stack_fp Documentation (.c)

AUTHOR

Version Version 1.0.0

Mon Jan 20 2020

Using the library

This section describes how to compile programs that use libstack_fp, and introduces its conventions.

An example program

```
#include "stack_fp.h"

int main (int argc, char **argv){
   Stack *st = NULL;
   char c[] = {'a', 'b', 'c'};
   int i = 0, len;
   char *ele = NULL;

   // init a stack with elements of type char
   st = stack_init (char_free, char_copy, char_print);
   if (!st) return EXIT_FAILURE;

   len = sizeof(c)/sizeof(char);

   // insert the values of the array c in the stack
   for (i = 0; i < len; i++)
      stack_push (st, c+i);    // CdE, memory allocation

   // Print the stack
   fprintf (stdout, "Stack size: %ld¥n", stack_size (st));
   stack_print (stdout, st);

   // extract the last element inserted in the stack
   ele = stack_pop (st);

   // Print extracted element
   fprintf(stdout, "Extracted element:¥n");
   char_print (stdout, ele);

   // Print the stack
   fprintf (stdout, "¥nStack size: %ld¥n", stack_size (st));
   stack_print (stdout, st);

   // free resources
   char_free (ele);
   stack_free (st);

   return (EXIT_SUCCESS);
}
```

The output is shown below

```
$ ./main
Stack size: 3
c
b
a
Extracted element:
c
Stack size: 2
b
a
```

Compiling and Linking

To compile the program `main.c` executes the command

$ gcc -c -o main.o main.c -Idir_lib

where `dir_lib` is the directory where is the library's interface (i.e, the `.h` files). After we have created the object file we want to use it to generate a executable program. This is done by adding the library's name to the list of object file names given to the linker, using a special flag, normally `-l` . Here is an example:

```
$gcc main.o -Ldir_lib -lstack_fp -o main
```

This will create a executable program using object file `main.o` , and any symbols it requires from the `stack_fp` static library. Note that we omitted the `lib` prefix and the `.a` suffix when mentioning the library on the link command. The linker attaches these parts back to the name of the library to create a name of a file to look for. Note also the usage of the `-L` flag - this flag tells the linker that libraries might be found in the given directory (in this example the directory `dir_lib`), in addition to the standard locations where the compiler looks for system libraries.

Creación y uso de bibliotecas estáticas

Introducción

Las bibliotecas son una forma sencilla y versatil de modularizar y reutilizar código. Una biblioteca es un archivo que contiene varios ficheros objetos `(.o)` y que , por tanto, puede ser usado como un único programa en la fase de enlazado con otros programas.

Los sistemas `UNIX` permiten crear dos tipos diferentes de bibliotecas, las bibliotecas estáticas y las bibliorecas dinámicas. En este documento se aborda el proceso de creación, complilación y uso de una biblioteca estática.

Creación de la biblioteca estática

Una biblioteca estática consta de:

- Un conjunto de ficheros objetos (`.o`) empaquetados en un único archivo cde *filetype* `.a` .
- Un conjunto de ficheros (`.h`) con la interfaz de la biblioteca.

Para crear una biblioteca estática debemos:

1. Compilar cada uno de los objetos que forman la biblioteca
2. Empaquetar los `.o` en un único archivo por medio la utilidad **ar**

Por convenio, los nombres de todas las bibliotecas estáticas comienzan por `lib` y tienen `.a` por extensión.

Ejemplo:

Supongamos que disponemos de los ficheros `stack_fp.c` y `stack_types.c` que contienen el código de las funciones que deseamos incluir en la biblioteca `libstack_fp.a`. Para obtener los ficheros objeto compilamos los ficheros fuente:

```
$ gcc -c -o stack_fp.o stack_fp.c
$ gcc -c -o stack_types.o stack_types.c
```

A continuación empaquetamos los ficheros obtenidos `.o` con ar (un empaquetador similar a `tar`).

```
$ ar rcs libstack_fp.a stack_fp.o stack_types.o
```

El significado de las opciones que se le dan a `ar` es el siguiente:

Flag	Significado
r	Reemplaza los ficheros si ya existían en el paquete.
c	Crea el paquete si no existe.
s	Construye un índice del contenido.
t	Lista el contenido de un paquete (o biblioteca).
x	Extrae un fichero de un paquete (o biblioteca).

