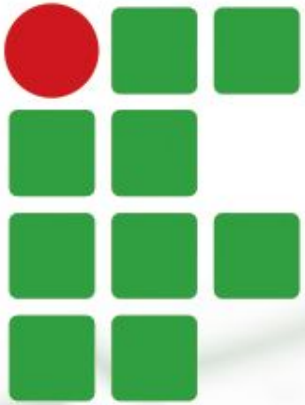


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# *Estruturas de Dados 2*

## *- Function Pointers -*

Prof. MSc. Adriano Antunes Prates

# Function Pointers

- Observe que interessante…

```
#include <stdio.h>

int soma(int a, int b){
    printf("Realizando Soma %d + %d == ",a,b);
    return a+b;
}

int prod(int a, int b){
    printf("Realizando Produto %d x %d == ",a,b);
    return a*b;
}
```

Temos duas funções (**soma e prod**) que calculam operações matemáticas, **mas sem apresentar o resultado final**…

# Function Pointers

- Observe que interessante…

```
#include <stdio.h>

void calcular(int (*func)(), int a, int b){
    int res = func(a,b);
    printf("%d\n",res);
}

int prod(int a, int b){
    printf("Realizando Produto %d x %d == ",a,b);
    return a*b;
}
```

A função **calcular** recebe como argumento, **um ponteiro para função (*Pointer Function*) que será executada internamente**, adaptando assim o seu comportamento.

# Function Pointers

- Observe que interessante…

```c
#include <stdio.h>

void calcular(int (*func)(), int a, int b){
    int res = func(a,b);
    printf("%d\n",res);
}

int main(int argc, char const *argv[]) {
    calcular(soma,2,5);
    calcular(prod,2,5);
}
```

Assim, quando invocamos a função **calcular**, devemos enviar a função (ponteiro) que desejamos que ela execute internamente.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# Function Pointers

- Observe que interessante...

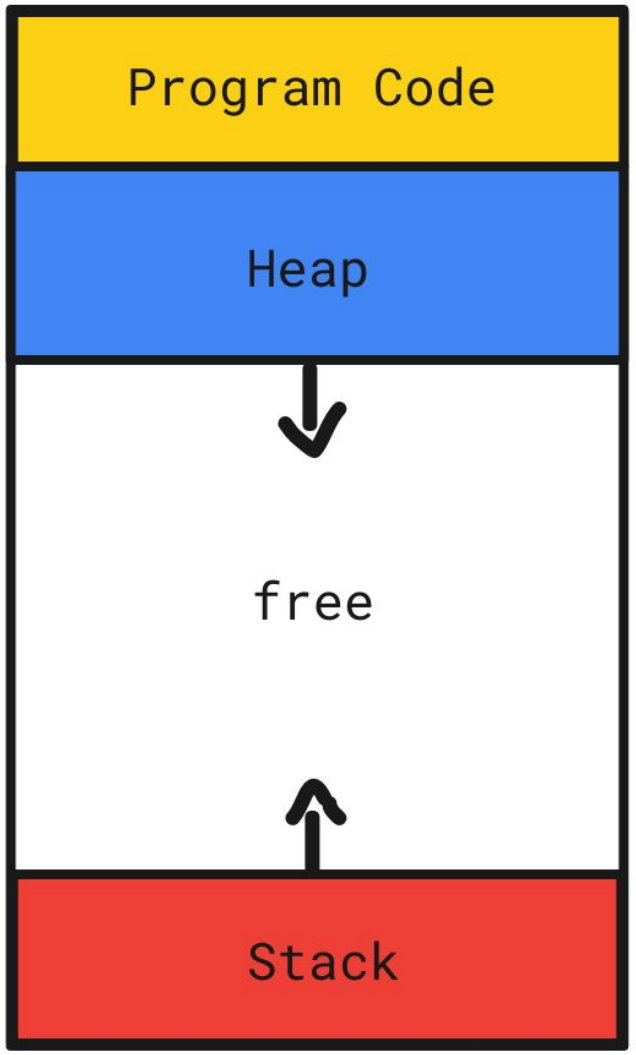
```
Terminal
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
Realizando Soma 2 + 5 == 7
Realizando Produto 2 x 5 == 10


------------------
(program exited with code: 0)
Press return to continue
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# Function Pointers

***Function Pointer***

(Ponteiro para Função) é um ponteiro (endereço de memória) que **faz referência a uma instrução executável**, em vez de dados, como nos ponteiros de variáveis já conhecidos.



# Function Pointers

- É um recurso muito útil para definir, **em tempo de execução**, **qual rotina que deverá executada**, sob supervisão de quem a invoca.

  *P.Ex.:* ***Callbacks, Plug-ins, OOP, etc…***

- **Função de Ordem Superior** é o nome dado às funções que recebem *function pointers* como argumento.

- Vamos ver mais um exemplo prático…

# Function Pointers

- Exemplo de ***Callback*** com ***Function Pointer***

