centro universitário FEI

CCP130 Desenvolvimento de Algoritmos

Prof. Danilo H. Perico

Gerenciamento de Memória

Memória do computador

- Áreas de Memória
  - ***Stack* (Alocação Estática)**:
    - Gerenciamento de memória automática
    - Funciona como uma pilha de pratos
    - Acesso muito rápido
  - ***Heap* (Alocação Dinâmica)**:
    - Gerenciamento por conta do programador
    - Variáveis podem ser acessadas globalmente
    - Não tem limite de tamanho (permite redimensionamento)
    - Acesso mais lento

Memória do computador

- Áreas de Memória
  - ***Stack* (Alocação Estática)**:
    - Ocorre em tempo de compilação
    - No momento em que se define uma variável é necessário que se definam seu tipo e tamanho
  - ***Heap* (Alocação Dinâmica)**:
    - Ocorre em tempo de execução
    - Variáveis são declaradas sem a necessidade de se definir seu tamanho, pois nenhuma memória será reservada ao colocar o programa em execução

Stack e Heap

- A alocação de ambos costumam ser realizados na Memória Principal (*RAM*)

Stack e Heap

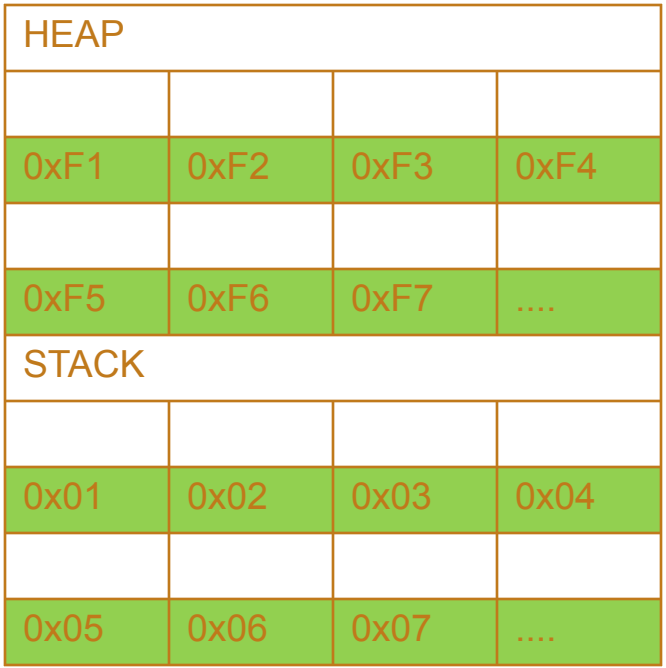
Stacks
Exemplo 1

Stacks

```
void main(){
 int i=0;
 int j=3;

 printf("i:%i j:%i",i,j);
}
```

gerenciador de memória

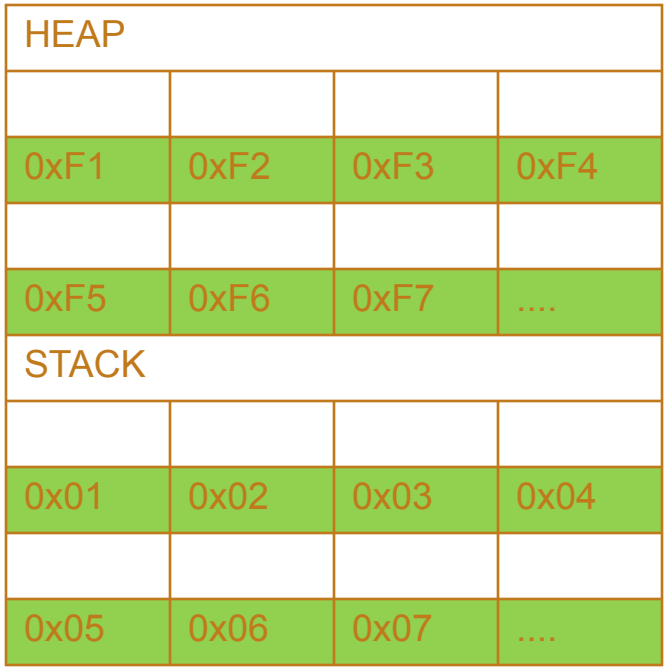


Stacks

```
void main(){
 int i=0;
 int j=3;

 printf("i:%i j:%i",i,j);
}
```

gerenciador de memória



Stacks

```
void main(){
 int i=0;
 int j=3;

 printf("i:%i j:%i",i,j);
}
```

Variável Estática!!
Alocado no Stack!
Push na memória!