Uso de una biblioteca estática

Para utilizar las funciones de la biblioteca estática en una aplicación (por e.g en un programa `main.c`) es necesario enlazar las funciones incluidas en la biblioteca cuando creamos el ejecutable de la aplicación. Para ello debemos indicar al compilador qué biblioteca queremos utilizar y el lugar donde se halla.

Ejemplo:

Supongamos que hemos escrito un programa `main.c` que hace uso de las funciones incuidas en la biblioteca `stack_fp.a` creada en el apartado anterior. Para crear el ejecutable `main` debemos:

1 Compilar el programa `main.c` :

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

Notad que al incluir el fichero `main.c` llamadas a las funciones cuyos prototipos estan declarados en los ficheros `.h` de la interfaz de la biblioteca (e.g., `#include` **`stack_fp.h`**), estos deberían estar un directorio incluido en el `path` del compilador. De no ser así debe indicarse al compilador donde se encuentran los ficheros de la interfaz mediante la opción `-I` :

```
$ gcc -c -o main.o main.c -Idir_lib
```

donde `dir_lib` es el directorio donde hemos guardado los ficheros `.h` de la biblioteca

2 Enlazar el programa con la biblioteca indicando donde está y cual es su nombre:

```
$ gcc -o main main.o -Ldir_lib -lstack_fp
```

Consideraciones :

En el ejemplo se supone que la biblioteca y su fichero de interfaz se encuentran en un directorio llamado dir_lib.

- La opción `-I` se usa para indicar donde se encuentran los ficheros de la interfaz.
- La opción `-L` indica el directorio donde se encuentra la biblioteca.
- La opción `-l` indica los nombres de la bibliotecas que se van a usar. Sin emabrgo los nombres de las bibliotecas no deben escribirse ni con el prefijo lib ni con la extensión .a ya que el compilador espera que se sigan las normas de nombrado anteriormente citadas.

Compilación con Makefile

A continuación se muestra un fichero `makefile` que permite automatizar todo el proceso, tanto la creación de la biblioteca como del programa. En este caso se supone que todos los ficheros fuentes se encuentran en el directorio actual.

```
CC=gcc
CFLAGS=-Wall -ggdb
IFLAGS=-I./
LDFLAGS=-L./
LDLIBS=-lstack_fp
### -lm enlaza la biblioteca matematica, -pthread enlaza la biblioteca de hilos
LIBS = -lm -pthread

all: libstack_fp.a main

######################################################################
# $@ es el item que aparece a la izquierda de ':'
# $< es el primer item en la lista de dependencias
# $^ son todos los archivos que se encuentran a la derecha de ':'
########################################################################

main: main.o libstack_fp.a
   $(CC) -o $@ $< $(LDFLAGS) $(LDLIBS) $(LIBS)

main.o: main.c
```

```
    $(CC) -c -o $@ $< $(CFLAGS) $(IFLAGS)

#####################################################################
###### Crea la biblioteca con las fuentes:  stack_fp.c stack_types.c
#####################################################################

stack_fp.o: stack_fp.c stack_fp.h stack_types.h
    $(CC) -c -o $@ $< $(CFLAGS)

stack_types.o: stack_types.c stack_types.h
    $(CC) -c -o $@ $< $(CFLAGS)

libstack_fp.a: stack_fp.o stack_types.o
    $(AR) rcs $@ $^

libs: libstack_fp.a

clean:
    rm  -f *.o *.a
```

Si ya disponemos del fichero de la biblioteca `(libstack_fp.a)` y unicamente queremos crear el ejecutable, nuestro `Makefile` se reduciría a las siguientes sentencias:

```
CC=gcc
CFLAGS=-Wall -ggdb
IFLAGS=-I./
LDFLAGS=-L./
LDLIBS=-lstack_fp
# -lm enlaza la biblioteca matematica, -pthread enlaza la biblioteca de hilos
LIBS = -lm -pthread

all: main

######################################################################
# $@ es el item que aparece a la izquierda de ':'
# $< es el primer item en la lista de dependencias
# $^ son todos los archivos que se encuentran a la derecha de ':'
########################################################################

main: main.o libstack_fp.a
    $(CC) -o $@ $< $(LDFLAGS) $(LDLIBS) $(LIBS)

main.o: main.c
    $(CC) -c -o $@ $< $(CFLAGS) $(IFLAGS)
```