```
void tarefa_demorada(void (*callback)()){
    //...
    //rotina demorada
    //...
    callback();
}

int main(){
    tarefa_demorada(send_Email);
    //tarefa_demorada(send_ZAP);
    //tarefa_demorada(ring_Alarm);
    //tarefa_demorada(do_Nothing);
}
```

# Function Pointers

- Exemplo de *Callback* com *Function Pointer*

```
void tarefa_demorada(void (*callback)()){
    //...
    //rotina demorada
    //...
    callback();
}

int main(){
    tarefa_demorada(send_Email);
    //tarefa_demorada(send_ZAP);
    //tarefa_demorada(ring_Alarm);
    //tarefa_demorada(do_Nothing);
}
```

**Função de Ordem Superior**, ou seja, uma função que recebe como argumento um ponteiro para função.

# Function Pointers

- Exemplo de *Callback* com *Function Pointer*

```
void tarefa_demorada(void (*callback)()){
    //...
    //rotina demorada
    //...
    callback();
}

int main(){
    tarefa_demorada(send_Email);
    //tarefa_demorada(send_ZAP);
    //tarefa_demorada(ring_Alarm);
    //tarefa_demorada(do_Nothing);
}
```

**Function Pointer.** Ponteiro para uma função que tem retorno `void`, e que não possui parâmetros. Internamente essa função se chamará **`callback()`**

# Function Pointers

- Exemplo de *Callback* com *Function Pointer*

```
void tarefa_demorada(void (*callback)()){
    //...
    //rotina demorada
    //...
    callback();
}

int main(){
    tarefa_demorada(send_Email);
    //tarefa_demorada(send_ZAP);
    //tarefa_demorada(ring_Alarm);
    //tarefa_demorada(do_Nothing);
}
```

**Invocação** da Function Pointer recebida como argumento.

# Function Pointers

- Exemplo de *Callback* com *Function Pointer*

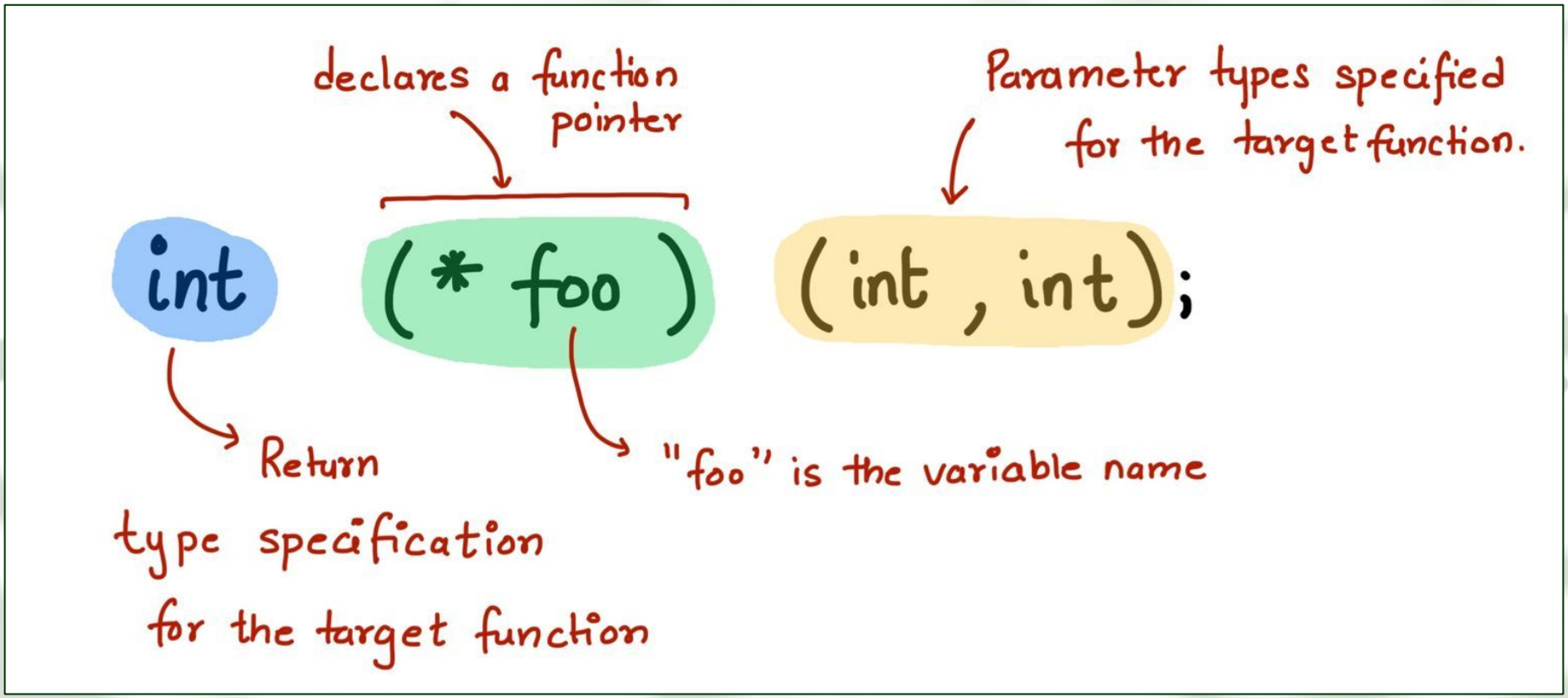
```
void tarefa_demorada(void (*callback)()){
    //...
    //rotina demorada
    //...
    callback();
}