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
i: 0	0	0	0
0x05	0x06	0x07	....

Stacks

```
void main(){
 int i=0;
 int j=3;

 printf("i:%i j:%i",i,j);
}
```

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
i: 0	0	0	0
0x05	0x06	0x07	....

Stacks

```
void main(){
 int i=0;
 int j=3;


 printf("i:%i j:%i",i,j);
}
```

Variável Estática!!
Alocado no Stack!
Push na memória!

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
j: 0	0	0	3
0x01	0x02	0x03	0x04
i: 0	0	0	0
0x05	0x06	0x07	....

Stacks

```
void main(){
  int i=0;
  int j=3;


  printf("i:%i j:%i",i,j);
}
```

Chamada da função print

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
j: 0	0	0	3
0x01	0x02	0x03	0x04
i: 0	0	0	0
0x05	0x06	0x07	....

Stacks

```
void main(){
 int i=0;
 int j=3;

 printf("i:%i j:%i",i,j);
}
```

Fim do Programa!
Tem coisa no Stack!
O computador limpa automaticamente

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
j: 0	0	0	3
0x01	0x02	0x03	0x04
i: 0	0	0	0
0x05	0x06	0x07	....

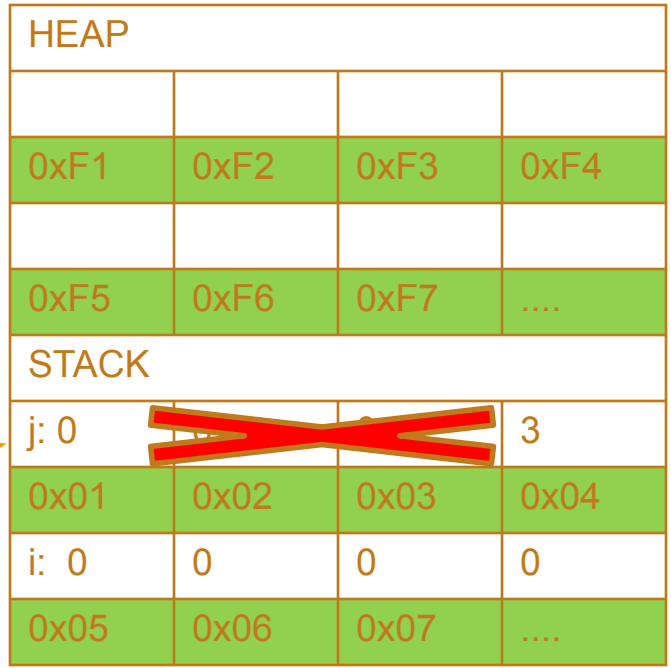
Stacks

```
void main(){
 int i=0;
 int j=3;

 printf("i:%i j:%i",i,j);
}
```

Desempilha o último empilhado

gerenciador de memória



Stacks

```
void main(){
 int i=0;
 int j=3;

 printf("i:%i j:%i",i,j);
}
```

Desempilha o último empilhado

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
i: 0	0	0	0
0x05	0x06	0x07	....

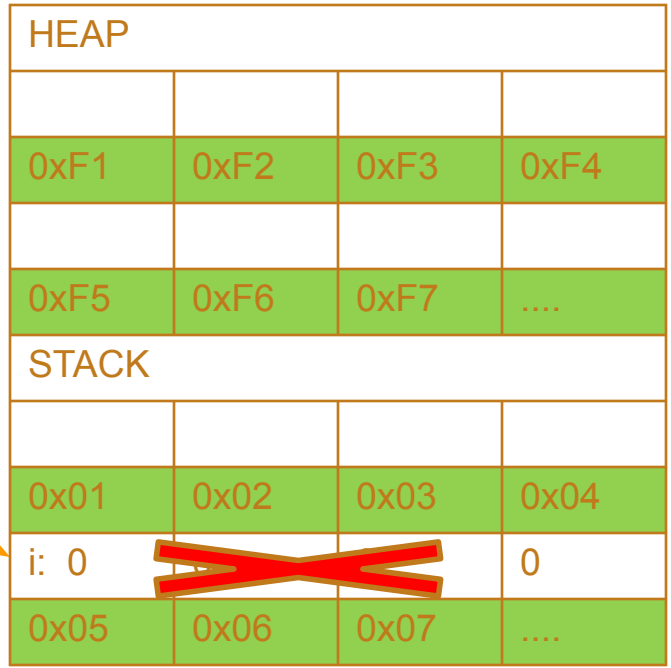
Stacks

```
void main(){
 int i=0;
 int j=3;

 printf("i:%i j:%i",i,j);
}
```

Desempilha o último empilhado

gerenciador de memória



Stacks

```
void main(){
 int i=0;
 int j=3;

 printf("i:%i j:%i",i,j);
}
```

Agora sim!

