

```

/*© Artur Marques, 2000*/
#include <stdio.h>
#include <stdlib.h> /*system*/
#include <string.h> /*strcat*/
#include <direct.h> /*chdir*/

#define DRV_LETTER_INT2CHAR_CONVERT 64
#define MAX_CHARS 255
#define CNOT_A_SEPARATOR '.'
#define CFOLDER_SEPARATOR '\\'

#define OK 1

#define STRING_BEFORE_DIR_NAME "<DIR>"

#define PATH_ARCHIVER "t:\\archivers\\winrar\\winrar.exe"
#define OPTIONS_ARCHIVER "a -r -m5 -sfx"
#define OUTPUT_PATH "d:\\rar_archives"
#define TMP_FILE "----1---.txt"

#define _DEBUG_

/*-----*/

/*
these structures are enough to build a dyn list of strings
*/

typedef struct _tp_node_lstr{
char *s;
struct _tp_node_lstr *next;
} tp_node_lstr;

typedef tp_node_lstr *tp_lstr; /*list of strings*/

/*-----*/

/*
argument(s):
c - a character that is to be investigated, if it is a white / separator char, or not
returns:
the character (!=0 == true) or false (0), depending on c being a separator char, or not
notes:
watch the cascading use of the switch instruction
*/

int b_char_separator (char c){
switch (c){
case ' ': /*space*/
case '\t': /*horizontal tab*/
case '\n': /*new line*/
case '\v': /*vertical tab*/
case '\r': /*carriage return*/
case '\f': /*form feed*/
return c; break; /*breaks aren't needed, because return will exit immediatly*/
default:
return 0; break;
}
}/*b_char_separator function ends*/

/*-----*/

/*
argument(s):
f - pointer to a file
returns:
next word in file, after position pointed by f
notes:
1) the returned string was dyn-memory -allocated
2) check the if that controls if eof was reached, despite the while condition
3) notice that feof returns NOT ZERO (but we can't say exactly what), as the 1st read after eof
happens
*/

char *sfile_read_word (FILE *f){
char sbuffer[MAX_CHARS]; /*to hold the word = sequence of chars until space*/
char c; /*init NOT important*/
int ipos=0; /*ipos will address successive buffer positions*/
int bseparator=0; /*boolean that control if separator / end of word is found*/
int ifeof_ret=0;

while ((!bseparator) && (!feof (f))){ /*while c is not a separator, go on forming sbuffer*/

```

```

        fscanf (f, "%c",&c); /*read a char, from file*/
        if (!feof(f)) /*must check again if eof was reached, because it will not be considered a
separator*/
            if (!(bseparator=b_char_separator (c)){ /*bseparator holds true of false, depending
on c being a separator, or not*/
                sbuffer[ipos]=c;
                ipos++;
            }
            else /*c was a separator*/
                sbuffer[ipos]='\0'; /*terminate sbuffer*/
        else
            sbuffer[ipos]='\0'; /*terminate sbuffer, if eof was reached*/
    }/*while*/

    return strdup(sbuffer); /*return new memory (buffer will now die)*/
}/*file_read_word function ends*/

/*-----*/

/*
argument(s):
    f - pointer to a file (MUST BE ALREADY OPENED!)
