File Packager: Simple Tar

By: Sebastián Bermúdez Acuña and Felipe Obando Arrieta

October 29th, 2023

Table of Contents

1. Introduction

The present project, "Simple Tar" (star) is an implementation of a file packer that seeks to replicate the essential functionality of the well-known *tar* command in UNIX environments. This program has been developed with the purpose of strengthening the knowledge and experience in the management of directories, files and contiguous memory, fundamental components in the operation of operating systems. "Simple Tar" allows users to create a single archive that stores multiple individual files, facilitating the organization and transfer of data. In addition, it offers the basic operations of creating, extracting, listing, deleting, updating, aggregating, and defragmenting, following principles of efficiency and optimal space utilization.

Throughout this document, the internal structure of the program will be explored in detail, describing the strategies used to manage packed files and their data, free space control, file access logic and defragmentation techniques, among other key aspects. In addition, the results of tests performed to verify its performance in various scenarios will be presented.

2. Problem description

The project focuses on the development of a file packager that mimics the essential functionality of the *tar* command in Unix environments. The main purpose is to allow users to store multiple files in a single packaged archive, thus facilitating data organization, distribution, and backup. The basic operations that "Simple Tar" implement include creating, extracting, listing, deleting, updating, aggregating, and defragmenting files in the packaged archive.

This project addresses several fundamental aspects of operating system programming and file management, including the need to manage internal directory structures within the packaged archive and to keep track of free space within the archive. Defragmentation is an additional feature that is proposed to optimize storage space in the packed archive. The program takes care of detailed display of its operation while running, as well as information about the files contained in the *tar* archive. "Simple Tar" addresses the creation of a versatile and efficient tool that simplifies the management of archives.

3. Data structures definition

In the "Simple Tar" project, several key data structures are used to make it possible to manage the stored files and free spaces within the *tar* file.

3.1. Header

The header is a critical component in the organization of packaged files in the "Simple Tar" format. Each file contained in the packaged archive is characterized and recorded by a set of attributes stored in this structure. The key elements of the header, for each file, include the file name; the file mode, indicating the file's permissions and attributes; the size of the file contents (in bytes); and two other attributes to indicate the position in bytes within the packed file where the corresponding file begins and ends; in addition, there is a flag that takes the value 0 to denote that the file has not been marked for deletion and 1 to denote that the file has been deleted.

```
struct File {
    char fileName[MAX_FILENAME_LENGTH];
    mode_t mode;
    off_t size; // 0: No file, > 0: Yes file
    off_t start;
    off_t end;
    int deleted;// 0: No, 1: Yes
};
struct Header{
    struct File fileList[MAX_FILES];
} header; // declaration of header
```

3.2. Blank Space

It is a simple linked list that accomplishes efficient space management within the packed file. Each node in this list stores information about the free space available within the packed file, identifying both the start and end of each blank space, as well as an index for reference.

Blank Space is used to keep track of unused blocks of space in the packed file, allowing them to be reused in case new files are added. When a file is removed from the file packager, the space it occupied is marked as free and recorded in this list.

```
struct BlankSpace{
    off_t start;
    off_t end;
    int index;
    struct BlankSpace * nextBlankSpace;
} * firstBlankSpace; // declaration of blank spaces
```

4. Main program components

This section describes the main components of the program to accomplish the operation of "Simple Tar".

4.1. File access mechanism

The program accesses the files to be packed or unpacked using UNIX system calls for reading and writing files. For this a function was developed that has the responsibility to open a file with some specific configuration in the file parameters, this in order to be able to perform different operations on the file. The difference in the use of different options depends on the use of the file to be accessed.

```
int openFile(const char * fileName,int option){
    int fd;
    if (option==0){ // Reading and writing without deleting content
        fd = open(fileName, O_RDWR  | O_CREAT, 0666);
        if (fd == -1) {
            perror("Error creating the packed file");
            exit(1);
        }
        //printf("File in option: 0 -> O_RDWR  | O_CREAT\n");
        return fd;
    }else if (option==1){ // Create empty file
        fd = open(fileName, O_WRONLY | O_CREAT | O_TRUNC, 0666);
        if (fd == -1) {
            perror("Error creating the packed file");
            exit(1);
        }
        //printf("File in option: 1 -> O_WRONLY | O_CREAT | O_TRUNC\n");
        return fd;
    }else if (option==2){ //Extend the file (for the body)
        fd = open(fileName, O_WRONLY | O_APPEND, 0666);
        if (fd == -1) {
            perror("Error creating the packed file");
            exit(1);
        }
        //printf("File in option: 2 -> O_WRONLY | O_APPEND\n");
        return fd;
    }else{
        printf("File creation/opening option wrong...\n");
        return -1;
        }
}
```

4.2. Structure of internal directories

The packed files are organized internally in the *tar* file, and a record of the location and size of each file is kept by the header. To better understand how the program works, a representation of how the packer file is being handled can be drawn. Consider the red blocks as files (content).

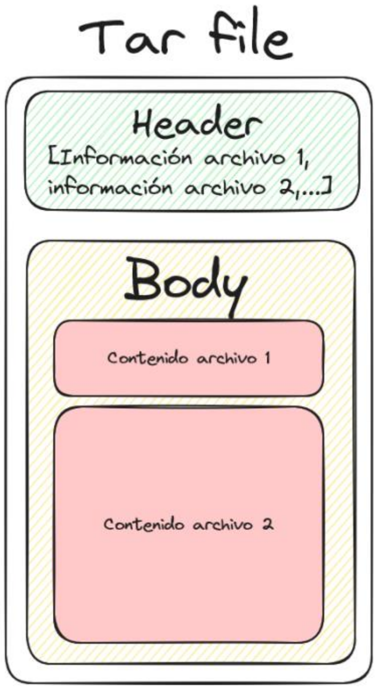


Figure #1: Structure representation for the management of the file packager

In case there are any blank spaces, the header stores information about the blanks in conjunction with the BlankSpace structure. Consider the green block as a blank space.

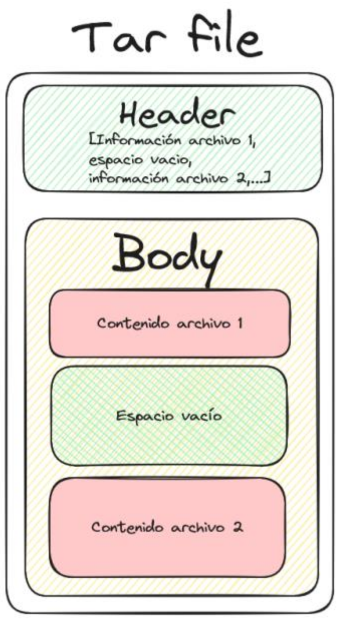


Figure #2: Blank spaces representation in the file packager

To efficiently manage the header, there are some key functions, which are presented below.

4.2.1 *Function: writeHeaderToTar*

Writes the contents of the header to the header of the file packager.

```
void writeHeaderToTar(int tarFile){
    printf("\nWriting header to tar...\n");
    char headerBlock[sizeof(header)];
    memset(headerBlock, 0, sizeof(headerBlock));
    memcpy(headerBlock, &header, sizeof(header));
    if(write(tarFile,headerBlock,sizeof(headerBlock))==-1){
        perror("writeHeaderToTar: Error writing header in tar file.");
        exit(1);
    }
}
```

4.2.2 *Function: readHeaderFromTar*

Reads the contents of the header stored in the tar file, i.e. the contents stored in secondary memory is loaded into main memory.

```
int readHeaderFromTar(int tarFile){
    if (lseek(tarFile, 0, SEEK_SET) == -1) {
        perror("readHeaderFromTar: Pointer error");
        close(tarFile);
        exit(1);
    }
    char headerBlock[sizeof(header)];
    ssize_t bytesRead=read(tarFile, headerBlock, sizeof(headerBlock));
    if (bytesRead < 0) {
        perror("readHeaderFromTar: Error Reading header.");
        exit(1);
    } else if (bytesRead < sizeof(header)) {
        fprintf(stderr, "readHeaderFromTar: All content not read.\n");
        exit(1);
    }
    memcpy(&header, headerBlock, sizeof(header))
    return 1;
}
```

4.2.3 Function: addFileToHeaderFileList

Adds a new item to the list of files in the header.

```
void addFileToHeaderFileList(struct File newFile) {
    printf("Adding \"%s\" to header's file list.\n", newFile.fileName);
    for (int i = 0; i < MAX_FILES; i++) {
        if (header.fileList[i].size==0){ // Empty position
            header.fileList[i]= newFile;
            numFiles++;
            break;
        }
    }
}
```

4.3. Blank spaces management

A first-fit strategy is used to manage free spaces within the packer file. The available blank spaces are tracked using a linked list, in which the available blanks are added between the files in the *tar*. Each item in the list stores the start and end location of the blank space, as well as the index to be used in the list of files in the header structure.