int main(){
    tarefa_demorada(send_Email);
    //tarefa_demorada(send_ZAP);
    //tarefa_demorada(ring_Alarm);
    //tarefa_demorada(do_Nothing);
}
```

**Funções** que poderiam ser enviadas como argumento para a Function Pointer. Essas funções devem estar implementadas em algum lugar do código.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# Function Pointers

- Declaração de *Function Pointers*...



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# Function Pointers

- Declaração de *Function Pointers*...

```
tipo_retorno (*nome_funcao) ( )
```

- **Tipo de dado que a função retornará.**
- **Nome que a função assumirá internamente.**
- **Tipos dos Parâmetro(s) necessários para a execução da função.**

```
void ordem_superior(int(*func)())
void ordem_superior(void(*send)(char*), char* dest)
void ordem_superior((*Pessoa)(*get)(Pessoa[],int), int v)
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# Function Pointers

- Declaração de *Function Pointers*...

```
tipo_retorno (*nome_funcao) ( )
```

- **Tipo** de dado que a função retornará.
- **Nome** que a função assumirá internamente.
- **Tipos dos Parâmetro(s)** necessários para a execução da função.

```
void ordem_superior(int(*func)())
void ordem_superior(void(*send)(char*), char* dest)
void ordem_superior((*Pessoa)(*get)(Pessoa[],int), int v)
```

**ATENÇÃO! As funções enviadas como argumento devem respeitar a assinatura requerida pela Function Pointer!**

# O Poder das Function Pointers

- **Plugins**

```
void tarefa_demorada(void (*callback)()){
    //...
    //rotina demorada
    //...
    callback();
}

int main(){
    void (*alerts[3])() = {send_Email,send_ZAP,ring_Alarm};
    int option = input("Como deseja ser notificado?");
    tarefa_demorada(alerts[option]);
}
```

**Array de Function Pointers** para oferecer flexibilidade/customização ao código.

**Novos alertas podem ser facilmente integrados ao código (plug-ins)**

# O Poder das Function Pointers

```
typedef struct{
    char nome[100];
    int raça;
    int hp, xp;
    int (*atacar)(int);
    void (*andar)(char);
    int (*morrer)();
}Personagem;
```

**Function Pointers como membros de structs**

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# O Poder das Function Pointers

```
typedef struct{
    char nome[100];
    int raça;
    int hp, xp;
    int (*atacar)(int);
    void (*andar)(char);
    int (*morrer)();
}Personagem;
```

**Function Pointers como membros de structs**

```
int main(){
    Personagem* elfo = new_Person("Legolas",2,1000,80);
    //...
    elfo->andar('d');
    elfo->atacar(5);
}
```