    cchar_stop - character that marks the end of the word to return
returns:
    next word in file, after position pointed by f, UNTIL cchar_stop
notes:
    1) the returned string was dyn-memory-allocated
    2) check the if that controls if eof was reached, despite the while condition
    3) notice that feof returns NOT ZERO (but we can't say exactly what), as the 1st read after eof
happens
*/

char *file_read_until_char (FILE *f, char cchar_stop){
    char sbuffer[MAX_CHARS]; /*to hold the word = sequence of chars until cchar_stop*/
    char c=cchar_stop+1; /*char that will hold successive chars from file*/
    int ipos=0; /*ipos will address successive buffer positions*/

    while ((!feof(f)) && (c!=cchar_stop)){
        fscanf (f, "%c", &c);
        if (!feof(f))
            if (c!=cchar_stop){
                sbuffer[ipos]=c;
                ipos++;
            }
            else /*c was the cchar_stop*/
                sbuffer[ipos]='\0'; /*terminate sbuffer*/
        else /*eof reached*/
            sbuffer[ipos]='\0'; /*terminate sbuffer*/
    }/*while*/

    return strdup(sbuffer); /*return new memory (buffer will now die)*/
}/*file_read_until_char function ends*/

/*-----*/

/*
argument(s):
    s - a string like "    Program Files", already processed after a system "dir >file.txt /on /ad"
command

returns:
    the folder name

notes:
    1) the returned string was dyn-memory-allocated
    2) 100% DEPENDENT ON THE FORMAT OF THE FILE RETURNED BY THE SYSTEM COMMAND
    3) illustrates smart use of pointer arithmetics
*/

char *sdir_name_from_sys_dir_string (char *sdir){
    char sbuffer[MAX_CHARS];
    int ipos=0;

    while ((*sdir)==' '){/*while we have those trailing spaces...*/
        sdir++;/*advance pointer*/
    }

    while ((*sdir]!='\0'){
        sbuffer[ipos]=*sdir;
        ipos++;
        sdir++;
    }/*form sbuffer*/
}

```

```

        sbuffer[ipos]='\0';/*terminate sbuffer*/
    return strdup(sbuffer);
}/*sdir_name_from_sys_dir_string function ends*/

/*-----*/
/*
argument(s):
    lstr_list - dyn list of dyn strings, where we'll add the next argument
    s2add - dyn string to add to the dyn list of dyn strings (MUST ALREADY HAVE SANE MEMORY!)

returns:
    nothing - that is why the dyn list of dyn strings must be received by REFERENCE

notes:
    1) check the reference call for the dyn list => it is the ONLY WAY to change it, when it is NULL
*/
void lstr_add (tp_lstr *lstr_list, char *s2add){
    tp_lstr lstr_new_node=malloc (sizeof (tp_node_lstr)), aux=*lstr_list;

    if (lstr_new_node){
        /*build new node*/
        lstr_new_node->s=s2add;
        lstr_new_node->next=NULL;
        /*build done!*/

        /*if there already is a list*/
        if (aux){
            /*search end of list*/
            while (aux->next) aux=aux->next;
            /*join new node to the end*/
            aux->next=lstr_new_node;
        }
        else
            /*no list? list becomes the new node*/
            *lstr_list=lstr_new_node;
    }
    else ; /*no memory? do nothing*/
}/*function lstr_add ends*/

/*-----*/

/*
argument(s):
    lstr_list - dyn list of dyn strings, that is to have its 1st element removed

returns:
    nothing - that is why the dyn list of dyn strings must be received by REFERENCE

notes:
    1) check that two (2) free operations are needed to truly free memory
*/
void lstr_remove_header (tp_lstr *lstr_list){
    tp_lstr deleter=*lstr_list;

    if (lstr_list){
        if (deleter){
            (*lstr_list)=(*lstr_list)->next;
            free (deleter->s);
            free (deleter);
        }
    }
}/*function lstr_remove_header ends*/

/*-----*/

/*
argument(s):
    lstr_list - dyn list of dyn strings, that is to be ABSOLUTELY destroyed

returns:
    nothing - that is why the dyn list of dyn strings must be received by REFERENCE

notes:
    1) check that two (2) free operations are needed to truly free memory
*/