Uso de bibliotecas con Netbeans

Cuando para compilar el programa anterior (`main.c`) utilicemos el entorno de Netbeans, debemos modificar las propiedades del proyecto donde reside el programa `main.c` para enlazarlo con la biblioteca estática (`libstack_fp.a`). Recordad que la biblioteca consta del fichero con la biblioteca `(.a)` y su interfaz `(.h)`. Por tanto habrá que:

1 Seleccionar en la interfaz gráfica de Netbeans el directorio donde se halla la biblioteca. Para ello seleccionar la ventana donde se halla mediante la secuencia

```
File -> Project Properties -> Linkers -> Libraries -> Add Library
```

2 Seleccionar el directorio donde se halla la interfaz

```
File -> Project Properties -> C Compiler -> Include Directories
```

y una vez seleccionados los cambios se deben aplicar seleccionando `Apply`

Información extraída parcialmente de `http://arco.inf-cr.uclm.es/~dvilla/doc/repo/librerias/librerias.html`

File Index

File List

Here is a list of all documented files with brief descriptions:

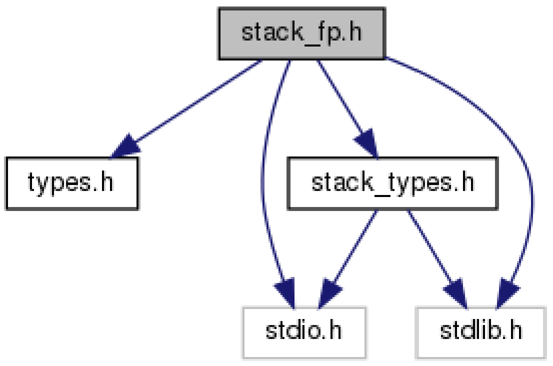
File Documentation

stack_fp.h File Reference

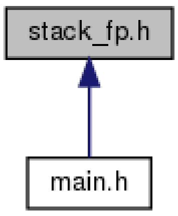
Stack library.

```
#include "types.h"
#include "stack_types.h"
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for stack_fp.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef struct **_Stack Stack**
  Structure to implement a stack. To be defined in stack_fp.c.
- typedef void(* **P_stack_ele_free**) (void *)
  Typedef for a function pointer to free a stack element.
- typedef void *(* **P_stack_ele_copy**) (const void *)
  Typedef for a function pointer to copy a stack element.
- typedef int(* **P_stack_ele_print**) (FILE *, const void *)
  Typedef for a function pointer to print a stack element at stream.

Functions

- **Stack** * **stack_init** (**P_stack_ele_free** f1, **P_stack_ele_copy** f2, **P_stack_ele_print** f3)
  This function initializes an empty stack.
- void **stack_free** (**Stack** *s)
  This function frees all the memory used by the stack.
- Status **stack_push** (**Stack** *s, const void *ele)
  This function is used to insert a element at the top of the stack.

- void * **stack_pop** (**Stack** *s)
  This function is used to extract a element from the top of the stack.
- void * **stack_top** (**Stack** *s)
  This function is used to reference the top (or the newest) element of the stack.
- Bool **stack_isEmpty** (const **Stack** *s)
  Returns whether the stack is empty.
- Bool **stack_isFull** (const **Stack** *s)
  Returns whether the stack is full.
- size_t **stack_size** (const **Stack** *s)
  This function returns the size of the stack.
- int **stack_print** (FILE *fp, const **Stack** *s)
  This function writes the elements of the stack to the stream.

Detailed Description

Stack library.

Author:
E. Serrano

Date:
10 Dec 2018

Version:
1.0

Stack interface

See also:
`http://www.stack.nl/~dimitri/doxygen/docblocks.html`
`http://www.stack.nl/~dimitri/doxygen/commands.html`

Function Documentation

void stack_free (Stack * *s*)

This function frees all the memory used by the stack.

Parameters:

s	A pointer to the stack

Stack* stack_init (P_stack_ele_free *f1*, P_stack_ele_copy *f2*, P_stack_ele_print *f3*)

This function initializes an empty stack.

Parameters:

f1	A pointer to a funcion to free a element of the stack.
f2	A pointer to a funcion to copy a element of the stack.
f3	A pointer to a funcion to print a element of the stack.

Returns:

This function returns a pointer to the stack or a null pointer if insufficient memory is available to create the stack.

Bool stack_isEmpty (const Stack * *s*)

Returns whether the stack is empty.

Parameters:

s	A pointer to the stack.

Returns:

TRUE or FALSE

Bool stack_isFull (const Stack * *s*)

Returns whether the stack is full.

Parameters:

s	A pointer to the stack.

Returns:

TRUE or FALSE

void* stack_pop (Stack * *s*)

This function is used to extract a element from the top of the stack.

The size of the stack is decreased by 1. Time complexity: O(1).

Parameters:

s	A pointer to the stack.

Returns:

This function returns a pointer to the extracted element on success or null when the stack is empty.

int stack_print (FILE * *fp*, const Stack * *s*)

This function writes the elements of the stack to the stream.

Parameters:

fp	A pointer to the stream
