ESAT INNOVATION SCHOOL | ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

Asignatura 2019-2020

Algoritmos

ESAT INNOVATION SCHOOL | ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

Jose Maria Maestre Quiles & Hector Ochando
Algoritmos
Curso 2022/2023
Ivan Sancho

Índice/Index

ESAT INNOVATION SCHOOL | ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

1.- Stack

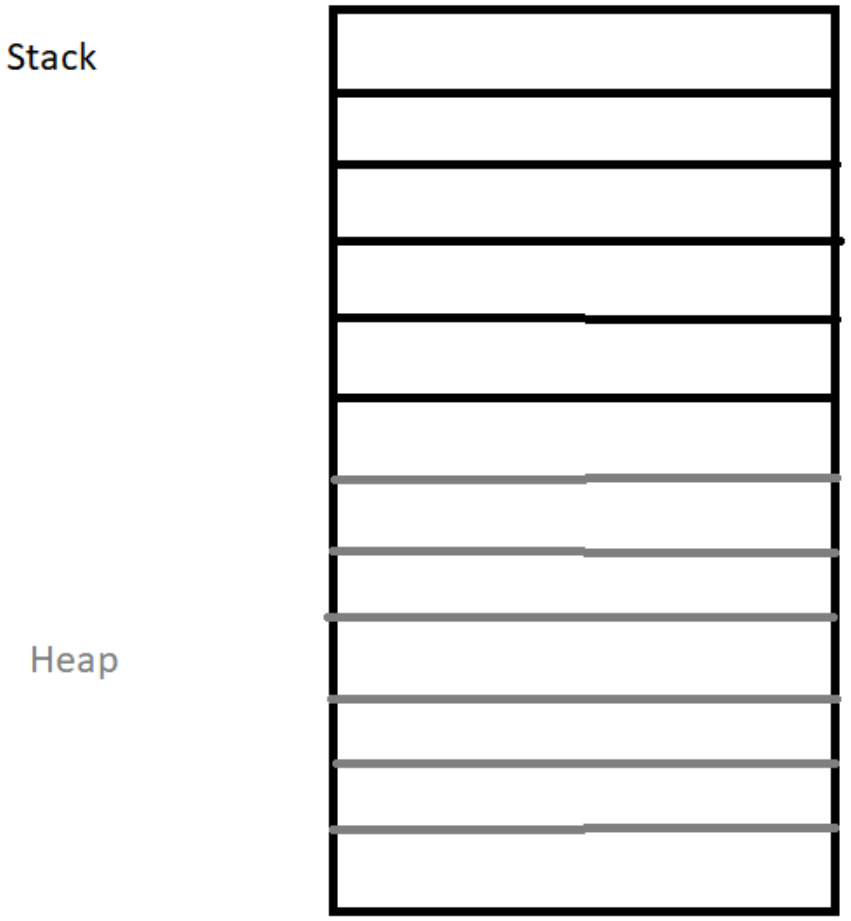
1.1.- ¿Qué es?

El stack es una zona de memoria donde se almacenan las variables locales automáticas. Esta zona de memoria se sitúa en las direcciones más altas. Es una memoria temporal, donde se almacenan las variables locales de las funciones llamadas. Para llamar a las funciones se hace “call” Cuando llamas a una función se crea un nuevo “frame” en el stack y se hace “push” de las variables locales de la función. Post cada función que se llama se crea un nuevo frame pointer. Cuando esta función acaba se hace un “pop” de las variables para liberar la memoria. Cuando esto ocurre se libera la memoria de de las variables y se elimina el frame pointer de esa función.
El stack también tiene un puntero especial que es el stack pointer, este puntero se encarga de delimitar el final de la memory stack.

La memory stack actúa como una lifo, es decir que cuando se llama(call) una función se hace un push de sus variables y se ejecuta, cuando acaba de ejecutarse se desapila la función haciendo pop de sus variables y se vuelve al anterior salto. Esto quiere decir que cuando se llama una función no se puede continuar la ejecución de la anterior hasta que se termine la primera. En el caso de que se llamen a muchas funciones una dentro de otra el stack crecerá mucho.