void lstr_destroy (tp_lstr *lstr_list){

```

```

tp_lstr deleter=*lstr_list; /*deleter will always point to what to delete*/

if (lstr_list)
    while (*lstr_list){
        deleter=*lstr_list;
        *lstr_list=(*lstr_list)->next;
        free (deleter->s);
        free (deleter);
    }
}/*function lstr_destroy ends*/

/*-----*/
/*
argument(s):
    list - dyn list of dyn strings, that is to be printed (to sdtout)

returns:
    nothing

notes:
    1) notice that because the call is "by value" there is no need to send a * to a *
*/

void lstr_print (tp_lstr list){
    int icounter=0;

    if (list)
        while (list){
            icounter++;
            printf ("%d = %s\n",icounter, list->s);
            list=list->next;
        }
    else
        printf ("NULL LIST\n");
}/*lstr_print function ends*/

/*-----*/
/*
argument(s):
    file_name - name of a file that is the result of a system "dir >filename /on /ad" command

returns:
    a dynamic list of dynamic strings, that are the DIRS' names

notes:
    1) because some of the functions it uses are system dependent, this function is (indirectly)
highly dependent on sys behavior too
*/

tp_lstr lstr_get_dirs_from_file (char *file_name){
    FILE *f=fopen (file_name, "r");
    char *s_a_word=NULL; /*word to be read*/
    char *sdir=NULL; /*final string of a dir name*/
    tp_lstr lstr_dirs=NULL; /*the dyn list of dyn strings to return*/
    int i=0;

    if (f){
        while (!feof (f)){
            s_a_word=sfile_read_word (f);
            /*if we found the string that is right BEFORE the folder name*/
            if (strcmp (s_a_word, STRING_BEFORE_DIR_NAME)==0){
                free(s_a_word);/*ok, we don't need the token / marker anymore*/
                s_a_word=sfile_read_until_char (f, '\n');/*go get what follows until \n*/
                sdir=sdir_name_from_sys_dir_string (s_a_word);/*NEW sane memory*/

                free (s_a_word);/*again, did its job and is not needed anymore*/
                s_a_word=NULL;

                /*notice: sending a reference!*/
                lstr_add (&lstr_dirs, sdir);/*add this string to the list of strings*/
            }

            if (s_a_word)
                free (s_a_word); /*only because it was dyn-mem from function
file_read_word*/
        }/*while*/
        fclose (f);
    }
    return lstr_dirs; /*returns the list of strings that are the DIRS' names*/
}/*lstr_get_dirs_from_file function ends*/

```

```

/*-----*/
/*
    argument(s):
        list - the list whose size is to be known

    returns:
        list size
*/
int lstr_size (tp_lstr list){
    int isize=0;

    while (list){
        isize++;
        list=list->next;
    }
    return isize;
}/*function lstr_size ends*/

/*-----*/
/*
    argument(s):
        idrive - an int as usable by _chdrive
        sabs_path - an absolute path (as returned by _getcwd, ie with drive name and single \)
*/
tp_lstr get_list_of_dirs (int idrive, const char *sabs_path){
    char sbuffer[MAX_CHARS];
    tp_lstr list_of_dirs=NULL;

    sbuffer[0]='\0';
    strcat (sbuffer, "dir >");
    strcat (sbuffer, TMP_FILE);
    strcat (sbuffer, " /on /ad");

    system (sbuffer);/*TMP_FILE is a tmp file created for listing the folder's contents*/

    /*go get them!*/
    list_of_dirs=lstr_get_dirs_from_file (TMP_FILE);

    return list_of_dirs;
}/*function get_list_of_dirs ends*/

/*-----*/
/*
    argument(s):
        spath - a full path, with NO drive mention, eg "\\aa\bb\cc"

    returns:
        a dynamic list of dynamic strings, that are the DIRS in the path, eg "aa" -> "bb" -> "cc"

    notes:
        1) interesting string manipulation
*/
tp_lstr get_list_of_folders_in_path (char *spath){
    char sbuffer[MAX_CHARS];
    int      ibpos=0, /*buffer position*/
            ipos=0, /*position on original string*/
            isize=strlen (spath); /*length of original string*/

    tp_lstr list_of_folders=NULL; /*list of folders mentioned in original string path*/

