gthc.c low_level.rs

```
GREEN TEAM HACKER CLUB {
    [Programação Low Level]

        < Heap e Stack >

}
```

Tabela de ‘Conteúdo’ {

}

```
01 {
    [Revisão]
        < Structs >
}
```


Definição < /1 > {

É uma estrutura é um tipo de dado definido pelo usuário que contém vários campos de dados. Cada campo tem um nome e um tipo de dado especificado na definição da estrutura.

}

Exemplo < /2 > {

```
struct ponto
  {
    int x;
    int y;
  };
```

}

Memória < /3 > {

Os campos da estrutura aparecem no layout da memória na ordem em que são declarados. Quando possível, os campos consecutivos ocupam bytes consecutivos dentro da estrutura. No entanto, se o tipo de um campo exigir mais alinhamento do que seria obtido dessa forma, C fornecerá o alinhamento necessário. deixando uma lacuna após o campo anterior.

Uma vez dispostos todos os campos, é possível determinar o alinhamento e o tamanho da estrutura. O alinhamento da estrutura é o alinhamento máximo de qualquer um dos campos nela contidos. exigem deixar uma lacuna no final da estrutura.

}

Exemplo < /4 > {

```c
typedef unsigned int type;
struct type {
 char c;
};

int main( int argc, char** argv ) {
 struct type st;
 type t;
 return 0;
}
```

}

Unions < /4 > {

```
union dword {
 int integer;
 short shorts[2];
};
...
dword test;
test.integer = 0×AABBCCDD;
```

}

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

02 {

[Memória]

< Heap e Stack >

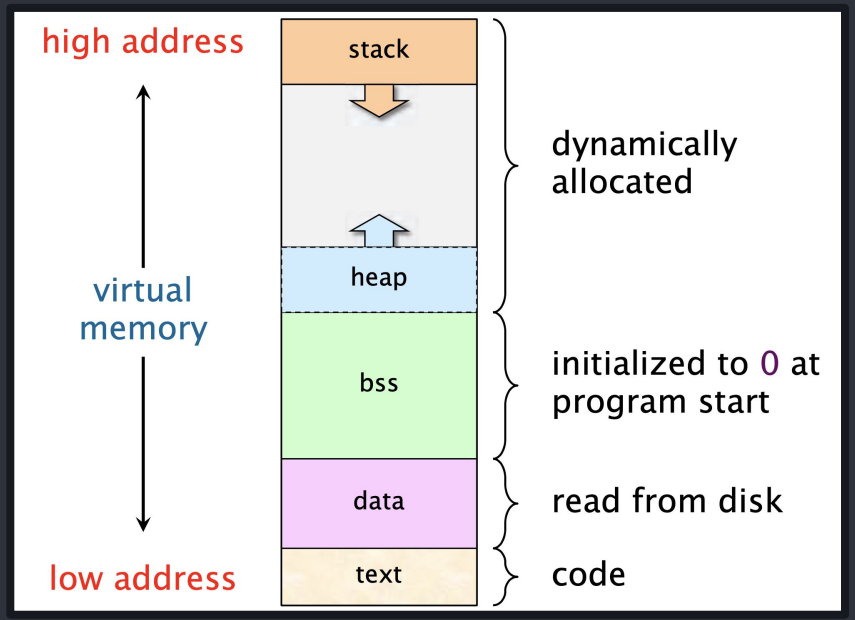
}

Como a Memória é estruturada? {

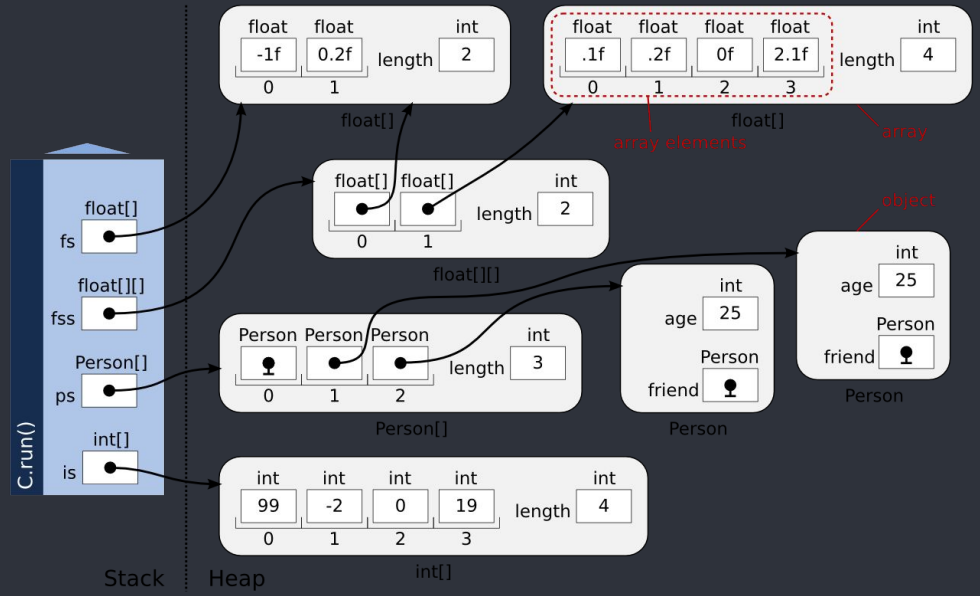
A memória é estruturada em blocos:

