Malloc Task

=======main.h==========

int \_putchar(char c);

char \*create\_array(unsigned int size, char c);

char \*\_strdup(char \*str);

char \*str\_concat(char \*s1, char \*s2);

int \*\*alloc\_grid(int width, int height);

void free\_grid(int \*\*grid, int height);

char \*argstostr(int ac, char \*\*av);

char \*\*strtow(char \*str);

======== 0 - main.c===========

#include "main.h"

#include <stdio.h>

#include <stdlib.h>

/\*\*

\* simple\_print\_buffer - prints buffer in hexa

\* @buffer: the address of memory to print

\* @size: the size of the memory to print

\*

\* Return: Nothing.

\*/

void simple\_print\_buffer(char \*buffer, unsigned int size)

{

   unsigned int i;

   i = 0;

   while (i < size)

   {

       if (i % 10)

       {

           printf(" ");

       }

       if (!(i % 10) && i)

       {

           printf("\n");

       }

       printf("0x%02x", buffer[i]);

       i++;

   }

   printf("\n");

}

/\*\*

\* main - check the code for ALX School students.

\*

\* Return: Always 0.

\*/

int main(void)

{

   char \*buffer;

   buffer = create\_array(98, 'H');

   if  (buffer == NULL)

   {

       printf("failed to allocate memory\n");

       return (1);

   }

   simple\_print\_buffer(buffer, 98);

   free(buffer);

   return (0);

}

========== 0-create\_array.c ============

#include "main.h"

#include <stdlib.h>

/\*\*

\* create\_array - creates an array of chars,

\* and initializes it with a specific char.

\* @size: the size of the array

\* @c: initial value

\*

\* Return: a pointer to the array, or NULL if it fails

\*/

char \*create\_array(unsigned int size, char c)

{

  char \*array;

  unsigned int i;

  if (size == 0)

     return (NULL);

  array = (char \*)malloc(sizeof(char) \* size);

  if (array == NULL)

     return (NULL);

  for (i = 0; i < size; i++)

     array[i] = c;

  return (array);

}

========= 1-strdup.c ==============

#include "main.h"

#include <stdlib.h>

/\*\*

\* \_strdup - returns a pointer to a newly allocated space in memory,

\* which contains a copy of the string given as a parameter.

\* @str: the source string

\*

\* Return: returns a pointer to the duplicated string.

\* It returns NULL if insufficient memory was available

\*/

char \*\_strdup(char \*str)

{

  char \*copy;

  int i, len = 0;

  if (str == NULL)

     return (NULL);

  while (str[len] != '\0')

     len++;

  copy = (char \*)malloc((sizeof(char) \* len) + 1);

  if (copy == NULL)

     return (NULL);

  for (i = 0; i < len; i++)

     copy[i] = str[i];

  copy[len] = '\0';

  return (copy);

}

=============== 2-str\_concat.c ===============

#include "main.h"

#include <stdlib.h>

#include <stdio.h>

/\*\*

\* str\_concat - concatenates two strings.

\* @s1: first string

\* @s2: second string

\*

\* Return: a pointer to a newly allocated space in memory which

\* contains the contents of s1, followed by the contents of s2,

\* and null terminated. NULL on failure

\*/

char \*str\_concat(char \*s1, char \*s2)

{

  int i, j, len1, len2, len;

  char \*result;

  len1 = len2 = 0;

  if (s1 != NULL)

  {

     i = 0;

     while (s1[i++] != '\0')

        len1++;

  }

  if (s2 != NULL)

  {

     i = 0;

     while (s2[i++] != '\0')

        len2++;

  }

  len = len1 + len2;

  result = (char \*)malloc(sizeof(char) \* (len + 1));

  if (result == NULL)

     return (NULL);

  for (i = 0; i < len1; i++)

     result[i] = s1[i];

  for (j = 0; j < len2; j++, i++)

     result[i] = s2[j];

  result[len] = '\0';

  return (result);

}

============= 3-alloc\_grid.c ==============

#include "main.h"

#include <stdlib.h>

/\*\*

\* alloc\_grid - returns a pointer to a 2 dimensional array of integers.

\* @width: width of array

\* @height: height of array

\*

\* Return: a pointer to a 2 dimensional array of integers.