s	A pointer to the element to the stack

Returns:

Upon successful return, these function returns the number of characters writted. The function returns a negative value if there was a problem writing to the file.

Status stack_push (Stack * *s*, const void * *ele*)

This function is used to insert a element at the top of the stack.

A copy of the element is added to the stack container and the size of the stack is increased by 1. Time complexity: O(1). This function allocates memory for a copy of the element.

Parameters:

s	A pointer to the stack.
ele	A pointer to the element to be inserted

Returns:

This function returns OK on success or ERROR if insufficient memory is available to allocate the element.

size_t stack_size (const Stack * *s*)

This function returns the size of the stack.

Time complexity: O(1).

Parameters:

s	A pointer to the stack.

Returns:

the size

void* stack_top (Stack * *s*)

This function is used to reference the top (or the newest) element of the stack.

Parameters:

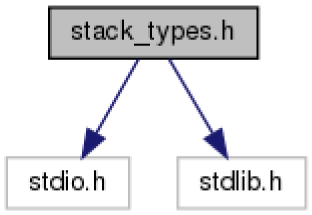
s	A pointer to the stack.

Returns:

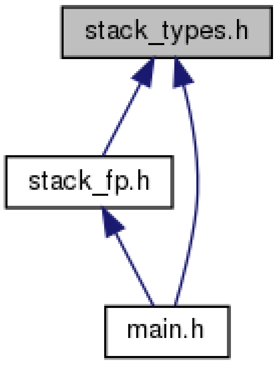
This function returns a pointer to the newest element of the stack.

stack_types.h File Reference

Stack elements.
```
#include <stdio.h>
#include <stdlib.h>
```
Include dependency graph for stack_types.h:



This graph shows which files directly or indirectly include this file:



Functions

- int * **int_init** (int a)
  This function initializes a stack element of type int.
- void * **int_copy** (const void *a)
  This function copies the integer pointed by a.
- int **int_cmp** (const void *c1, const void *c2)
  This function compares two integeres c1 and c2.
- void **int_free** (void *a)
  This function frees all the memory used by the integer.
- int **int_print** (FILE *pf, const void *a)
  This function writes the content of the pointer to integer to the stream.
- char * **char_init** (char a)
  This function initializes a stack element of type char.
- void * **char_copy** (const void *a)
  This function copies the char pointed by a.
- int **char_cmp** (const void *c1, const void *c2)
  This function compares two chars c1 and c2.
- void **char_free** (void *a)
  This function frees all the memory used by the char.
- int **char_print** (FILE *pf, const void *a)
  This function writes the content of the pointer to char to the stream.
- float * **float_init** (float a)

This function initializes a stack element of type float.

- void * **float_copy** (const void *a)
  This function copies the float pointed by a.
- int **float_cmp** (const void *c1, const void *c2)
  This function compares two floats c1 and c2.
- void **float_free** (void *a)
  This function frees all the memory used by the float.
- int **float_print** (FILE *pf, const void *a)
  This function writes the content of the pointer to char to the stream.
- void * **string_copy** (const void *src)
  This function copies the string pointed by src.
- int **string_cmp** (const void *c1, const void *c2)
  This function compares two strings c1 and c2.
- void **string_free** (void *str)
  This function frees all the memory used by the string.
- int **string_print** (FILE *pf, const void *str)
  This function writes a string to the stream.

Detailed Description

Stack elements.

Author:
E. Serrano

Date:
10 Dec 2018

Version:
1.0

Functions to manage the stack elements

See also:
`http://www.stack.nl/~dimitri/doxygen/docblocks.html`
`http://www.stack.nl/~dimitri/doxygen/commands.html`

Function Documentation

int char_cmp (const void * *c1*, const void * *c2*)

This function compares two chars c1 and c2.

Parameters:

c1	Pointer to the first char
c2	Pointer to the second char

Returns:
It returns an integer less than, equal to, or greater than zero if c1 is found, respectively, to be less than, to match, or be greater than c2.

void* char_copy (const void * *a*)

This function copies the char pointed by a.

This function allocates memory for the copy.

Parameters:

a	Pointer to the char

Returns:

This function returns a pointer to the new char or null if insufficient memory is available to create the char.

void char_free (void * *a*)

This function frees all the memory used by the char.

Parameters:

a	A pointer to the char

char* char_init (char *a*)