    /*do we have a string to check?*/
    if (spath){
        /*yes, we do*/
        sbuffer[0]='\0'; /*make sure buffer is NULL*/

        /*while we didn't investigate the whole of the original string*/
        while (ipos <= isize){

            /*go build a sbuffer, until a \\ appears*/
            if (((*spath)!=CFOLDER_SEPARATOR) && ((*spath)!='\0')){
                sbuffer[ibpos]=(*spath);
                ibpos++;
            }
            else{
                /*did we get only a single \\, or something else?*/

```

```

                                if (ibpos!=0){
                                    /*if something else, then we got a folder name*/
                                    sbuffer[ibpos]='\0'; /*finalize buffer*/
                                    lstr_add (&list_of_folders, strdup (sbuffer)); /*add folder name to
list*/
                                    sbuffer[0]='\0'; ibpos=0; /*reset buffer to next usage*/
                                }
                            }
                            spath++; /*check next char on original string*/
                            ipos++;
                        } /*while*/

#ifdef _DEBUG_
                            lstr_print (list_of_folders);
#endif
                    } /*if*/
                    return list_of_folders;
} /*function get_list_of_folders_in_path ends*/

/*-----*/

/*
    argument(s):
        sdir_name - directory name to which the (archiver) system command is to be applied
        sfull_path - string that says the full_path up to (inclusive) the directory that is be system
archived (this full path has no drive ref)
        imirror - boolean that says if the sys command string to return, should ask archiving to a mirror
folder in the destination path

    returns:
        a complete system command string et
        "t:\archivers\winrar\winrar.exe a -r -m5 -sfx
d:\rar_archives\w3\audio\hw\naim_audio_dot_com naim_audio_dot_com\*. *"

    notes:
        1) interesting string manipulation
        2) notice it returns a strdup => other functions should free memory, when not needed
*/
char *sbuild_sys_command_string (char *sdir_name, char *sfull_path, int imirror){
    char sbuffer[2*MAX_CHARS];
    char sdest_archive_name [MAX_CHARS];

    sbuffer[0]='\0';
    strcat (sbuffer, PATH_ARCHIVER);
    strcat (sbuffer, " ");
    strcat (sbuffer, OPTIONS_ARCHIVER);
    strcat (sbuffer, " ");
    /*aim is to form a string like (for example) "t:\archivers\winrar\winrar.exe a -r -m5 -sfx" */

    /*form destination archive full path name (INCLUDING DRIVE)*/
    sdest_archive_name[0]='\0';
    strcat (sdest_archive_name, OUTPUT_PATH);
    if (imirror)
        strcat (sdest_archive_name, sfull_path);
    /*sdest_archive_name should mirror current path, eg OUTPUT_PATH\current_full_path\*/

    strcat (sdest_archive_name, "\\");
    strcat (sdest_archive_name, sdir_name);

    /*
    if imirror_path is set then
        sdest_archive_name == OUTPUT_PATH\current_full_path\archive_name
    else
        sdest_archive_name == OUTPUT_PATH\archive_name
    */

    /*use the destination archive name to end the system command string*/
    strcat (sbuffer, sdest_archive_name);
    strcat (sbuffer, " ");
    strcat (sbuffer, sdir_name);
    strcat (sbuffer, "\\*. *");

    /*keep the .tpp too => SAVING THE TPP CUTS GENERALITY*/
    strcat (sbuffer, sdir_name); strcat (sbuffer, ".tpp");
    /*aim is to complete the above string with archive and file names, eg " archive_name files"*/

    return strdup (sbuffer);
} /*function sbuild_sys_command ends*/

/*-----*/

```

```

/*
argument(s):
    list - list of strings that are DIRs and that are to be archived
    icurrent_drive - drive from where the compression will happen
    scurrent_full_path - the full path to the DIR where compression will happen
    imirror_path - should the DIRs structure be mirrored in the archive folder?

