gthc.c

low_level.rs

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

        < Conteúdo: Revisão Introdução>

}
```

Tabela de ‘Conteúdo’ {

}

01 {

[Revisão]

< Sintaxe Rust >

}

Variáveis < /1 > {

```
fn main() {
    let x = 5;
    println!("The value of x is: {x}");
    x = 6;
    println!("The value of x is: {x}");
} //Variável imutável
```

}

Tipos < /2 > {

```
<Int: i8, i32, i64>
<Float: f32>
<Tuple→let tup: (i32, f64, u8) = (500, 6.4, 1);>
```

}

Funções < /3 > {

```
    fn plus_one(x: i32) → i32 {
        x + 1
    } // retorna x+1
```

}

Cargo< /4 > {

```
    <cargo new hello_cargo> //cria um projeto
    <cargo build> //compila o projeto
    <cargo run> //executa o projetos
```

}

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

02 {

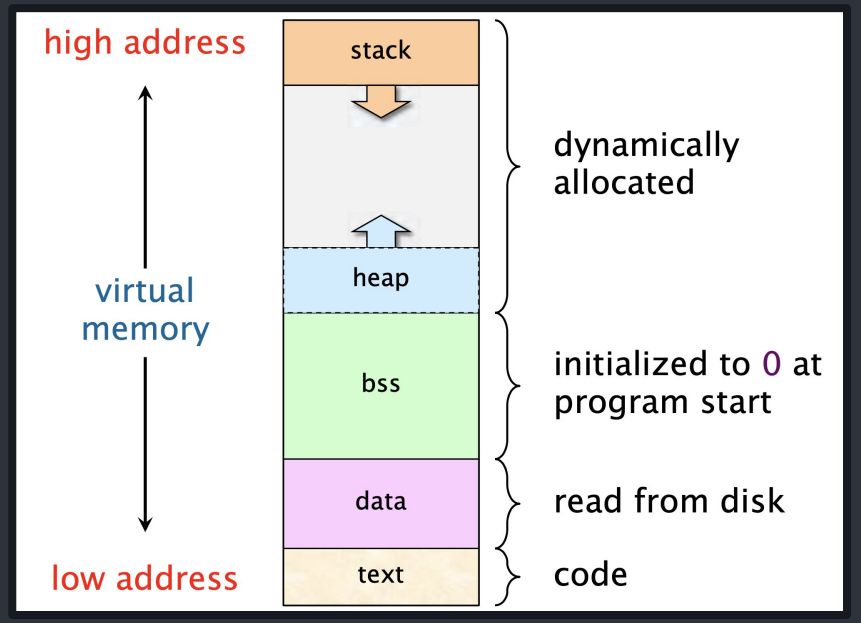
[Memória]

< Heap e Stack >

}

Como a Memória é estruturada? {

```
    < A memória é estruturada em
    blocos:
    -text
    -data
    -bss
    -heap
    -stack
    >
}
```



STACK; {

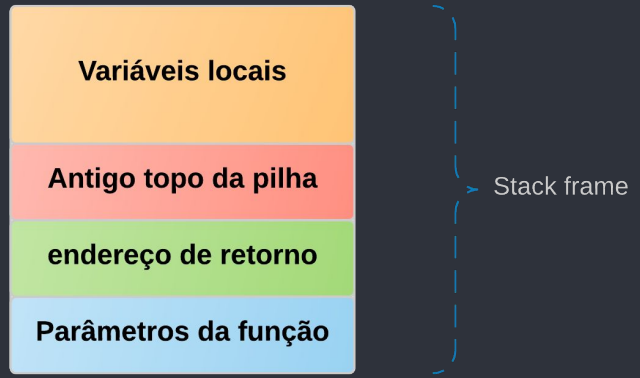
‘A alocação acontece em blocos contíguos de memória’

{ 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
```

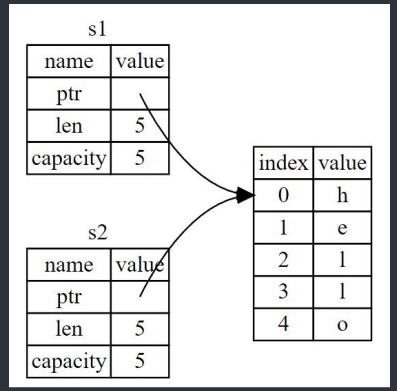

```
Overflow; {

    ‘Um dos riscos de manipular a memória’

        { Pode acontecer tanto na heap quanto na
        stack. Caso manipulado, é possível modificar o
        endereço de retorno de uma função, acessando
        locais indevidos na memória >

        < Buffer Overflow / Stack Overflow >

          }

}
```

gthc.c

low_level.rs

UFABC

```
RUST; {
    'Rust e ownership'
        {
        < Rust utiliza o sistema de ownership, onde
        as variáveis (como strings) alocadas na heap
        são desalocadas de acordo com o escopo. Ao
        reatribuir variáveis, a variável antiga
        também é desalocada. O conceito de ownership
        também é utilizado ao passar variáveis como
        parâmetros >
          }
}
```



Shallow Copy

Recursos Rust{

Crates

< https://crates.io >

Rust Book

<doc.rust-lang.org/book/title-page.html>

}

Próximo Encontro {

< Entrega do projeto; Definição de horário nas férias; Referências (4.2) >

}

```
1  Obrigado; {
2
3      'Dúvidas?'
4
5          luccas.h.cortes@hotmail.com
6          https://github.com/Cortesz/
7
8            
9
10                 CREDITS: This presentation template was
11                 created by Slidesgo, including icons by
12                 Flaticon, and infographics & images by Freepik
13
14                 < https://github.com/greenteamhc >
15 }
```