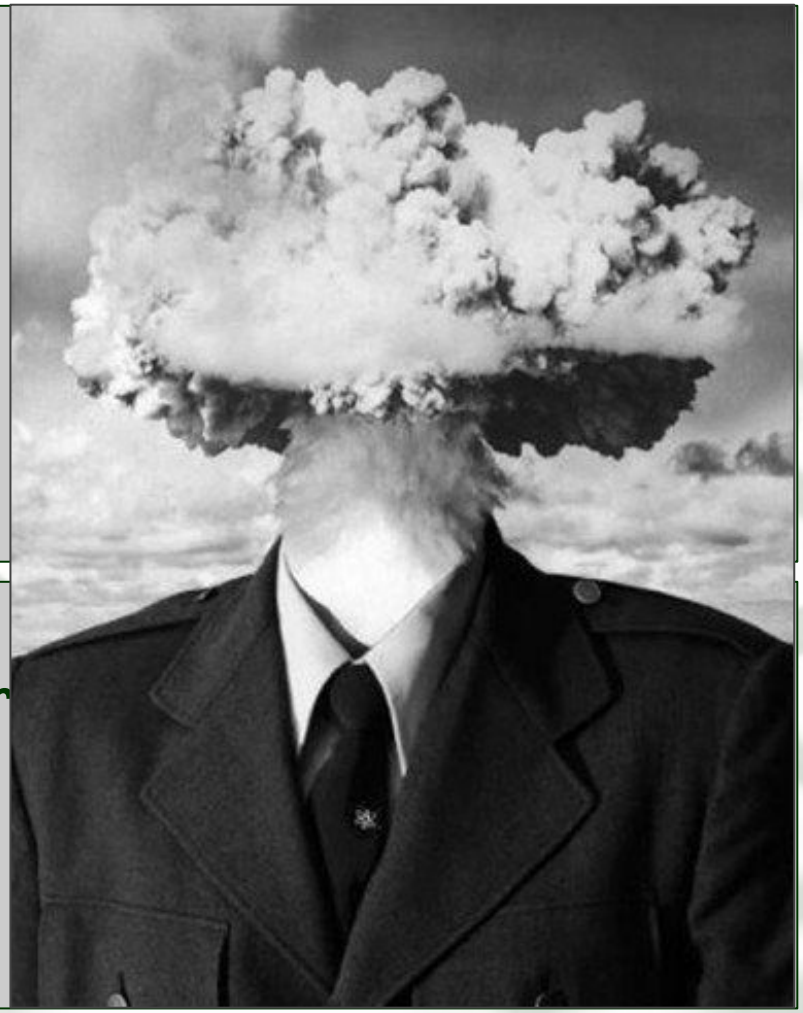
**Em que isso se parece???**

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# O Poder das Function Pointers

```
typedef struct{
    char nome[100];
    int raça;
    int hp, xp;
    int (*atacar)(int);
    void (*andar)(char);
    int (*morrer)();
}Personagem;
```

```
int main(){
    Personagem* elfo = new_Per
    //...
    elfo->andar('d');
    elfo->atacar(5);
}
```



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# O Poder das Function Pointers

```
#define new(TYPE) new_##TYPE()

typedef struct{
    char nome[100];
    int raça;
    int hp, xp;
    int (*atacar)(int);
    void (*andar)(char);
    int (*morrer)();
}_Personagem;

typedef struct _Personagem* Personagem;
```

Torna-se "transparente" o uso de Ponteiros

```
int main(){
    Personagem elfo = new(Personagem);
    elfo->set(elfo,"Legolas",2,1000,80);
    elfo->andar('d');
}
```

"Construtor" de um Objeto Genérico

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# O Poder das Function Pointers