This function initializes a stack element of type char.

Parameters:

a	Value of the char

Returns:

This function returns a pointer to the element if insufficient memory is available to create the stack.

int char_print (FILE * *pf*, const void * *a*)

This function writes the content of the pointer to char to the stream.

Parameters:

pf	A pointer to the stream
a	A pointer to the element

Returns:

Upon successful return, these function return the number of characters. If an output error is encountered, a negative value is returned.

int float_cmp (const void * *c1*, const void * *c2*)

This function compares two floats c1 and c2.

Parameters:

c1	Pointer to the first float
c2	Pointer to the second float

Returns:

It returns an integer less than, equal to, or greater than zero if c1 is found, respectively, to be less than, to match, or be greater than c2.

void* float_copy (const void * *a*)

This function copies the float pointed by a.

This function allocates memory for the copy.

Parameters:

a	Pointer to the float

Returns:

This function returns a pointer to the new float or null if insufficient memory is available to create the float.

void float_free (void * *a*)

This function frees all the memory used by the float.

Parameters:

a	A pointer to the float

float* float_init (float *a*)

This function initializes a stack element of type float.

Parameters:

a	Value of the float

Returns:

This function returns a pointer to the element if insufficient memory is available to create the stack.

int float_print (FILE * *pf*, const void * *a*)

This function writes the content of the pointer to char to the stream.

Parameters:

pf	A pointer to the stream
a	A pointer to the element

Returns:

Upon successful return, these function return the number of characters. If an output error is encountered, a negative value is returned.

int int_cmp (const void * *c1*, const void * *c2*)

This function compares two integeres c1 and c2.

Parameters:

c1	Pointer to the first integer

c2	Pointer to the second integer

Returns:

It returns an integer less than, equal to, or greater than zero if c1 is found, respectively, to be less than, to match, or be greater than c2.

void* int_copy (const void * *a*)

This function copies the integer pointed by a.

This function allocates memory for the copy.

Parameters:

a	Pointer to the integer

Returns:

This function returns a pointer to the new integer or null if insufficient memory is available to create the stack.

void int_free (void * *a*)

This function frees all the memory used by the integer.

Parameters:

a	A pointer to the integer

int* int_init (int *a*)

This function initializes a stack element of type int.

Parameters:

a	Value of the integer

Returns:

This function returns a pointer to the element if insufficient memory is available to create the stack.

int int_print (FILE * *pf*, const void * *a*)

This function writes the content of the pointer to integer to the stream.

Parameters:

pf	A pointer to the stream
a	A pointer to the element

Returns:

Upon successful return, these function return the number of characters. If an output error is encountered, a negative value is returned.

int string_cmp (const void * *c1*, const void * *c2*)

This function compares two strings c1 and c2.

This is a wrapper funtion of the function strcmp included in the library string.h

Parameters:

c1	Pointer to the first string
c2	Pointer to the second string

Returns:

It returns an integer less than, equal to, or greater than zero if c1 is found, respectively, to be less than, to match, or be greater than c2.

void* string_copy (const void * *src*)

This function copies the string pointed by src.

This function allocates memory for the copy.

Parameters:

src	Pointer to the source string

Returns:

This function returns a pointer to the new string or null if insufficient memory is available to create the string.

void string_free (void * *str*)

This function frees all the memory used by the string.

Parameters:

str	A pointer to the string

int string_print (FILE * *pf*, const void * *str*)

This function writes a string to the stream.

Parameters:

pf	A pointer to the stream
str	A pointer to the string

Returns:

Upon successful return, these function return the number of characters. If an output error is encountered, a negative value is returned.