Some of the most important functions for handling blank spaces are presented below.

4.3.1. Function: addBlankSpace

Adds a blank space to the list.

```
int addBlankSpace(off_t start, off_t end, int index){
    if (isBlankSpaceRepeated(start,end)==1){
            return 0;
        }
        if (start > end) {
            fprintf(stderr, "addBlankSpace: Error start>end.\n");
            exit(1);
        }
        if (start==0 && end==0)
            return 0;

        // Create a new node
        Struct      BlankSpace      *newBlankSpace      =      (struct
BlankSpace*)malloc(sizeof(struct BlankSpace));
        if (newBlankSpace == NULL) {
            fprintf(stderr, "addBlankSpace: Error Malloc.\n");
            exit(1);
        }
        // Initialize new node of blank space;
        newBlankSpace->start = start;
        newBlankSpace->end = end;
        newBlankSpace->index = index;
        newBlankSpace->nextBlankSpace = NULL;

        // Empty list or new node goes before the first one: Index lower.
        if  (firstBlankSpace  ==  NULL  ||  newBlankSpace->index  <
firstBlankSpace->index) {
            newBlankSpace->nextBlankSpace = firstBlankSpace;
            firstBlankSpace = newBlankSpace;
        } else { // List with elements
            struct BlankSpace * current = firstBlankSpace;
            while ((current->nextBlankSpace != NULL) &&
            (current->nextBlankSpace->index <= newBlankSpace->index)) {
                current = current->nextBlankSpace;
            }
            newBlankSpace->nextBlankSpace = current->nextBlankSpace;
            current->nextBlankSpace = newBlankSpace;
        }
        return 1;
    }
```

4.3.2. Function: deleteBlankSpace

Deletes a blank space from the list.

```
    void deleteBlankSpace(int targetIndex) {
        struct BlankSpace * current = firstBlankSpace;
        struct BlankSpace * prev = NULL;

        // Search for the element that matches the target index
        while (current != NULL && current->index != targetIndex) {
            prev = current; // Leaves pointer in the previous element.
            current = current->nextBlankSpace; // Advances
        }

        if (current == NULL) { // Element not found
            printf("deleteBlankSpace: Element not found");
            return;
        }
        // Adjuste pointers to delete the element
        if (prev == NULL){
            firstBlankSpace = current->nextBlankSpace;
        }else{ // Not the first element.
            prev->nextBlankSpace = current->nextBlankSpace;
        }
        // Free memory
        free(current);
    }
```

4.4. File Defragmentation

To accomplish the defragmentation of the file packager, several steps are performed.

- **Recap of existing content**: To accomplish the archive defragmentation, first all existing content of the files stored in the tar archive is collected. The content is stored in a buffer that contains all data sequentially.
- **Identification of existing files and adjustment of position bytes**: After the content collection, the existing files are identified and the position bytes of these files are adjusted to accommodate the new structure of the file packager, so that the existing contents are stored sequentially.
- **Resetting the header**: The header is restarted so that it forgets the information it contains.
- **Adding files to the header**: Now, a loop is performed to add each file identified in the previous step to the header using the *addFileToHeaderFileList* function.

- **Updating the header in the tar file**: When all the previous procedure is completed, the information is inside the header, as desired in the file packager. The *tar* file is opened another time to allow again the writing of the updated header with the function *writeHeaderToTar*.
- **Truncation of the *tar* file**: Finally, the packer file is truncated. This shortens the archive to a size that includes only the header.
- **Writing the contents**: Then, the buffer with the contents of all stored files is written to the same packer file.

The function in charge of the defragmentation is *pack*:

```
void pack(const char * tarFileName){
    int tarFile = openFile(tarFileName,0);
    if (readHeaderFromTar(tarFile)!=1){ // Read header from tar
        printf("extractAll: Error reading the header of the tar file.\n");
        close(tarFile);
        exit(10);
    }
    close (tarFile);
    calculateBlankSpaces(tarFileName); // Calculate blank spaces

    printf("PACK\n");
    int sumFiles = 0;
    char*   bodyContentBuffer   =   getWholeBodyContentInfo(tarFileName,
&sumFiles);
    struct File * modifiedFiles = modifiedExistentFiles(sumFiles);
    resetHeader();

    // Adds the files with the new info in the header
    for (int j=0; j<sumFiles; j++){
        addFileToHeaderFileList(modifiedFiles[j]);
    }
    tarFile = openFile(tarFileName,0);
    writeHeaderToTar(tarFile); // Re-write header in tar file
    close(tarFile);

    truncateFile(tarFileName, sizeof(header));
    tarFile = openFile(tarFileName, 0);
    if (lseek(tarFile, 0, SEEK_END) == -1) {
        perror("pack: Error moving the pointer.");
        close(tarFile);
        exit(1);
    }
    // Write the content stored in bodyContentBuffer
    if (write(tarFile,bodyContentBuffer,strlen(bodyContentBuffer))==-1) {
        perror("pack: Error writing the content in the tar file.");
        close(tarFile);
        exit(1);
    }
    close(tarFile);
    resetBlankSpaceList(); // Reset blank spaces list
    printHeader();
    printBlankSpaces();
}
```

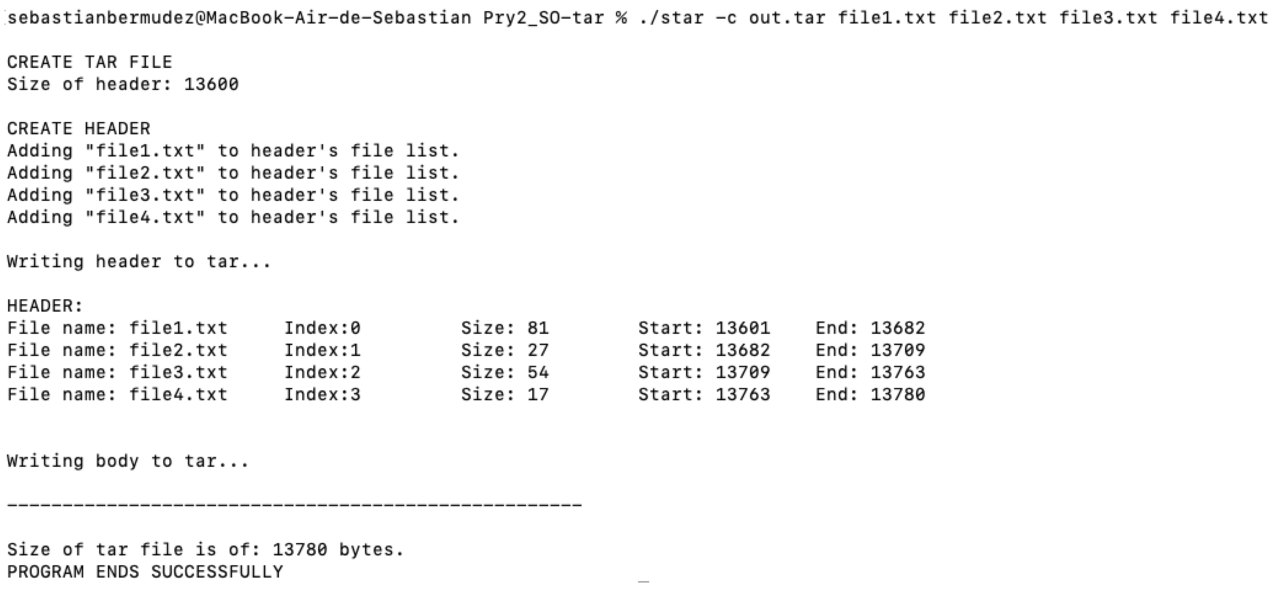
5. Performance tests

With the application of the performance tests for the "Simple Tar" program, the aim is to ensure and check its correct functioning in various scenarios and operations for all its functionalities.