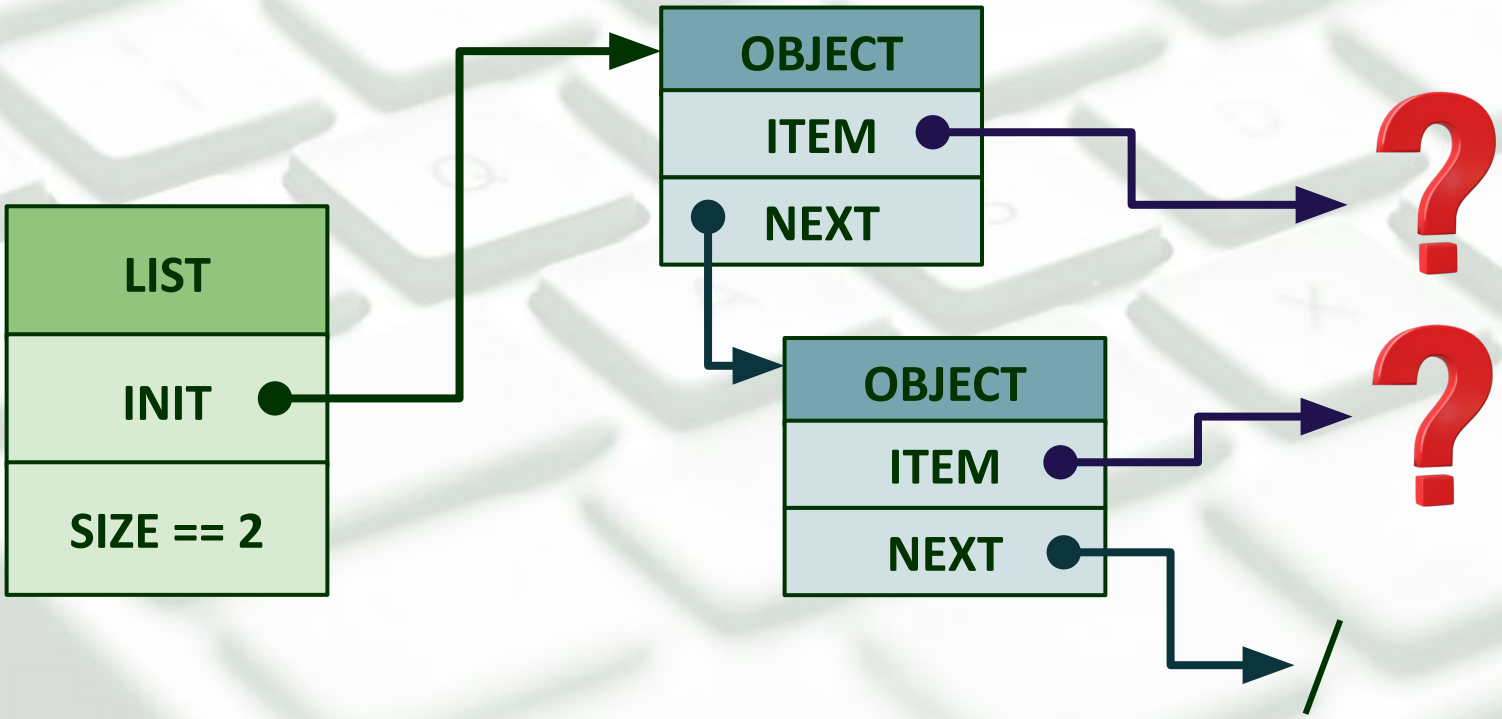
```
#define new(TYPE) new_##TYPE()

typedef struct{
    char nome[100];
    int raça;
    int hp, xp;
    int (*atacar)(int);
    void (*andar)(char);
    int (*morrer)();
}_Personagem;

typedef struct _Personagem* Per
```

```
int main(){
    Personagem elfo = new(Perso
    elfo->set(elfo,"Legolas",2,
    elfo->andar('d');
}
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# 2º Desafio

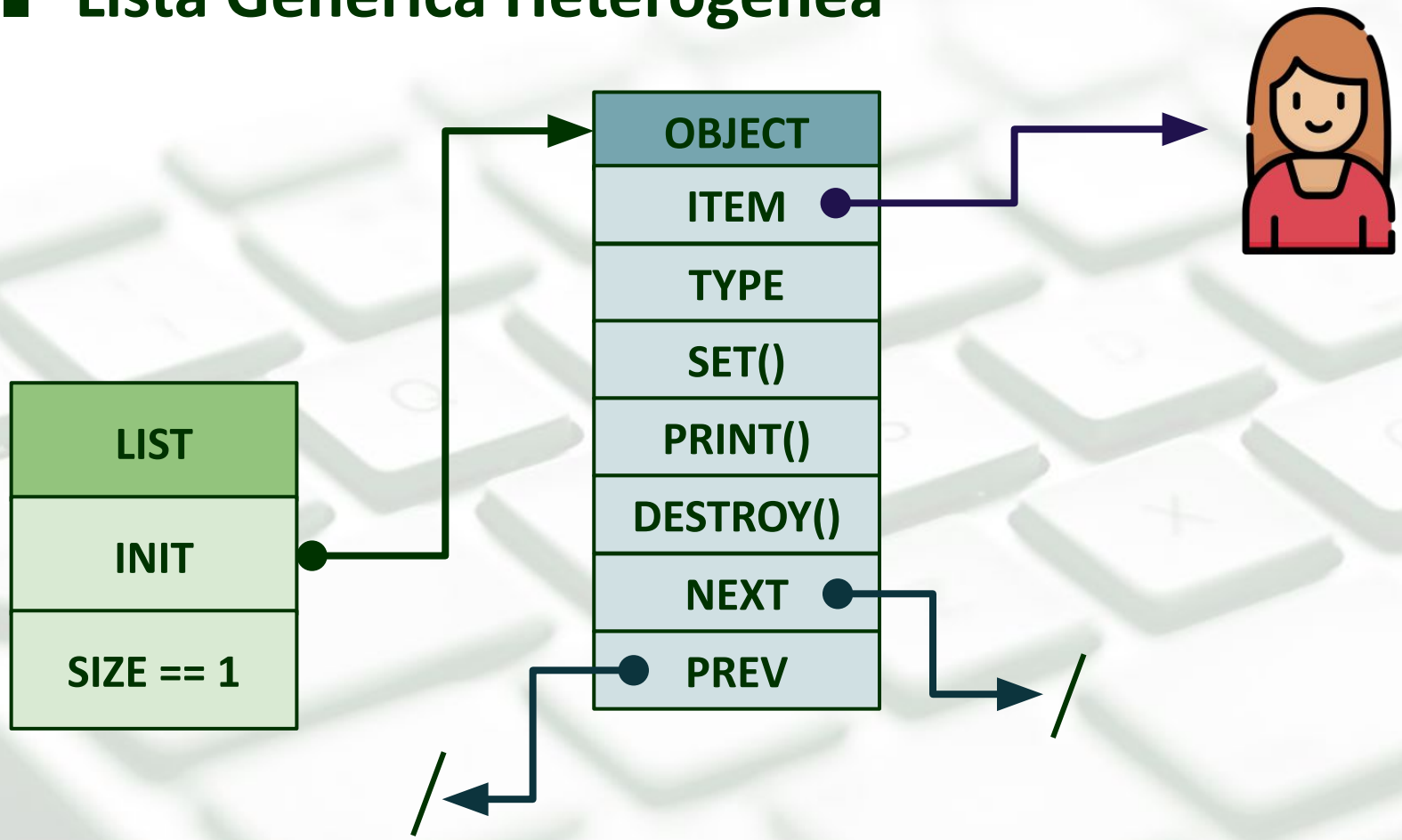
- Como os itens de uma Lista Genérica são "***indefinidos***", **como executar funções específicas *(p.ex. imprimir os dados do Nó)* para estes itens**?



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# Projeto de Implementação

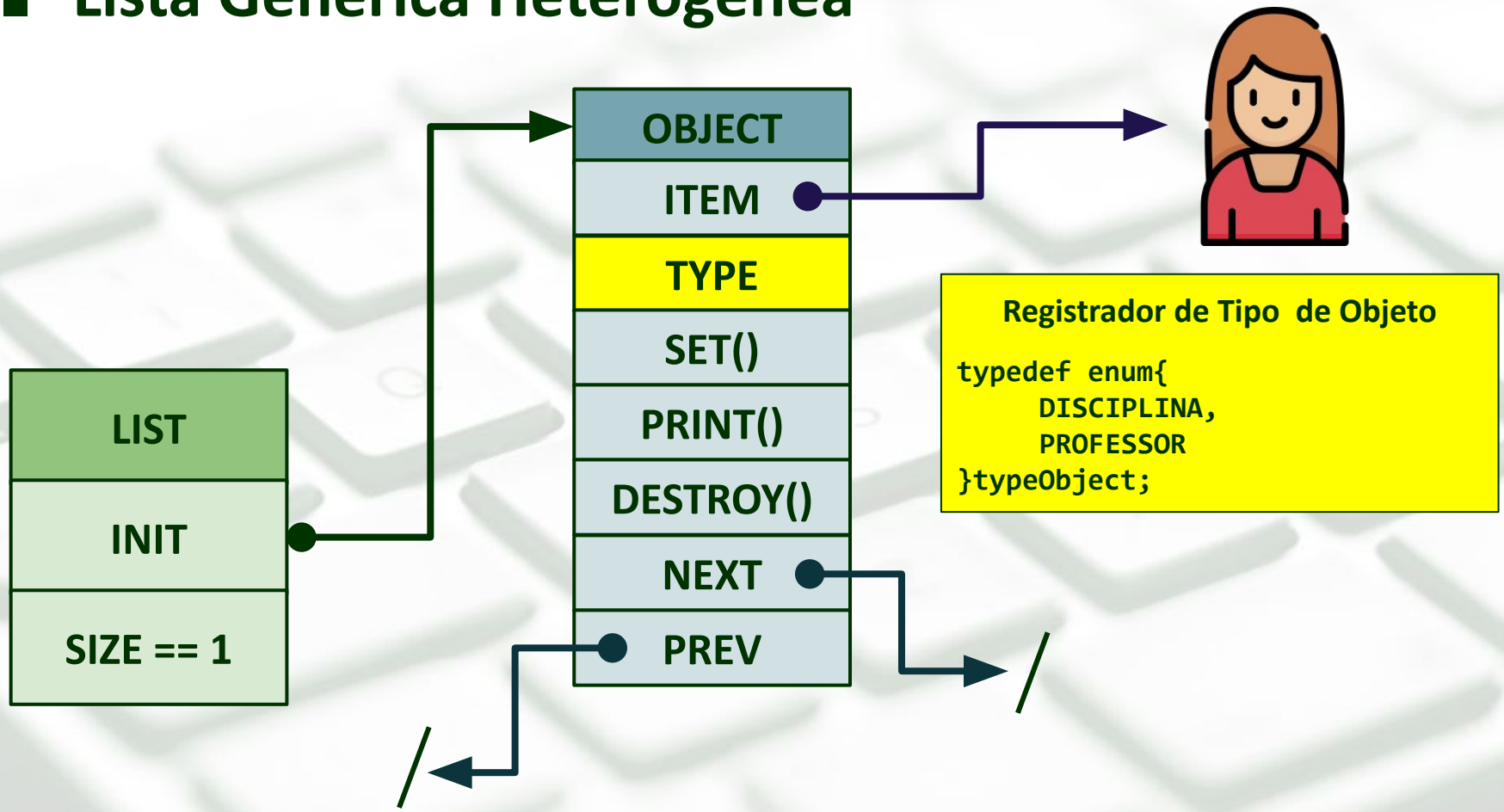