1.2.- Ejemplo gráfico

Tenemos la memoria Stack y a continuación tenemos el heap



Cuando creamos una variable dentro de una función, esta se almacena en el stack

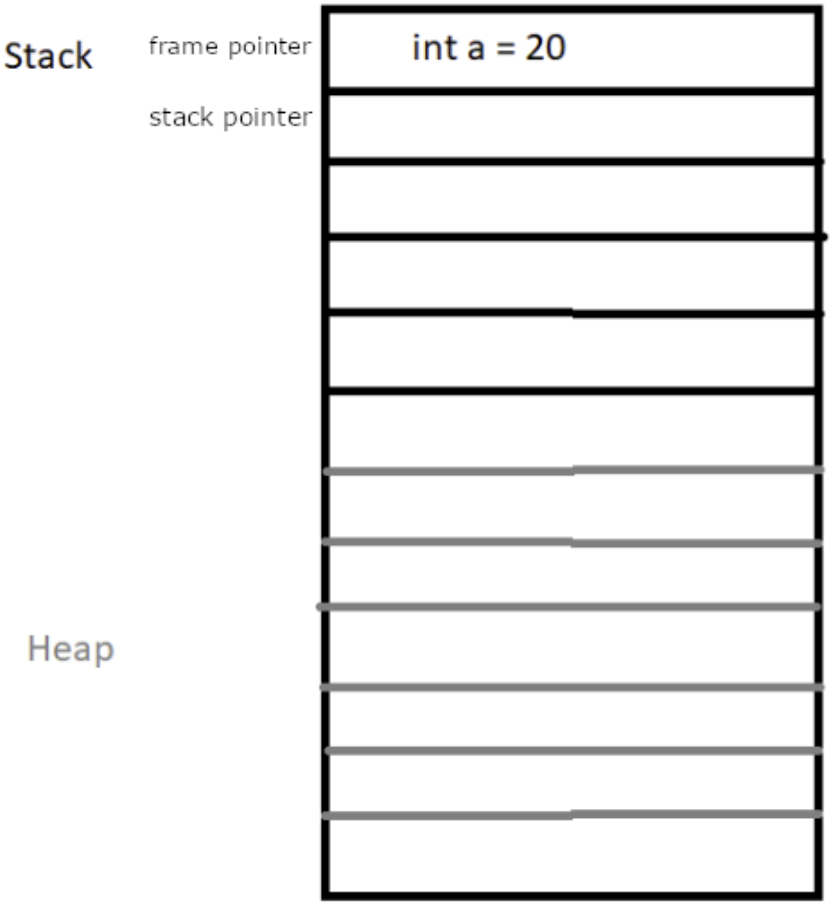
main.cc

```
#include <stdio.h>


int main(){

  int a;
  a = 20;


  return 0;
}
```

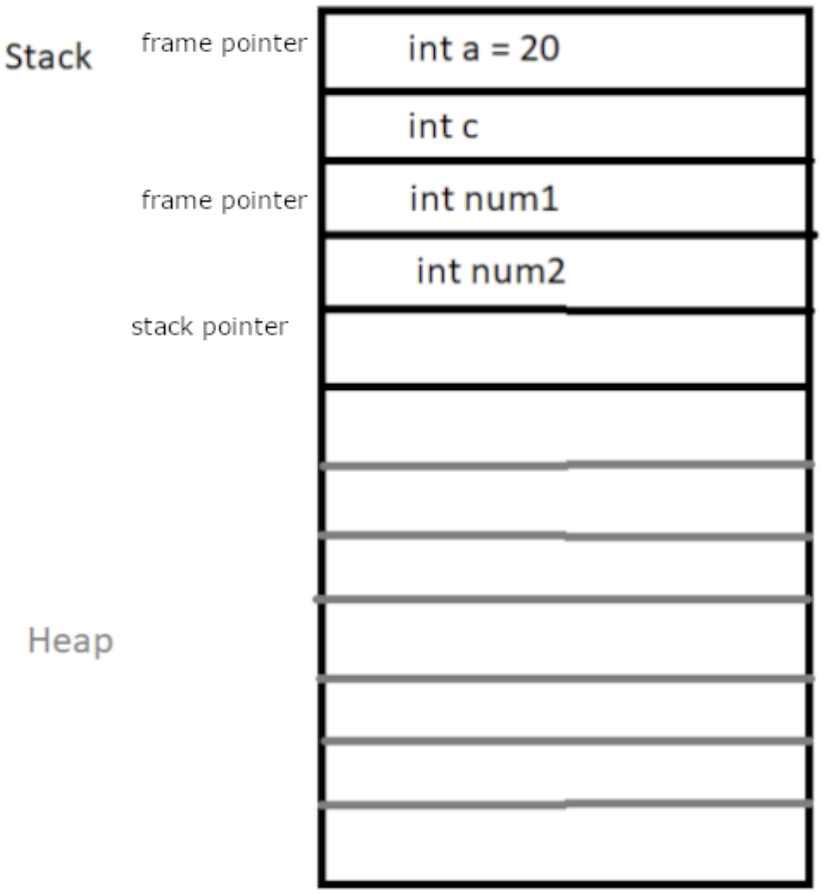


Cuando llamamos a una función, las variables utilizadas en ella, se almacenan en el stack.