5.1. Option: *create* (-c)

Has the function to create a new packed file. Allows the user to select a series of individual files and combine them into a single packaged file.

Test #1: create a file *out.tar* with the content of 4 files.

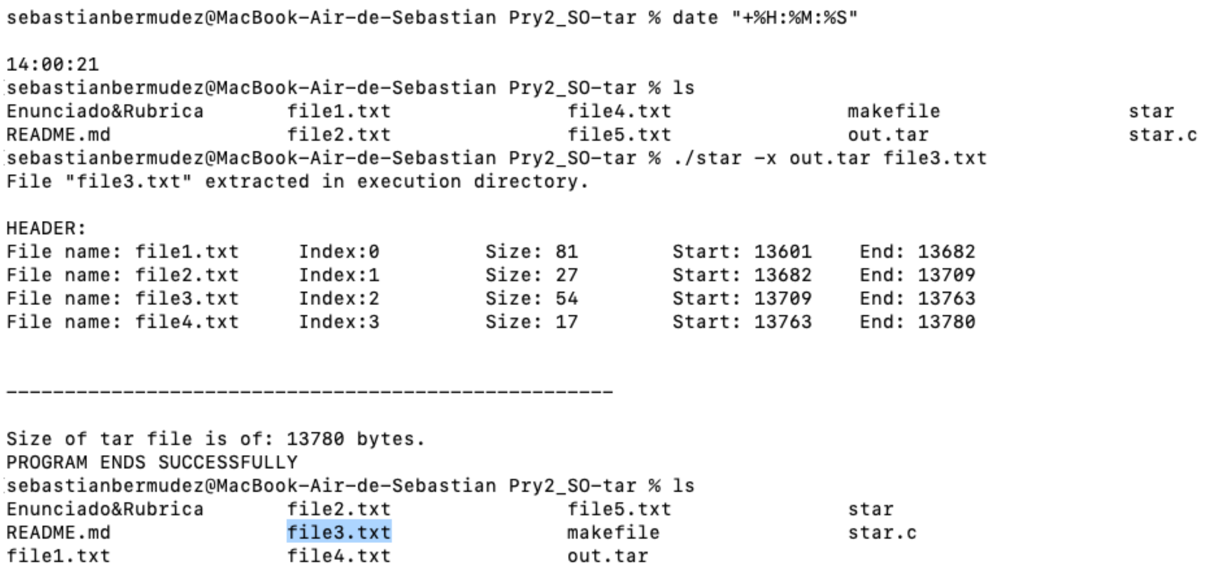


```
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -c out.tar file1.txt file2.txt file3.txt file4.txt

CREATE TAR FILE
Size of header: 13600

CREATE HEADER
Adding "file1.txt" to header's file list.
Adding "file2.txt" to header's file list.
Adding "file3.txt" to header's file list.
Adding "file4.txt" to header's file list.

Writing header to tar...

HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file2.txt     Index:1         Size: 27        Start: 13682    End: 13709
File name: file3.txt     Index:2         Size: 54        Start: 13709    End: 13763
File name: file4.txt     Index:3         Size: 17        Start: 13763    End: 13780


Writing body to tar...

----------------------------------------------------

Size of tar file is of: 13780 bytes.
PROGRAM ENDS SUCCESSFULLY
```

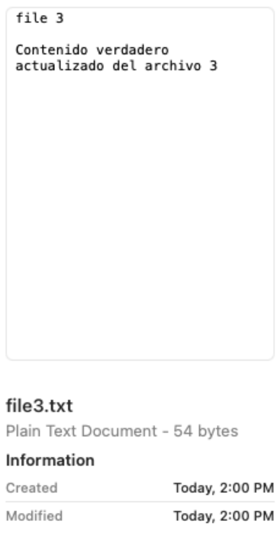
5.2. Option: *extract* (-x)

Used to extract several or all files contained in the packed file. It allows the user to retrieve previously packed individual files, restoring them to the current directory. When extracted, it stores the original content that was kept in the packer file.

Test #1: extracts the file *file3.txt* from the packed file. Note that *file3.txt* did not exist in the directory, so it was the program that created it.



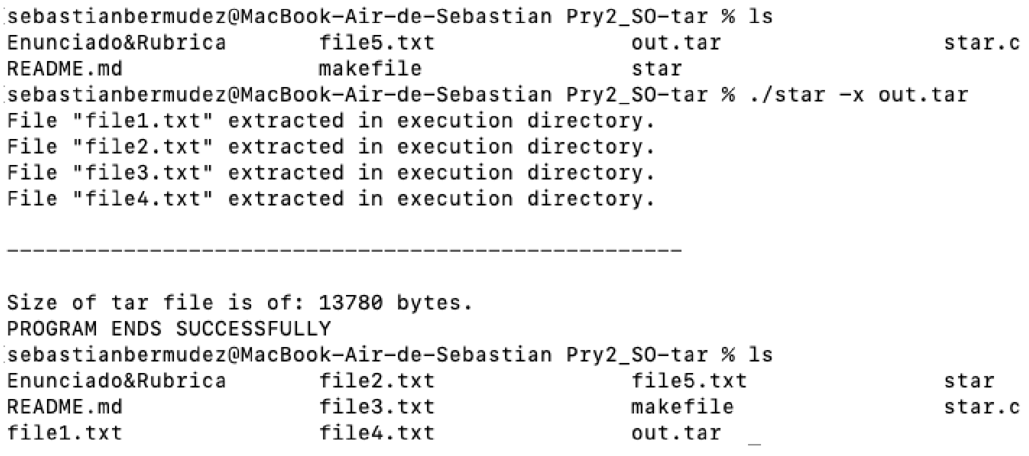
```
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % date "+%H:%M:%S"

14:00:21
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ls
Enunciado&Rubrica       file1.txt               file4.txt               makefile                star
README.md               file2.txt               file5.txt               out.tar                 star.c
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -x out.tar file3.txt
File "file3.txt" extracted in execution directory.

HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file2.txt     Index:1         Size: 27        Start: 13682    End: 13709
File name: file3.txt     Index:2         Size: 54        Start: 13709    End: 13763
File name: file4.txt     Index:3         Size: 17        Start: 13763    End: 13780


----------------------------------------------------

Size of tar file is of: 13780 bytes.
PROGRAM ENDS SUCCESSFULLY
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ls
Enunciado&Rubrica       file2.txt               file5.txt               star
README.md               file3.txt               makefile                star.c
file1.txt               file4.txt               out.tar
```



```
file 3

Contenido verdadero
actualizado del archivo 3
```

file3.txt
Plain Text Document - 54 bytes
Information
Created Today, 2:00 PM
Modified Today, 2:00 PM

Test #2: extraction of all files contained in the packed archive.

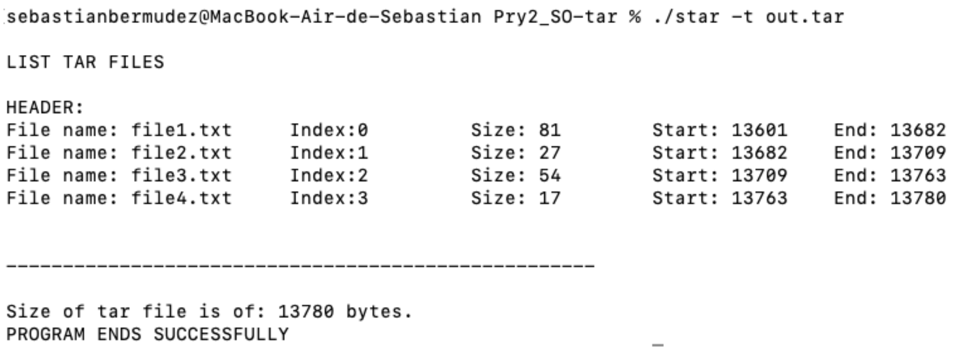


```
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ls
Enunciado&Rubrica       file5.txt               out.tar                 star.c
README.md               makefile                star
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -x out.tar
File "file1.txt" extracted in execution directory.
File "file2.txt" extracted in execution directory.
File "file3.txt" extracted in execution directory.
File "file4.txt" extracted in execution directory.

-----------------------------------------------------

Size of tar file is of: 13780 bytes.
PROGRAM ENDS SUCCESSFULLY
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ls
Enunciado&Rubrica       file2.txt               file5.txt               star
README.md               file3.txt               makefile                star.c
file1.txt               file4.txt               out.tar
```

5.3. Option: *list* (-t)

Allows the user to list the contents of a packed file. Provides a detailed view of the packed files, including their names, sizes and attributes, without extracting the files themselves.