- **Lista Genérica Heterogênea**



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# Projeto de Implementação

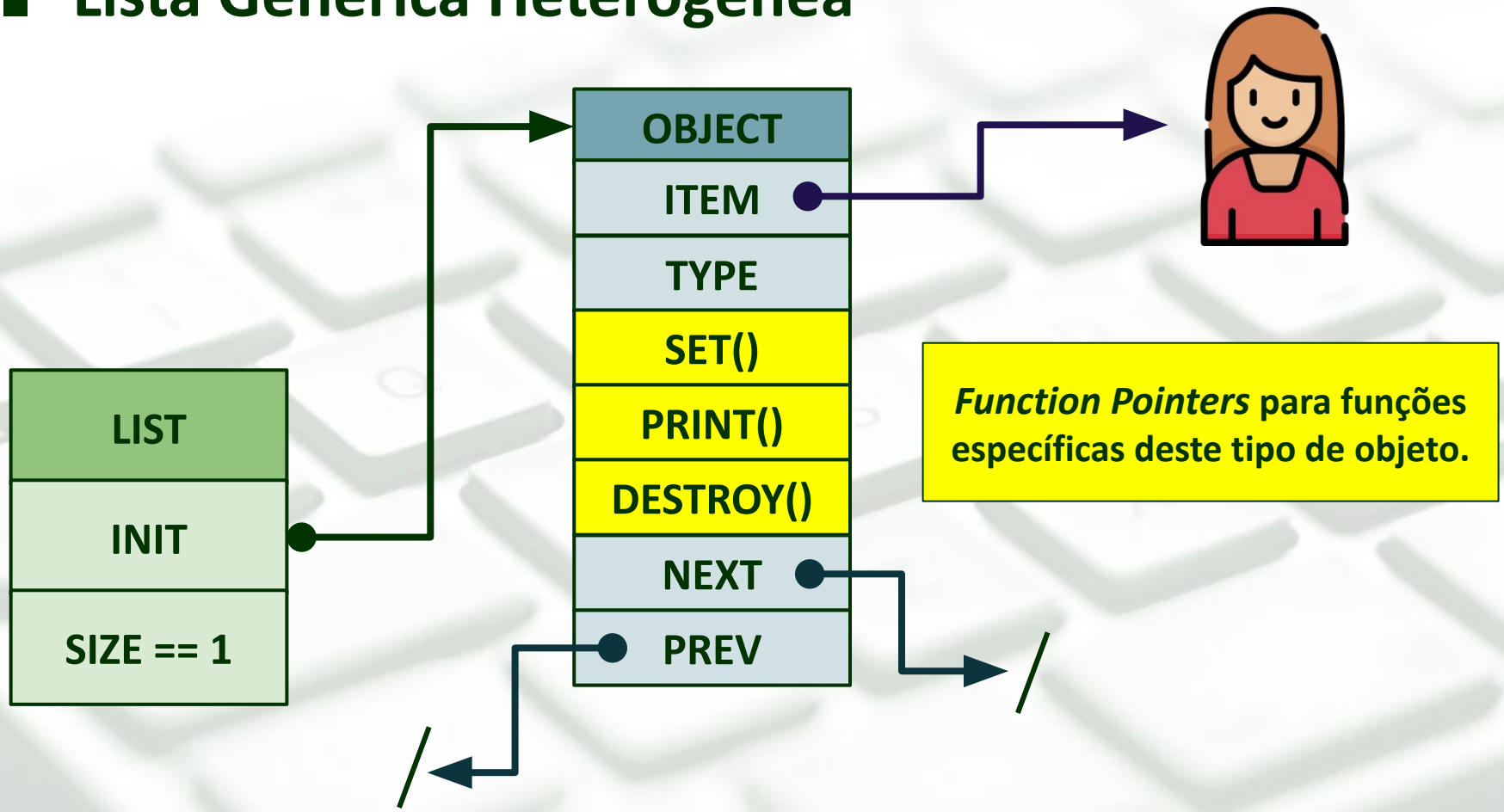
- **Lista Genérica Heterogênea**



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# Projeto de Implementação

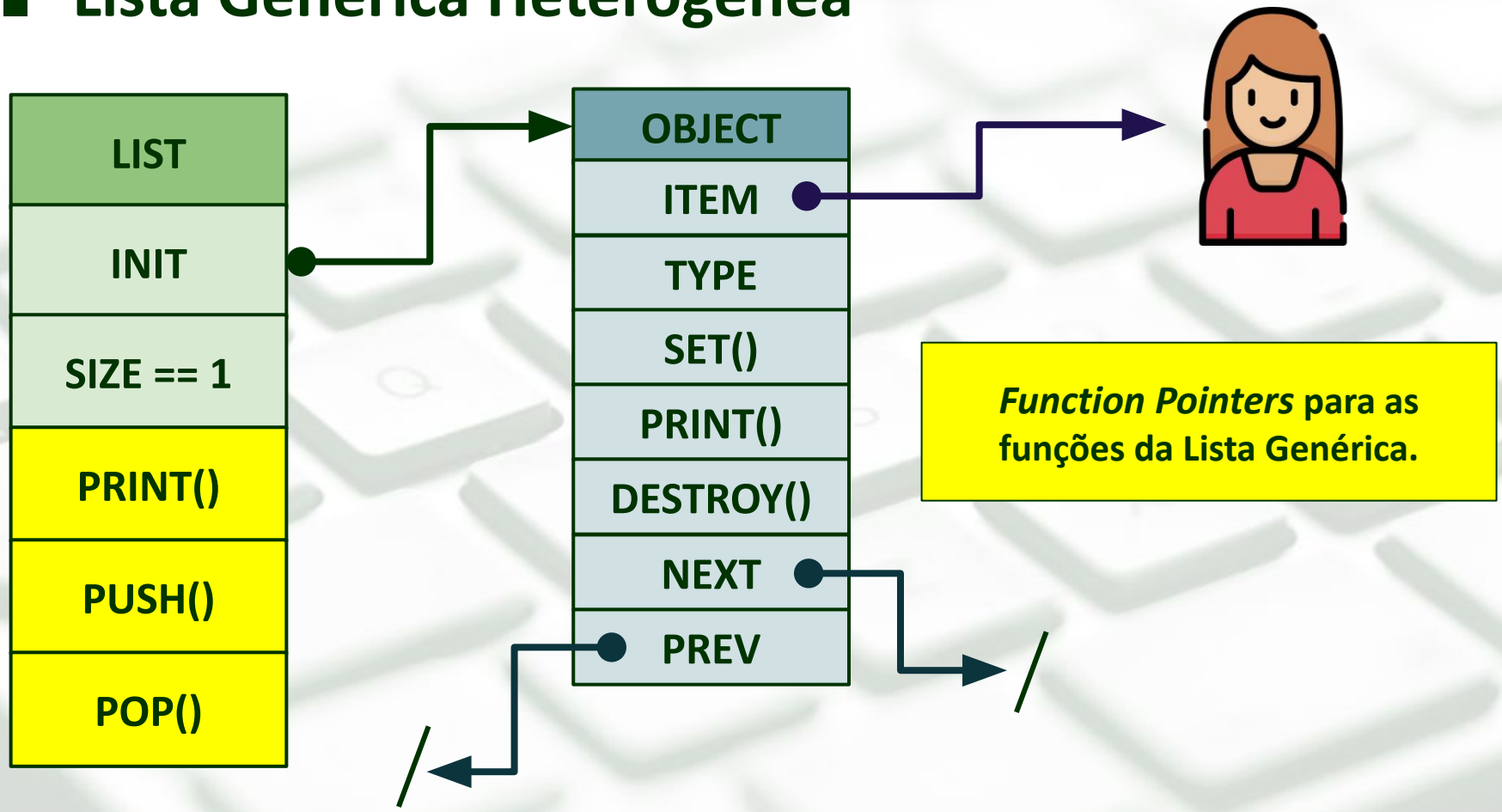
- **Lista Genérica Heterogênea**



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# Projeto de Implementação

- **Lista Genérica Heterogênea**



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# Bora CODAR!!!

## Lista de Objetos Genéricos v2.0

| | |
|---|---|
| `Object obj = new(?)`<br><Criar um novo Objeto Genérico> | `obj->destroy()`<br><Deletar Objeto da Memória> |
