a new access control list for file:
grade.sh.
printList: (File: grade.sh. , satish08 (x), k
$sha12 (orwx), vegdahl (rw))

commandsDe2.txt was successfully opened.
Parsing commands from file.
Delete user k$sha12
Delete user vegdahl
Delete user satish08
printList: (File: grade.sh. No entries.)

Thanks for playing.

AND

./prog3 acl1.txt commandsDe.txt
acl1.txt was successfully opened.
Parsing access control entries from file.
Creating a new access control list for file:
main.c.
printList: (File: main.c. , jwhite04 (rw),
ramon13 (x), crenshaw (orwx), vegdahl (r))

commandsDe.txt was successfully opened.
Parsing commands from file.
Delete user vegdahl
Delete user crenshaw
Delete user jwhite04
Delete user satish08
printList: (File: main.c. , ramon13 (x))

Thanks for playing.
```

Test 5: Does the program execute without segmentation fault on an empty Access Control List? (10 points)

```
$ ./prog3 acl0.txt commands1.txt
acl0.txt was successfully opened.
Parsing access control entries from file.
printList: ( empty access control list )

commands1.txt was successfully opened.
Parsing commands from file.
...
printList: ( empty access control list )

Thanks for playing.
```

Test 6: Does the program execute without segmentation fault on an Access Control List with no entries? (10 points)

```
./prog3 acl3.txt commands1.txt
acl3.txt was successfully opened.
Parsing access control entries from file.
Creating a new access control list for file:
main.c.
printList: (File: main.c.   No entries.)

commands1.txt was successfully opened.
Parsing commands from file.
...
printList: (File: main.c.   No entries.)

Thanks for playing.
```

To receive full points, you must utilize good programming practice (30 points):

• Variables must have meaningful names and global variables must not be used.	2
• Preprocessor directives must be used for constant values.	2
• Code must be documented with useful comments and should use standard tabbing rules for good readability.	9
• Code should not be redundant. If two snippets of code have similar functionality, make a function or write a loop.	6
• A makefile must be used to compile the program and should be submitted with your homework submission.	2
• All files opened by the program and all memory allocated to the program should be closed and freed before program exit.	5
• Only the main function and printX functions should use printf(); other functions should not. Instead, the main() function should print a message depending on the return value of a function.	4
Total	30

Students are given an opportunity for extra credit for implementing this additional feature (10 points)

Alter the addEntry() function in aclist.c to insert AccessControlEntries with the "own" right at the front of the list, and all other entries at the front of the non-owners.

To achieve maximum points on your submission, consider using this submission checklist before submitting your program to the Moodle course website:

- ☐ "I did not change the name of the source files in the starter code. The names of my program sources are main.c, aclist.c and aclistt.h."
- ☐ **"**I submitted a makefile."
- ☐ "My program compiles successfully with the makefile I submitted."
- ☐ "I ran all the tests (see above) to make sure my program executes correctly."
- ☐ "I followed the five pieces of guidance on commenting programs."
- ☐ "I compressed my source *files* into a zipfile named with my username, e.g., crenshaw13.zip"
- ☐ "I did **not** compress my source files using .rar, .z7, or some other proprietary compression program."
- ☐ "I did **not** compress a DIRECTORY of files."
- ☐ "I uploaded my zipfile to Moodle."