Test #1: displays the information of the files inside the packed file.

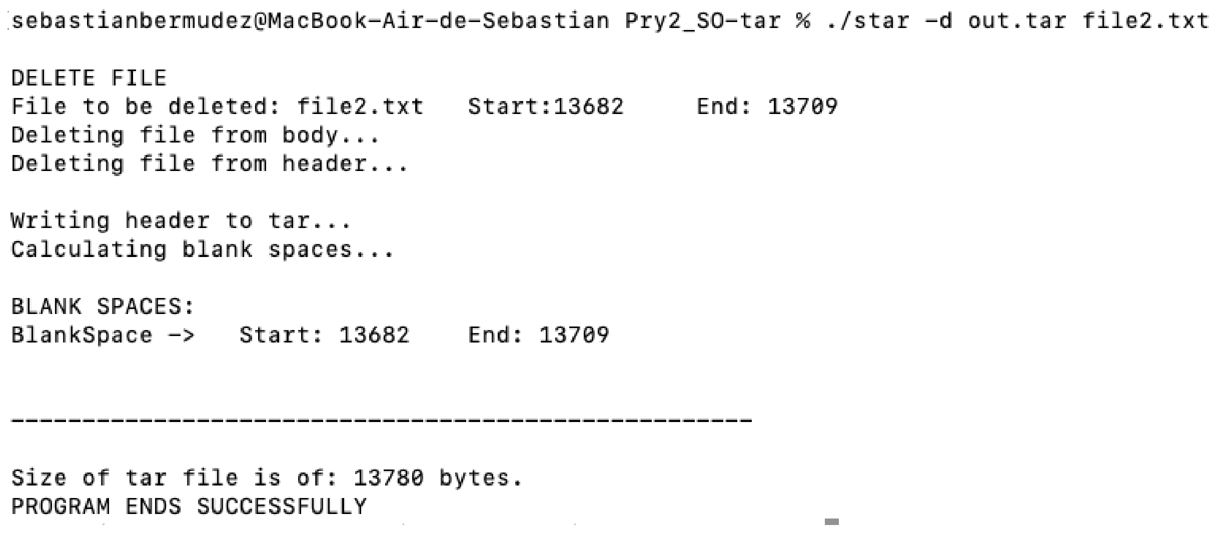


```
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -t out.tar

LIST TAR FILES

HEADER:
File name: file1.txt     Index:0        Size: 81       Start: 13601    End: 13682
File name: file2.txt     Index:1        Size: 27       Start: 13682    End: 13709
File name: file3.txt     Index:2        Size: 54       Start: 13709    End: 13763
File name: file4.txt     Index:3        Size: 17       Start: 13763    End: 13780


----------------------------------------------------

Size of tar file is of: 13780 bytes.
PROGRAM ENDS SUCCESSFULLY
```

5.4. Option: *delete* (-d)

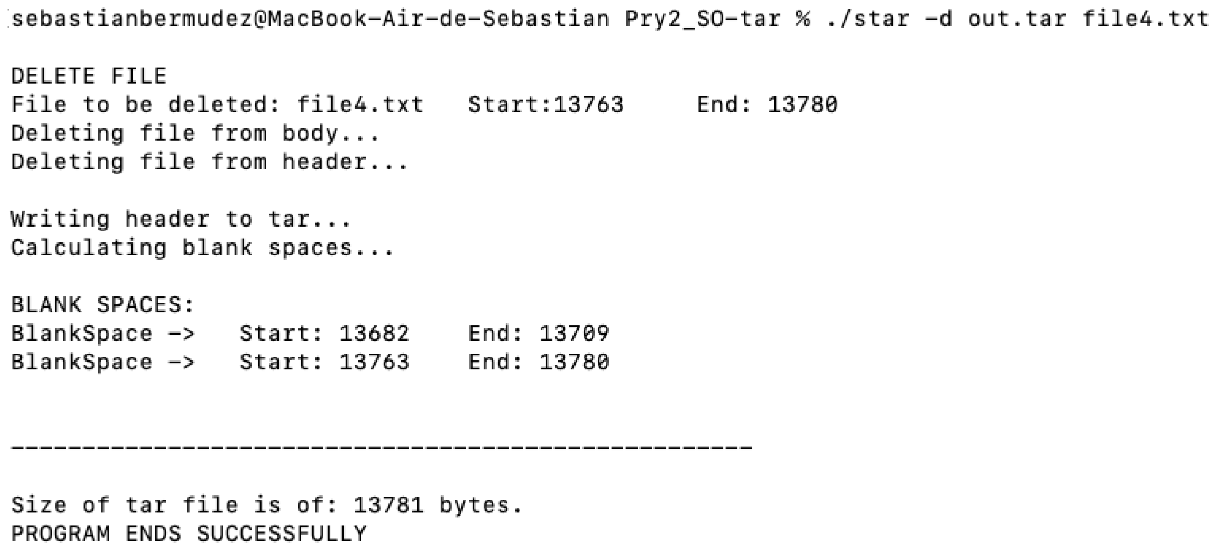
Used to remove specific files from the packed archive. Allows the user to select some file and mark it for deletion.

Test #1: deletion of *file2.txt*.



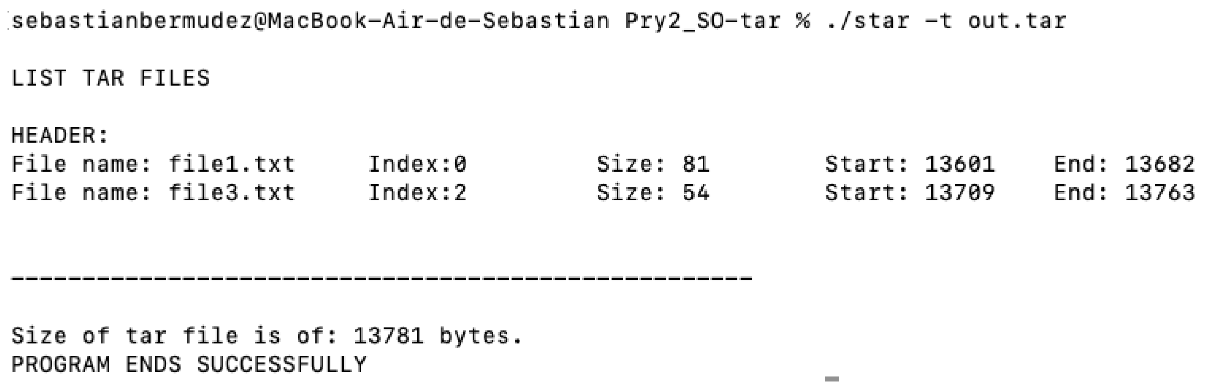
```
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -d out.tar file2.txt

DELETE FILE
File to be deleted: file2.txt   Start:13682     End: 13709
Deleting file from body...
Deleting file from header...

Writing header to tar...
Calculating blank spaces...

BLANK SPACES:
BlankSpace ->   Start: 13682    End: 13709


----------------------------------------------------

Size of tar file is of: 13780 bytes.
PROGRAM ENDS SUCCESSFULLY
```

Test #2: deletion of *file4.txt*.



```
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -d out.tar file4.txt

DELETE FILE
File to be deleted: file4.txt   Start:13763     End: 13780
Deleting file from body...
Deleting file from header...

Writing header to tar...
Calculating blank spaces...

BLANK SPACES:
BlankSpace ->   Start: 13682    End: 13709
BlankSpace ->   Start: 13763    End: 13780


----------------------------------------------------

Size of tar file is of: 13781 bytes.
PROGRAM ENDS SUCCESSFULLY
```

Note that, in both cases, blank spaces are added for the free space left by the files. After deletion, the packer file has changed its storage information.