| `obj = set()`<br><Alimentar dados do Objeto> | `obj->print()`<br><Imprimir dados do Objeto> |
| `List lst = new(List)`<br><Criar uma Lista Vazia> | `lst->dequeue()`<br><Obter último Obj da Lista> |
| `lst->enqueue()`<br><Adicionar Obj ao Fim da Lista> | `lst->pop()`<br><Obter primeiro Obj da Lista> |
| `lst->push()`<br><Adicionar Obj ao Início da Lista> | `lst->get()`<br><Obter Obj específico (key)> |
| `lst->print()`<br><Listar todos Obj da Lista> | `lst->clear()`<br><Limpar/Excluir todos Obj da Lista> |

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

# Bora CODAR!!!

## Requisitos Técnicos...

- Implementação de ***Function Pointers*** nas Structs das Entidades.
- Tipagem dinâmica de objetos (mesmo nome, entidades distintas).
- Lista poderá operar com Entidades Heterogêneas

```
int main(){
     List lst = new(List);
     Object obj = new(Disciplina);
     setDisciplina(obj,"ED2",3,80);
     list_push(lst,obj);
     obj = new(Professor);
     setProfessor(obj,"Adriano","BSI");
     list_push(lst,obj);
     Object obj = list_pop(lst);
     printProfessor(obj);
     destroy(obj);
     list_print(lst);
     list_clear(lst);
}
```

```
int main(){
     List lst = new(List);
     Object obj = new(Disciplina);
     obj->set(obj,"ED2",3,80);
     lst->push(lst,obj);
     obj = new(Professor);
     obj->set(obj,"Adriano","BSI");
     lst->push(lst,obj);
     Object obj = lst->pop(lst);
     obj->print(obj);
     obj->destroy(obj);
     lst->print(lst);
     lst->clear(lst);
}
```