-text: armazena o código
-data: armazena as variáveis estáticas inicializadas, como variável global, variável estática.
-bss: armazena os dados estáticos não inicializados, como a declaração static int i em C
-heap
-stack

}



Como a Memória é estruturada? {



}

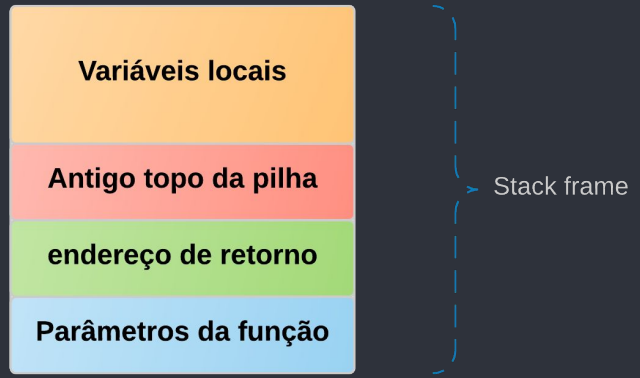
STACK; {

‘A alocação acontece em blocos contíguos de memória’

A pilha de funções (stack) é uma área da memória que aloca dados/variáveis ou ponteiros quando uma função é chamada e desalocada quando a função termina.

Essa área funciona como uma estrutura de dados LIFO (last in first out)

```
}
}
```



```
HEAP; {

    'Heap é a memória global do programa'

        O Heap, ou área de alocação dinâmica, é um espaço
        reservado para variáveis e dados criados durante a
        execução do programa (runtime)

        Utilizado para strings, structs
        Exemplo em C: malloc()

        É necessário desalocar a memória*

          }

}
```


```
1  HEAP < /Tempo > {
2
3      < No caso do Heap o acesso é relativamente baixo e depende muito do
4      runtime (forma de execução) da linguagem e da biblioteca que faz
5      alocação. >
6  }
7
8  Stack < /Tempo > {
9
10     < O acesso a variáveis alocadas na Stack são extremamente rápidos.
11     Como eles dependem apenas de um deslocamento de ponteiros, essa
12     operação tem custo muito baixo.  >
13 }
14
```


HEAP < /Scope > {

< No Heap temos que o escopo das variáveis é global. Tendo uma referência para o endereço da memória que contém o dado, é possível acessar essa variável dentro de qualquer função. >

}

Stack < /Scope > {

< Variáveis alocadas dentro da pilha (Stack) são acessíveis apenas no escopo local à função responsável por aquele stack frame. Ao final da execução da função, ou seja, ao ser desempilhadas, essas variáveis são desalocadas. >

}

```
1  HEAP < /Free > {
2
3      < Variáveis alocadas no Heap somente são desalocadas através de uma
4      instrução explícita do programa através de free() >
5
6  }
7
8  Stack < /Free > {
9
10     < Variáveis alocadas na Stack, são desalocadas quando a função
11     retorna, sendo assim desempilhadas da stack de funções  >
12
13 }
14
```


Exemplo < /1 > {

```c
#define ARRAY_SIZE 100

int main() {
      double numbers[ARRAY_SIZE];
}
```

}

Exemplo < /2 > {

```c
#define ARRAY_SIZE 100

int main() {
      double* numbers = malloc(ARRAY_SIZE * sizeof(double));
      free(numbers);
}
```

}

```
1  Overflow; {
2
3      'Um dos riscos de manipular a memória'
4
5          Pode acontecer tanto na heap quanto na stack.
6          Caso manipulado, é possível modificar o
7          endereço de retorno de uma função, acessando
8          locais indevidos na memória >
9
10         < Buffer Overflow / Stack Overflow >
11
12           }
13
14 }
```

gthc.c

low_level.rs

UFABC

Exemplo em tempo real{

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

}

Referências {

<Low Level Programming Book - https://evalandaply.neocities.org/books/lowlevelprogramming.pdf >

}

Próximo Encontro {

< Estruturas de dados e exercício >

}

```
Obrigado; {

    'Dúvidas?'

        luccas.h.cortes@hotmail.com
        https://github.com/Cortesz/

                CREDITS: This presentation template was
                created by Slidesgo, including icons by
                Flaticon, and infographics & images by Freepik

                < https://github.com/greenteamhc >

}
```