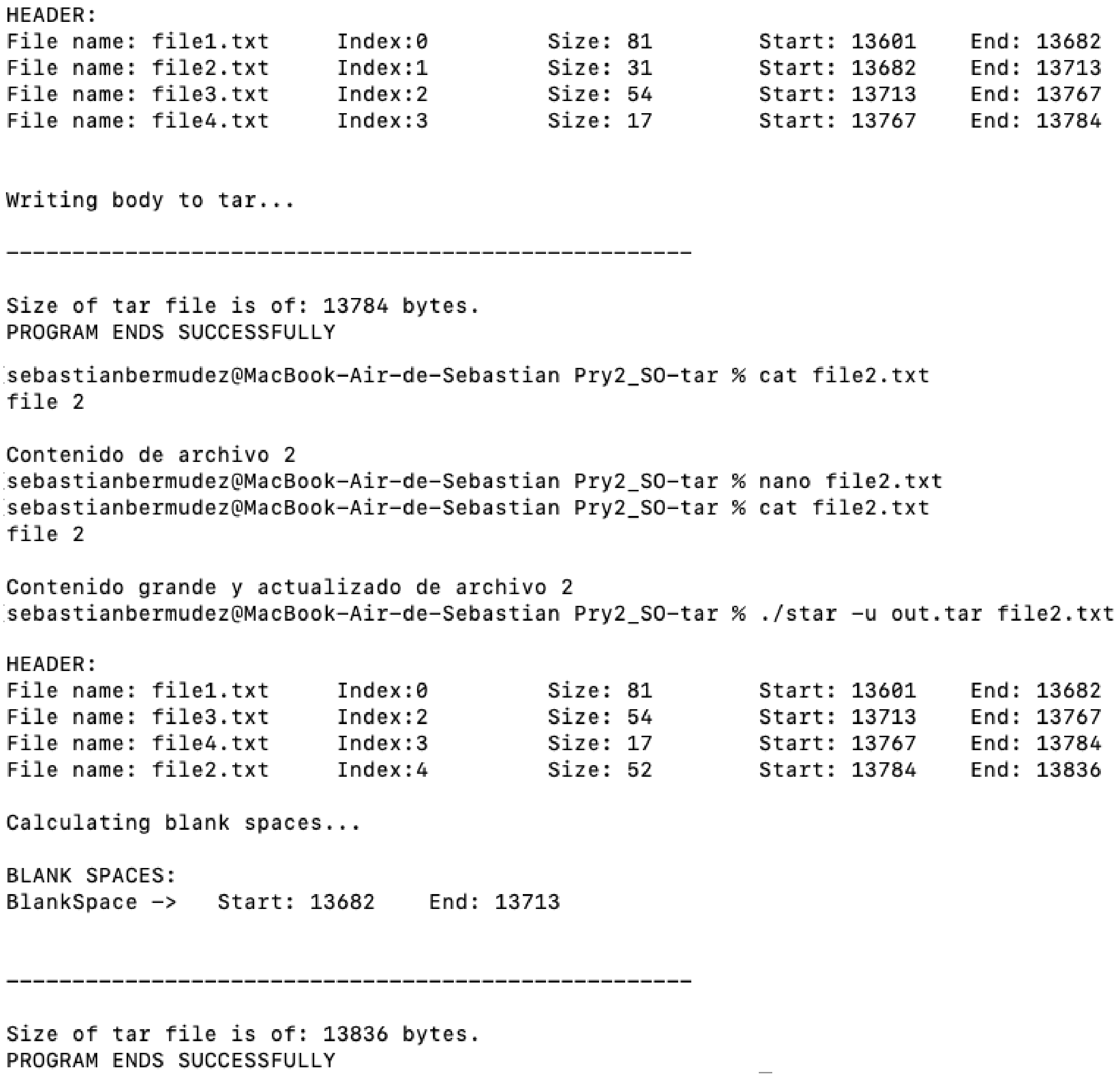


```
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -t out.tar

LIST TAR FILES

HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file3.txt     Index:2         Size: 54        Start: 13709    End: 13763


----------------------------------------------------

Size of tar file is of: 13781 bytes.
PROGRAM ENDS SUCCESSFULLY
```

5.5. Option: *update* (-u)

Allows the user to update the contents of a packed file. If the content of a file is modified, this option updates the content.

Test #1: updating *file2.txt*. Modifies *file2.txt* and make its content bigger.

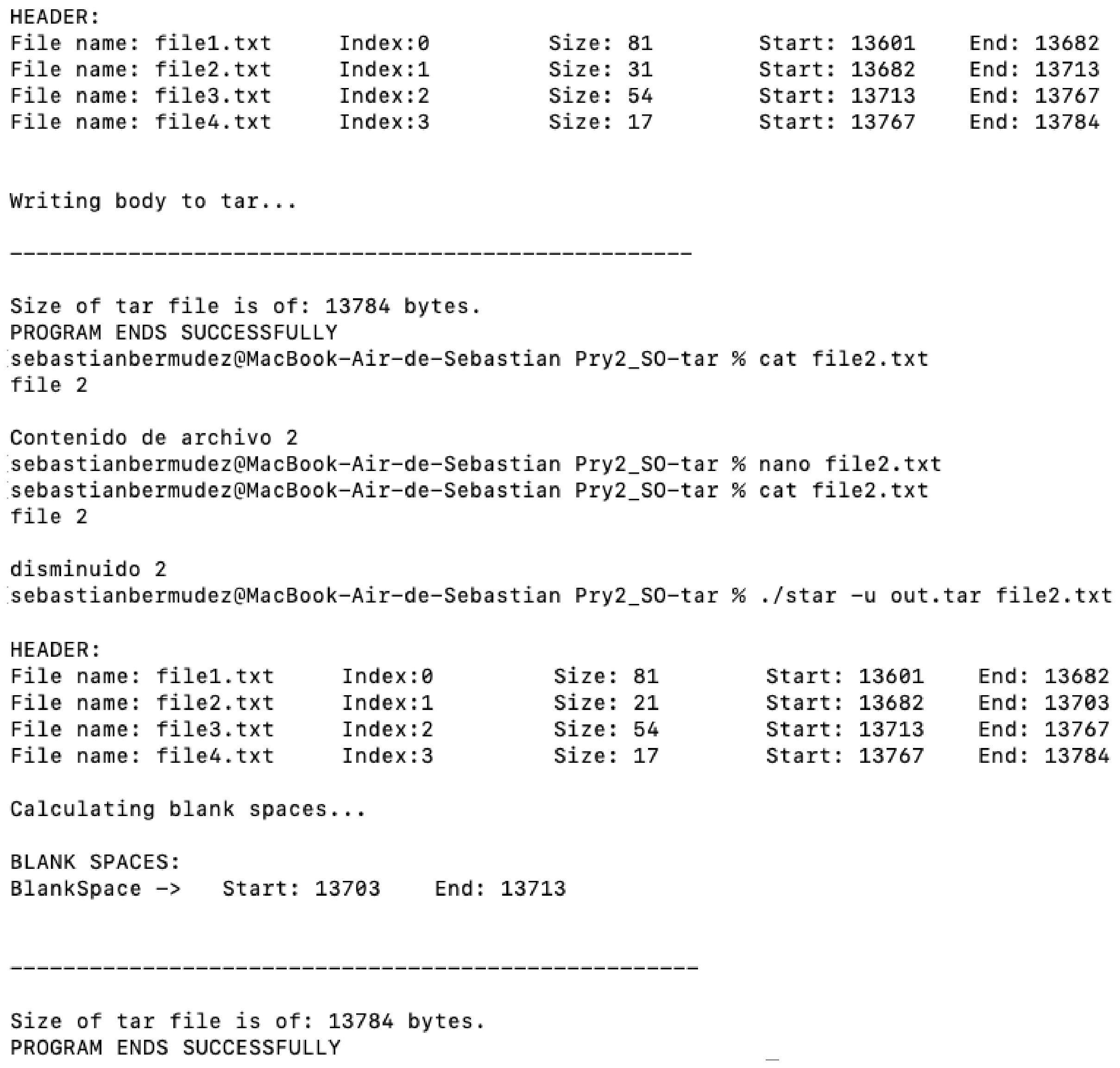


```
HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file2.txt     Index:1         Size: 31        Start: 13682    End: 13713
File name: file3.txt     Index:2         Size: 54        Start: 13713    End: 13767
File name: file4.txt     Index:3         Size: 17        Start: 13767    End: 13784


Writing body to tar...

----------------------------------------------------

Size of tar file is of: 13784 bytes.
PROGRAM ENDS SUCCESSFULLY

sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % cat file2.txt
file 2

Contenido de archivo 2
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % nano file2.txt
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % cat file2.txt
file 2

Contenido grande y actualizado de archivo 2
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -u out.tar file2.txt

HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file3.txt     Index:2         Size: 54        Start: 13713    End: 13767
File name: file4.txt     Index:3         Size: 17        Start: 13767    End: 13784
File name: file2.txt     Index:4         Size: 52        Start: 13784    End: 13836

Calculating blank spaces...

BLANK SPACES:
BlankSpace ->   Start: 13682    End: 13713


----------------------------------------------------

Size of tar file is of: 13836 bytes.
PROGRAM ENDS SUCCESSFULLY
```

After this execution it is important to highlight several aspects that had effect on the program:

- *file2.txt* changes its size from 31 bytes to 52 bytes.
- As its size is bigger and does not fit where it was, its location is moved to the end of the file.
- The size of the *tar* file increased due to the previous point.
- A blank space is added for the free field left by moving to *file2.txt* at the end of the packed file.