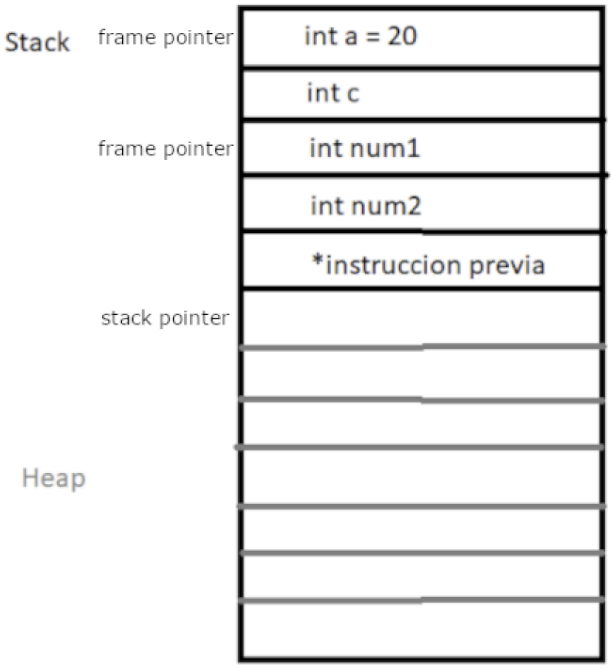
```
4   int Operation(int num1, int num2){
5
6     return num1+num2;
7   }
```

Estas variables se crean en el momento de la llamada

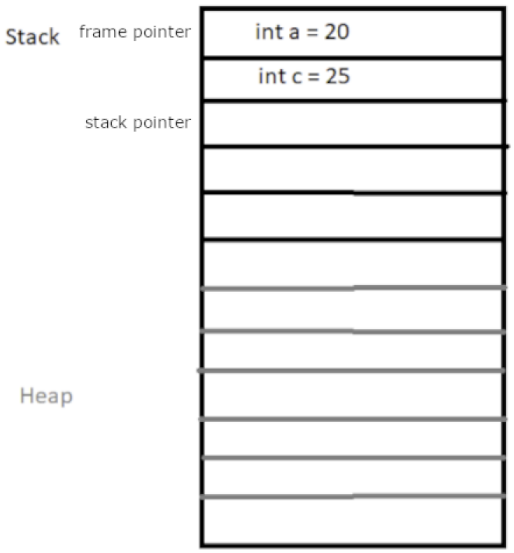
```
16    int c = Operation(a,5);
17
```



Además, se almacena un puntero a la siguiente instrucción que debe ejecutar, este puntero apunta al último bloque de memoria utilizado.

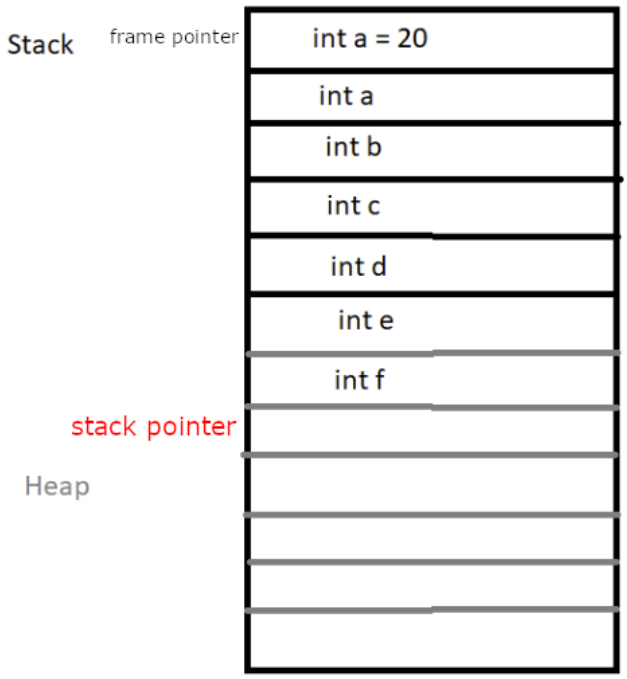


Cuando finaliza la función, todo lo que había en ella se destruye y llama a la instrucción previa



Si utilizamos excesiva memoria en el stack, sobrepasará al heap

```
4  int Operation(int num1, int num2){
5      int a;
6      int b;
7      int c;
8      int d;
9      int e;
10     int f;
11     return num1+num2;
12 }
13
```



Y obtendremos un error de overflow y se interrumpirá la ejecución del programa

2.- Bibliografía

Stack based memory [digital information] wikipedia.org [consulted 14/12/2022]. Available on: https://en.wikipedia.org/wiki/Stack-based_memory_allocation

Memory stack [digital information] tutorialspoint.com [consulted 14/12/2022]. Available on: https://www.tutorialspoint.com/what-is-memory-stack-in-computer-architecture

Memory Stack [digital information] sciencedirect.com [consulted 14/12/2022]. Available on: https://www.sciencedirect.com/topics/engineering/stack-memory

Memory Stack [digital information] sciencedirect.com [consulted 14/12/2022]. Available on: https://www.philadelphia.edu.jo/academics/qhamarsheh/uploads/Lecture%2015%20Stack%20Operations.pdf