\*/

int \*\*alloc\_grid(int width, int height)

{

  int \*\*array, i, j;

  int len = width \* height;

  if (len <= 0)

     return (NULL);

  array = (int \*\*)malloc(sizeof(int \*) \* height);

  if (array == NULL)

     return (NULL);

  for (i = 0; i < height; i++)

  {

     array[i] = (int \*)malloc(sizeof(int) \* width);

     if (array[i] == NULL)

     {

        for (i--; i >= 0; i--)

           free(array[i]);

        free(array);

        return (NULL);

     }

  }

  for (i = 0; i < height; i++)

     for (j = 0; j < width; j++)

        array[i][j] = 0;

  return (array);

}

=============== 4-free\_grid.c =============

#include "main.h"

#include <stdlib.h>

/\*\*

\* free\_grid - frees a 2 dimensional grid

\* @grid: the grid

\* @height: height of the grid

\*/

void free\_grid(int \*\*grid, int height)

{

  int i;

  for (i = 0; i < height; i++)

     free(grid[i]);

  free(grid);

}

============== 100-argstostr.c ============

#include "main.h"

#include <stdlib.h>

/\*\*

\* argstostr - concatenates all the arguments of your program.

\* @ac: arguments count

\* @av: arguments vector

\*

\* Return: a pointer to a new string, or NULL if it fails

\*/

char \*argstostr(int ac, char \*\*av)

{

  char \*str, \*s;

  int i, j, k, len = 0;

  if (ac == 0 || av == NULL)

     return (NULL);

  for (i = 0; i < ac; i++)

  {

     s = av[i];

     j = 0;

     while (s[j++])

        len++;

     len++;

  }

  str = (char \*)malloc(sizeof(char) \* (len + 1));

  if (str == NULL)

     return (NULL);

  for (i = 0, j = 0; i < ac && j < len; i++)

  {

     s = av[i];

     k = 0;

     while (s[k])

     {

        str[j] = s[k];

        k++;

        j++;

     }

     str[j++] = '\n';

  }

  str[j] = '\0';

  return (str);

}

============== 101-strtow.c =========

#include "main.h"

#include <stdlib.h>

void util(char \*\*, char \*);

void create\_word(char \*\*, char \*, int, int, int);

/\*\*

\* strtow - splits a string into words.

\* @str: the string

\*

\* Return: returns a pointer to an array of strings (words)

\*/

char \*\*strtow(char \*str)

{

  int i, flag, len;

  char \*\*words;

  if (str == NULL || str[0] == '\0' || (str[0] == ' ' && str[1] == '\0'))

     return (NULL);

  i = flag = len = 0;

  while (str[i])

  {

     if (flag == 0 && str[i] != ' ')

        flag = 1;

     if (i > 0 && str[i] == ' ' && str[i - 1] != ' ')

     {

        flag = 0;

        len++;

     }

     i++;

  }

  len += flag == 1 ? 1 : 0;

  if (len == 0)

     return (NULL);

  words = (char \*\*)malloc(sizeof(char \*) \* (len + 1));

  if (words == NULL)

     return (NULL);

  util(words, str);

  words[len] = NULL;

  return (words);

}

/\*\*

\* util - a util function for fetching words into an array

\* @words: the strings array

\* @str: the string

\*/

void util(char \*\*words, char \*str)

{

  int i, j, start, flag;

  i = j = flag = 0;

  while (str[i])

  {

     if (flag == 0 && str[i] != ' ')

     {

        start = i;

        flag = 1;

     }

     if (i > 0 && str[i] == ' ' && str[i - 1] != ' ')

     {

        create\_word(words, str, start, i, j);

        j++;

        flag = 0;

     }

     i++;

  }

  if (flag == 1)

     create\_word(words, str, start, i, j);

}

/\*\*

\* create\_word - creates a word and insert it into the array

\* @words: the array of strings

\* @str: the string

\* @start: the starting index of the word

\* @end: the stopping index of the word

\* @index: the index of the array to insert the word

\*/

void create\_word(char \*\*words, char \*str, int start, int end, int index)

{

  int i, j;

  i = end - start;

  words[index] = (char \*)malloc(sizeof(char) \* (i + 1));

  for (j = 0; start < end; start++, j++)

     words[index][j] = str[start];

  words[index][j] = '\0';

}