Test #2: updating *file2.txt*. Modifies *file2.txt* and make its content smaller.



```
HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file2.txt     Index:1         Size: 31        Start: 13682    End: 13713
File name: file3.txt     Index:2         Size: 54        Start: 13713    End: 13767
File name: file4.txt     Index:3         Size: 17        Start: 13767    End: 13784


Writing body to tar...

----------------------------------------------------

Size of tar file is of: 13784 bytes.
PROGRAM ENDS SUCCESSFULLY
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % cat file2.txt
file 2

Contenido de archivo 2
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % nano file2.txt
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % cat file2.txt
file 2

disminuido 2
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -u out.tar file2.txt

HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file2.txt     Index:1         Size: 21        Start: 13682    End: 13703
File name: file3.txt     Index:2         Size: 54        Start: 13713    End: 13767
File name: file4.txt     Index:3         Size: 17        Start: 13767    End: 13784

Calculating blank spaces...

BLANK SPACES:
BlankSpace ->   Start: 13703    End: 13713


----------------------------------------------------

Size of tar file is of: 13784 bytes.
PROGRAM ENDS SUCCESSFULLY
```

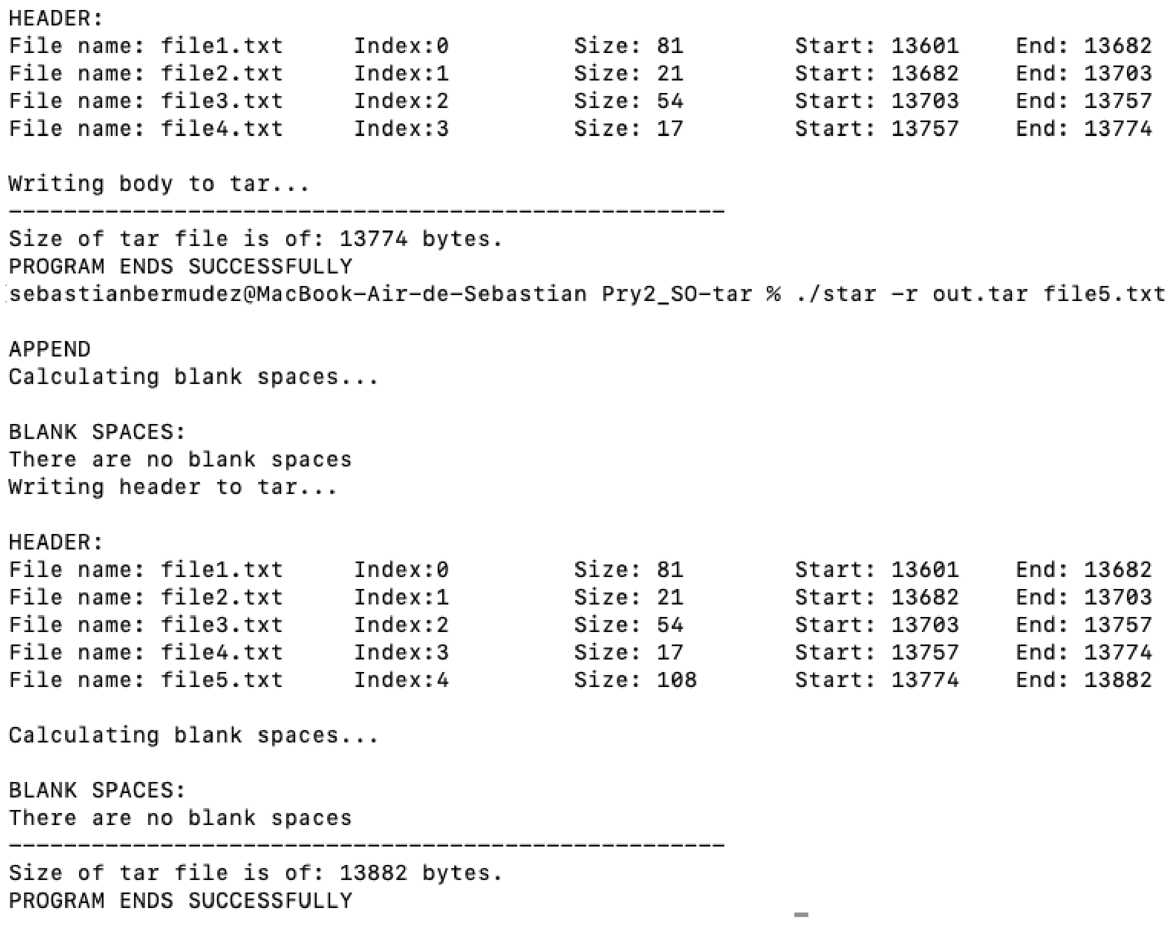
After this execution, it is also important to highlight several aspects that had effect on the program:

- *file2.txt* changes its size, going from 31 bytes to 21 bytes.
- Since the content does fit in the original space, its location does not change.
- The size of the *tar* file was not changed.
- A blank space is added for the free space which meant decreasing the size of the file.

5.6. Option: *append* (-r)

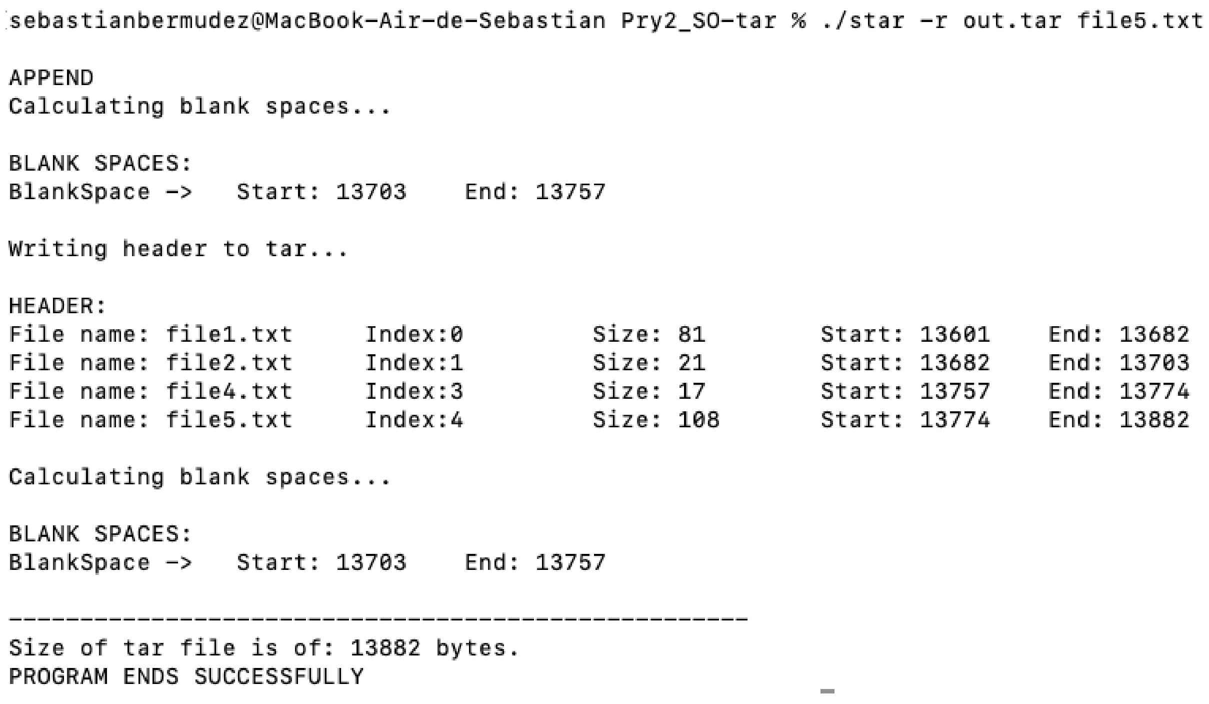
It is used to add additional content to a packed file. The program first checks the available free spaces within the packed file to determine if the new file can fit into any of these spaces. If there is insufficient free space, the new content is added to the end of the packed file, resulting in an increase in the size of the packed file to accommodate the additional content.