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
0x05	0x06	0x07	....

Stacks
Exemplo 2

Stacks

```
void main(){
 int i=0;
 int j=3;

 printf("i:%i j:%i",i,j);
 printf("&i = %p", &i);
}
```

```
i:0 j:3
&i = 0x05
```

gerenciador de memória

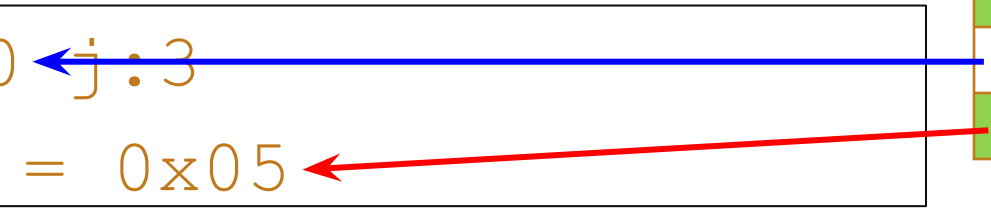
HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
j: 3			
0x01	0x02	0x03	0x04
i: 0			
0x05	0x06	0x07	....

Stacks

```
void main(){
 int i=0;
 int j=3;

 printf("i:%i j:%i",i,j);
 printf("&i = %p", &i);
}
```

```
i:0 j:3
&i = 0x05
```



gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
j: 3			
0x01	0x02	0x03	0x04
i: 0			
0x05	0x06	0x07	....

Heap

Heap - Ponteiros

- Por meio dos **ponteiros** podemos apontar para lugares reservados da memória ou para lugares que já estão em uso por outro processo
- Não precisamos criar variáveis alocadas estaticamente para obter endereços “válidos” na hora de trabalhar com ponteiros
- Podemos usar a alocação dinâmica de memória

Heap

- ***Heap*:**
  - Onde são alocadas as **variáveis dinâmicas**
  - A alocação e desalocação ocorrem através das chamadas: ***malloc*** e ***free***
  - ***#include <stdlib.h>***

Heap

- ***malloc***
  - A função ***malloc*** (***memory allocation***) aloca espaço para um bloco de bytes consecutivos na memória **RAM** e devolve o endereço desse bloco
    - Se não conseguir alocar, retorna ***NULL***
  - O número de bytes a serem alocados é especificado no argumento da função

Heap

- *malloc*
  - *exemplo:*

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int *j;
    // malloc aloca 4 bytes
    j=malloc(4);
    *j = 4;
    printf("%d", *j);
    free(j);
}
```

Heap

- ***malloc - sizeof***
  - ***sizeof*** - permite saber o número de bytes ocupado por um determinado tipo de variável.
  - Exemplo: aloca o tamanho de **um *int***