returns:
    an int - for now, always OK

notes:
    1) check directory creation
    2) check successive system calls
*/
int act_on_list_of_dirs (tp_lstr list, int icurrent_drive, char *scurrent_full_path, int imirror_path){
    char sbuffer_mkdir[MAX_CHARS];

    char *s4sys;
    tp_lstr list_of_sys_commands=NULL;
    tp_lstr list_of_folders_in_path=get_list_of_folders_in_path (scurrent_full_path);
    tp_lstr bk_folders=list_of_folders_in_path, bk_commands=NULL;
    int ifolders_depth=list_size (list_of_folders_in_path), icurrent_depth=0, f=0, icommand=0;

    /*make DIRs, if requested*/
    if(imirror_path){
        _mkdir (OUTPUT_PATH);

#ifdef _DEBUG_
        printf ("mkdir %s\n",OUTPUT_PATH);
#endif

        while(icurrent_depth < ifolders_depth){
            icurrent_depth++;
            sbuffer_mkdir[0]='\0';
            strcat (sbuffer_mkdir, OUTPUT_PATH);
            strcat (sbuffer_mkdir, "\\");
            for (f=0; f<icurrent_depth; f++){
                strcat (sbuffer_mkdir, list_of_folders_in_path->s);
                _mkdir (sbuffer_mkdir);/*try to create DIR (no problem if it already exists,
when it is to mirror the original location*/
                strcat (sbuffer_mkdir, "\\");
                list_of_folders_in_path=list_of_folders_in_path->next;/*go down a level in
folders' hierarchy*/
            }
            list_of_folders_in_path=bk_folders;/*reset hierarchy of folders*/

#ifdef _DEBUG_
            printf ("mkdir %s\n",sbuffer_mkdir);
#endif

        }/*while*/
    }/*if*/
    printf ("-----the above DIRs / folders were created-----\n");
    /*END OF make DIRs*/

    /*list of system commands*/
    while (list){
        /*get system command string*/
        s4sys=sbuild_sys_command_string (list->s, scurrent_full_path, imirror_path);

        /*add system command string to current list of system commands*/
        lstr_add (&list_of_sys_commands, s4sys);

        /*check next directory name, so the respective system command can be done*/
        list=list->next;
    }/*all system commands should now be formed*/

#ifdef _DEBUG_
    /*
    lstr_print (list_of_sys_commands);
    */
#endif

    /*do it!*/
    icommand=0;
    bk_commands=list_of_sys_commands;
    while (list_of_sys_commands){
        icommand++;
        printf ("EXECUTING command #%-d:\n%s\n",icommand, list_of_sys_commands->s);/*inform
user*/
        system (list_of_sys_commands->s);
    }
}

```

```

        list_of_sys_commands=list_of_sys_commands->next;
    }
    /*recover original list, to be destroyed*/
    list_of_sys_commands=bk_commands;
    /*end of action*/

    /*free memory*/
    lstr_destroy (&list_of_folders_in_path);
    lstr_destroy (&list_of_sys_commands);
    return OK;
}/*function act_on_list_of_dirs ends*/

/*-----*/

/* just cuts the 1st two chars from the argument string, eg "f:\bla bla" to "\bla bla"*/
char *sabs_to_relative_string (char *s){
    char sbuffer[MAX_CHARS], *sold=s;
    int ipos=0, isize=strlen (s), f=0;

    if (s && (isize>=3)){
        for (f=2; f<isize; f++){
            sbuffer[ipos]=s[f];
            ipos++;
        }
        sbuffer[ipos]='\0';
    }

    return strdup(sbuffer);
}/*function sabs_to_relative_string ends*/

/*-----*/

void main (void){
    char sbuffer[MAX_CHARS], *sabs_path=NULL;
    tp_lstr list_of_dirs=NULL;

    _getcwd (sbuffer, MAX_CHARS);
    list_of_dirs=get_list_of_dirs (_getdrive(),sbuffer);

#ifdef _DEBUG_
    printf ("current drive = %c\t Current DIR = %s\n",_getdrive()+64, sbuffer);

    /*show what we got*/
    printf ("hierarchy of identified DIRs / folders: \n");
    lstr_print (list_of_dirs);
    printf ("-----\n");
#endif

    sabs_path=sabs_to_relative_string (sbuffer);

    lstr_remove_header (&list_of_dirs);/*get rid of .*/
    lstr_remove_header (&list_of_dirs);/*get rid of ..*/

    act_on_list_of_dirs (list_of_dirs, _getdrive(), sabs_path,1);

    /*free memory*/
    lstr_destroy (&list_of_dirs);
    free(sabs_path);

#ifdef _DEBUG_
    /*show what we got*/
    lstr_print (list_of_dirs);
#endif
}/*function main ends*/

```