Test #1: adds *file5.txt*. There are no blank spaces.



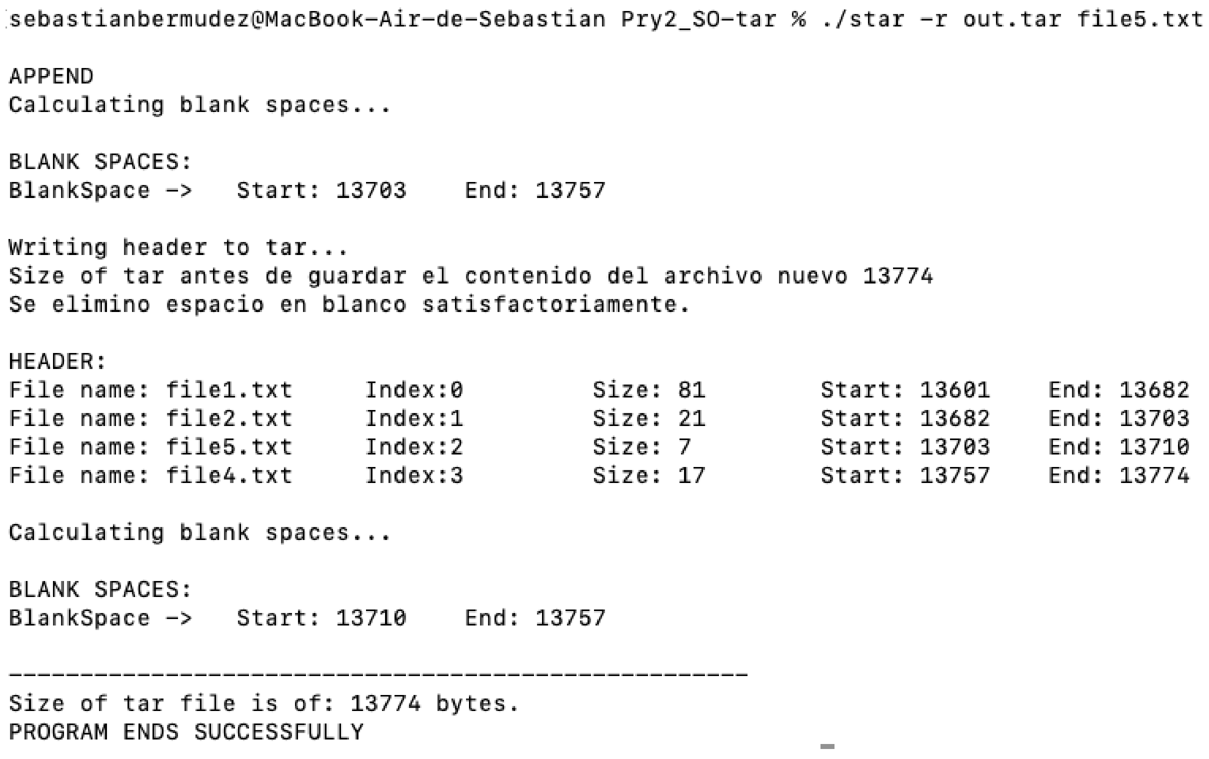
```
HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file2.txt     Index:1         Size: 21        Start: 13682    End: 13703
File name: file3.txt     Index:2         Size: 54        Start: 13703    End: 13757
File name: file4.txt     Index:3         Size: 17        Start: 13757    End: 13774

Writing body to tar...
----------------------------------------------------
Size of tar file is of: 13774 bytes.
PROGRAM ENDS SUCCESSFULLY
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -r out.tar file5.txt

APPEND
Calculating blank spaces...

BLANK SPACES:
There are no blank spaces
Writing header to tar...

HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file2.txt     Index:1         Size: 21        Start: 13682    End: 13703
File name: file3.txt     Index:2         Size: 54        Start: 13703    End: 13757
File name: file4.txt     Index:3         Size: 17        Start: 13757    End: 13774
File name: file5.txt     Index:4         Size: 108       Start: 13774    End: 13882

Calculating blank spaces...

BLANK SPACES:
There are no blank spaces
----------------------------------------------------
Size of tar file is of: 13882 bytes.
PROGRAM ENDS SUCCESSFULLY
```

Test #2: adds *file5.txt*. There are blank spaces, but the new file does not fit on any.



```
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -r out.tar file5.txt

APPEND
Calculating blank spaces...

BLANK SPACES:
BlankSpace ->   Start: 13703    End: 13757

Writing header to tar...

HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file2.txt     Index:1         Size: 21        Start: 13682    End: 13703
File name: file4.txt     Index:3         Size: 17        Start: 13757    End: 13774
File name: file5.txt     Index:4         Size: 108       Start: 13774    End: 13882

Calculating blank spaces...

BLANK SPACES:
BlankSpace ->   Start: 13703    End: 13757

---------------------------------------------------
Size of tar file is of: 13882 bytes.
PROGRAM ENDS SUCCESSFULLY
```

Test #3: adds *file5.txt*. There are blank spaces, and the new file fits in one of these.

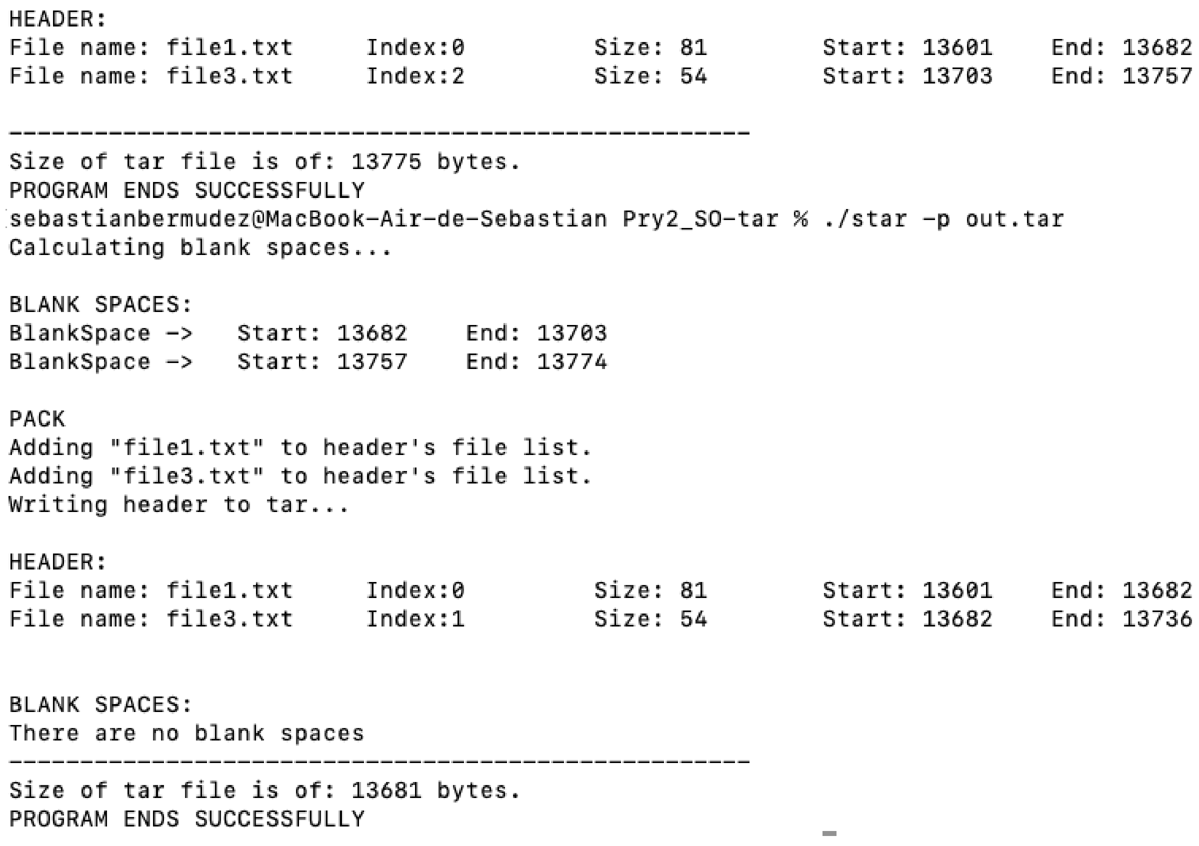


```
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -r out.tar file5.txt

APPEND
Calculating blank spaces...

BLANK SPACES:
BlankSpace ->   Start: 13703    End: 13757

Writing header to tar...
Size of tar antes de guardar el contenido del archivo nuevo 13774
Se elimino espacio en blanco satisfactoriamente.

HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file2.txt     Index:1         Size: 21        Start: 13682    End: 13703
File name: file5.txt     Index:2         Size: 7         Start: 13703    End: 13710
File name: file4.txt     Index:3         Size: 17        Start: 13757    End: 13774

Calculating blank spaces...

BLANK SPACES:
BlankSpace ->   Start: 13710    End: 13757

---------------------------------------------------
Size of tar file is of: 13774 bytes.
PROGRAM ENDS SUCCESSFULLY
```