```
int *ptr = malloc(sizeof(int));
```

Heap

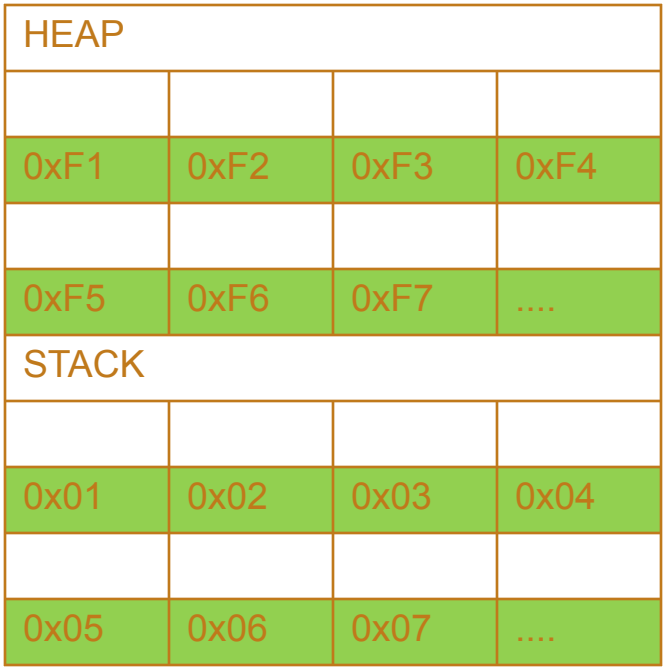
- ***free***
  - As variáveis alocadas **estaticamente** dentro de uma função, desaparecem assim que a execução da função termina. Nenhuma ação adicional é necessária
  - Já as variáveis alocadas **dinamicamente** continuam a existir mesmo depois que a execução da função termina
    - Assim, para liberar a memória ocupada por essas variáveis, é preciso recorrer à função ***free***

Heap
Exemplo

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

gerenciador de memória



Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
0x05	0x06	0x07	....

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

Onde são armazenados os endereços que os ponteiros apontam?

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
0x05	0x06	0x07	....

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

Na stack!

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: lixo	lixo	lixo	lixo
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: lixo	lixo	lixo	lixo
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

"Solicito 4 Bytes da memória"

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: lixo	lixo	lixo	lixo
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

ALOCADO!!
Posição 0xF1

gerenciador de memória

HEAP			
lixo	lixo	lixo	lixo
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: lixo	lixo	lixo	lixo
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

ALOCADO!!
Posição 0xF1

gerenciador de memória

HEAP			
lixo	lixo	lixo	lixo
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: 0	0	0	F1
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

gerenciador de memória

HEAP			
lixo	lixo	lixo	lixo
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: 0	0	0	F1
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

"Navegue" até 0xF1
e escreva "4" neste endereço

gerenciador de memória

HEAP			
lixo	lixo	lixo	lixo
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: 0	0	0	F1
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

gerenciador de memória

HEAP			
0	**0**	**0**	**4**
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: 0	0	0	F1
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

Imprima o conteúdo do endereço 0xF1

gerenciador de memória

HEAP			
0	**0**	**0**	**4**
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: 0	0	0	F1
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

4

gerenciador de memória

HEAP			
0	**0**	**0**	**4**
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: 0	0	0	F1
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

Libere o bloco de memória que começa no endereço 0xF1

4

gerenciador de memória

HEAP			
0	**0**	**0**	**4**
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: 0	0	0	F1
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

ACABOU!!

4

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: 0	0	0	F1
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

Ops...Falta a Stack

```
4
```

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
j: 0	0	0	F1
0x05	0x06	0x07	0x08

Heap

```
void main(){
 int *j;
 j=malloc(4);
 *j = 4;
 printf("%i", *j);
 free(j);
}
```

Agora sim!

```
4
```

gerenciador de memória

HEAP			
0xF1	0xF2	0xF3	0xF4
0xF5	0xF6	0xF7	....
STACK			
0x01	0x02	0x03	0x04
0x05	0x06	0x07	0x08

Heap

- ***malloc***
  - Exemplo: aloca o tamanho de **5** ***int*** (cria um **vetor** de ponteiros com 5 inteiros)

```
int *ptr = (int*)malloc(5*sizeof(int));
```

Heap

- ***calloc***
  - Podemos usar também a função ***calloc***
  - Faz a mesma coisa que ***malloc***, mas a chamada é diferente

```
int *ptr = (int*)calloc(5, sizeof(int));
```

Heap

- ***realloc***
  - Altera dinamicamente a alocação de memória de uma memória previamente alocada

```
int *ptr = (int*)calloc(5, sizeof(int));
```

```
ptr = realloc(ptr, 10 * sizeof(int));
```

Heap

- *realloc*
  - Altera dinamicamente a alocação de memória de uma memória previamente alocada

```
int *ptr = (int*)calloc(5, sizeof(int));
```

```
ptr = realloc(ptr, 10 * sizeof(int));
```

Exemplos em aula

Exercícios

Exercícios

1. Faça um programa em C para **criar dinamicamente** memória para **int**, **char** e **float**. Teste o programa recebendo e imprimindo valores para cada tipo.

2. Faça um programa que crie um **vetor de tamanho 10 dinamicamente por ponteiros**.

3. Escreva um programa em C para **alocar memória dinamicamente** para armazenar ***N*** números inteiros inseridos pelo usuário e, em seguida, exiba a soma de todos os ***N*** números.