**Program Functionality**

The program allows a user to backup files to an AWS S3 bucket. The main function of the program is to backup all of the files in the current directory, as well as the files in any subdirectories. The files maintain their directory structure in the S3 bucket.

The program will only upload files to the S3 bucket if the file has been changed since it was last uploaded.

If the user does not wish to backup all of the files in the current directory and subdirectory, the user can specify certain files to backup (limited to the current directory and subdirectories).

The program allows the user to specify a folder name for the files to be stored in the S3 bucket. This allows the user to implement a form of versioning for the directory files.

**Program Design**

The program prompts the user for a S3 bucket name. The program checks for the bucket, if no bucket exists (or if the name is invalid/unavailable) the program prompts the user for another bucket name. If the requested bucket is found or created, the user prompted to enter individual file names.

The user is prompted to enter a S3 folder name for the files to be stored. If no folder is specified, the current directory is used to store the files in the S3 bucket.

If the user chooses to enter specific files names to backup, the files are compared to the files in the local system to see if they exists. If the user doesn’t choose to enter specific file names, then the program proceeds to backup the entire current directory and subdirectories.

For each file to backup to the bucket, the program checks if the files exist in the S3 bucket, in the S3 folder specified to store the files. If the files exists in the bucket folder, the last modified time of the AWS file is compared to the last modified time of the local file. If the file was modified since previously uploading to the bucket, the program will upload the file. If it hasn’t been modified, the file will not upload.

Whenever the program connects to an S3 bucket, the code to connect is wrapped in a for loop that allows for a failed connection to be retried repeatedly, with a greater time lapse between each attempt. The maximum number of attempts varies depending of the action the program is taking. Some actions will only make 3 failed attempts before giving up, whereas other actions have 9 attempts. The reason why some actions (such as finding an existing S3 bucket) have such few attempts is because a response that is returned as invalid may be so only because the value could not be found in the system. Since this is most likely the reason for some actions, it seems a better design decision to only make a couple of attempts, instead of making the user wait for the computer to make repeated attempts to find something in AWS that doesn’t exist.

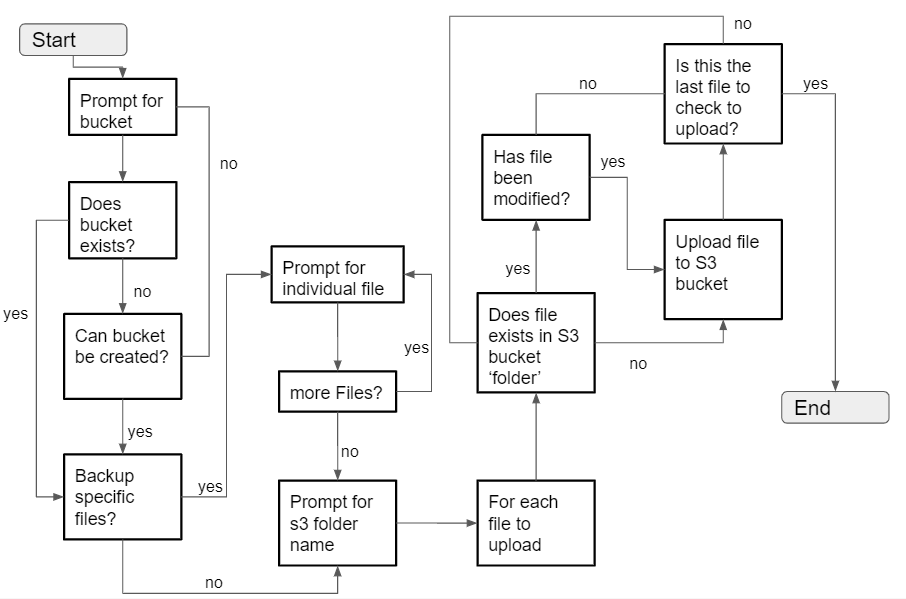
**Program Testing**

The program was tested in numerous ways, with the goal to check any event, or combination of events, that might cause a failure.

* Bucket names were tested, both existing, not existing, and buckets with illegitimate names (names probably in use by someone else or not acceptable by AWS).
* The program was tested to work in different directories.
* The program was tested to upload different file types.
* The program was tested by uploading files that had not been modified to test that the files would not be uploaded. Testing was also done to modify certain files to ensure that those files were uploaded.
* For specific file names to upload, legitimate and illegitimate file names were used. Testing was done to ensure that the program would check to make sure that a requested file existed in the local file system before attempting to upload the file.

Various combinations of the above tests were conducted to check for invalid program behavior.

**Program Model**

****