5.7. Option: *pack* (-p)

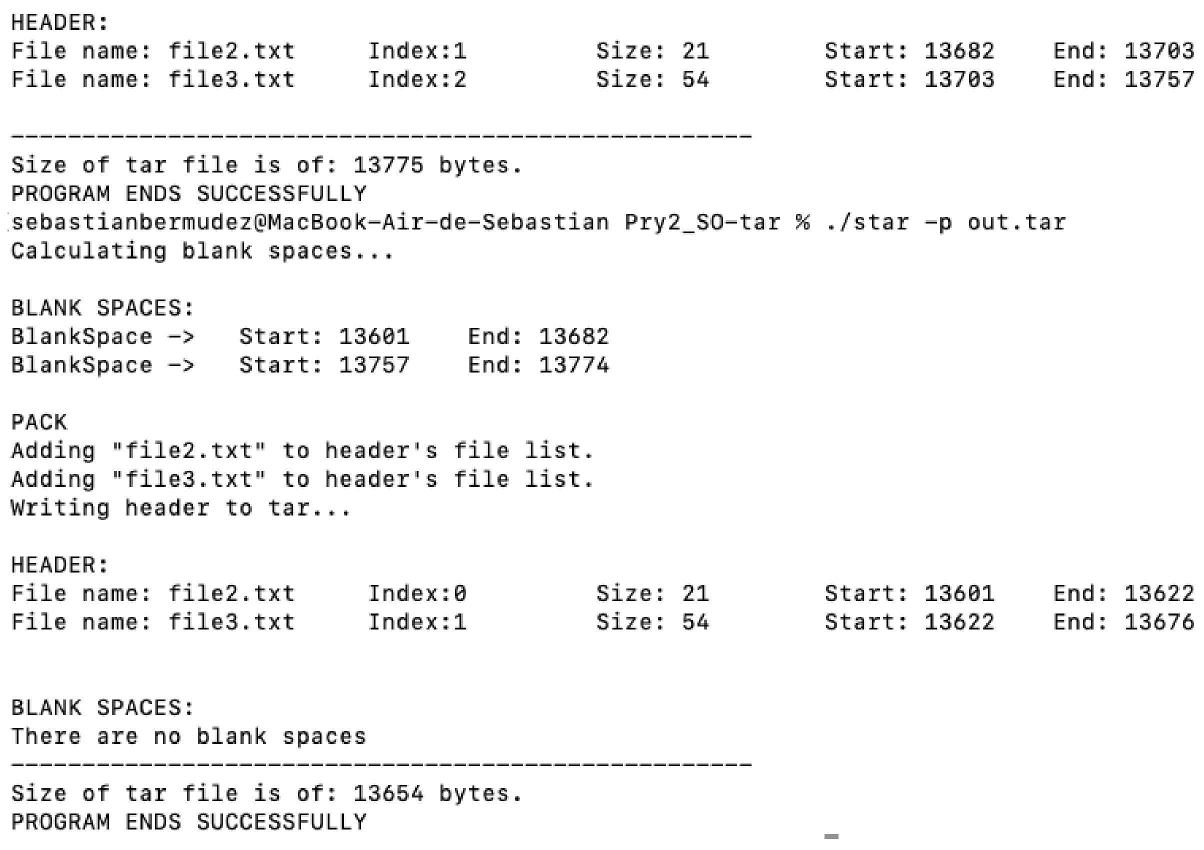
It is used to defragment the contents of the packed file and free up any unused space. This option adjusts the size of the packed file to the actual existing content, eliminating free blocks and optimizing storage space. Its logic is found in the File Defragmentation section of this document.

Test #1: observe the file layout and blank spaces in the packed file prior to execution. The file size decreases.



```
HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file3.txt     Index:2         Size: 54        Start: 13703    End: 13757

----------------------------------------------------
Size of tar file is of: 13775 bytes.
PROGRAM ENDS SUCCESSFULLY
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -p out.tar
Calculating blank spaces...

BLANK SPACES:
BlankSpace ->   Start: 13682    End: 13703
BlankSpace ->   Start: 13757    End: 13774

PACK
Adding "file1.txt" to header's file list.
Adding "file3.txt" to header's file list.
Writing header to tar...

HEADER:
File name: file1.txt     Index:0         Size: 81        Start: 13601    End: 13682
File name: file3.txt     Index:1         Size: 54        Start: 13682    End: 13736


BLANK SPACES:
There are no blank spaces
----------------------------------------------------
Size of tar file is of: 13681 bytes.
PROGRAM ENDS SUCCESSFULLY
```

Prueba #2: observe the file layout and blank spaces in the packed file prior to execution. The file size decreases.



```
HEADER:
File name: file2.txt     Index:1         Size: 21        Start: 13682    End: 13703
File name: file3.txt     Index:2         Size: 54        Start: 13703    End: 13757

----------------------------------------------------
Size of tar file is of: 13775 bytes.
PROGRAM ENDS SUCCESSFULLY
sebastianbermudez@MacBook-Air-de-Sebastian Pry2_SO-tar % ./star -p out.tar
Calculating blank spaces...

BLANK SPACES:
BlankSpace ->   Start: 13601    End: 13682
BlankSpace ->   Start: 13757    End: 13774

PACK
Adding "file2.txt" to header's file list.
Adding "file3.txt" to header's file list.
Writing header to tar...

HEADER:
File name: file2.txt     Index:0         Size: 21        Start: 13601    End: 13622
File name: file3.txt     Index:1         Size: 54        Start: 13622    End: 13676


BLANK SPACES:
There are no blank spaces
----------------------------------------------------
Size of tar file is of: 13654 bytes.
PROGRAM ENDS SUCCESSFULLY
```

5.8. Analysis of test results

The present "Simple Tar" (star) program has been extensively tested, evaluating all its functionalities and possibilities. With satisfaction, the program has demonstrated optimal performance and passed all tests successfully. All operations, including creation, extraction, listing, deletion, update, aggregation, and defragmentation, have been executed consistently and smoothly, fulfilling the expected functionality. The implementation of the *star* program has proven to be effective and robust in the management of packed archives.

6. Conclusions

The culmination of the "Simple Tar" project has resulted in the creation of an efficient and easy-to-use *tar* archive management tool. This successful implementation includes key operations such as creating, viewing, extracting, deleting, and updating packed archives, providing a complete solution for the organization and management of data within a packed archive. In addition, it has been possible to effectively manage free spaces and apply *tar* file defragmentation, optimizing storage space.

During the development of the project, valuable lessons and skills have been learned. Team collaboration was essential to success, improving the communication and coordination skills of project participants. Time management proved to be a critical factor in meeting deadlines and keeping the work on track. Technical and logical challenges have been met, promoting the development of creative and effective problem-solving skills.

The project has also enriched understanding of the *tar* file format and Unix system calls, while strengthening skills in data structure management and whitespace management. The importance of maintaining thorough documentation has become evident, making code easier to understand and maintain. Taken together, these lessons and accomplishments solidify a successful learning experience in system development and file management.