

# C “Avançado”

# Porque C?

- Equilíbrio entre controle e modernidade
- Estruturas de dados básicas não inclusas
- Extremamente popular
- Sintaxe base para muitas linguagens modernas

# “Pré-requisitos”

- Conseguir escrever um pequeno programa em C
- Funções, if/else, loops
- Entender o básico das diferenças de tipos
- Mas não quer dizer que eu não vou responder perguntas!

# Peculiaridades de C

- Compilada
- Tipos Estáticos
- “Portátil”
- Ponteiros

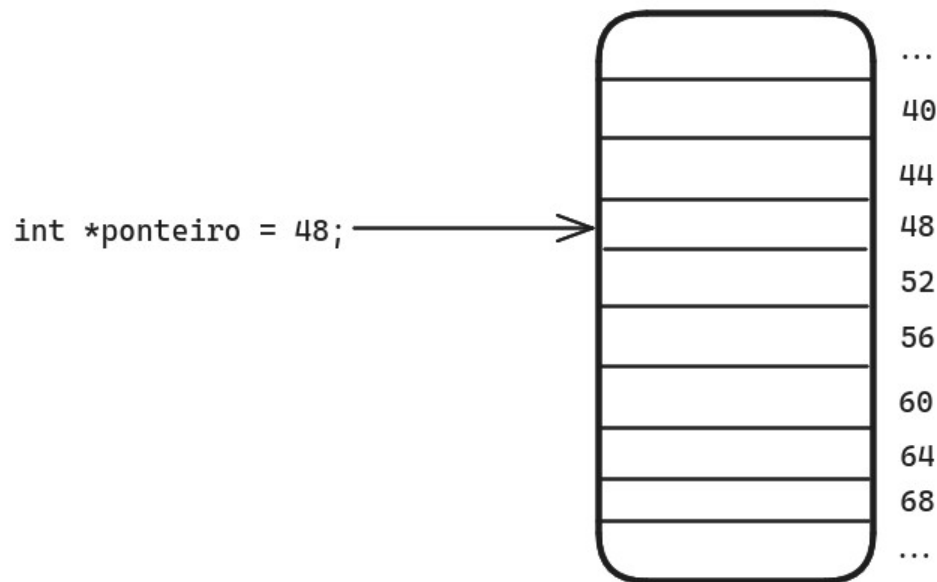
# “Avançado”?

- Vamos falar de 3 pontos importantes da linguagem
- Ponteiros
- Como usar o compilador
- Bibliotecas

# Ponteiros

- A funcionalidade mais controversa de C
- Tipo especial para “endereço de memória”
- Maior fonte de erros catastróficos
- Sintaxe complicada

# Conceito de Ponteiros



- “Índices” de memória
- Efetivamente, são números
- “Sabem” o tamanho do tipo
- Não são seguros

# Sintaxe de ponteiros

- Leia da direita para esquerda:
  - `float *apontador;`
    - Apontador de Float (*float pointer* em inglês)
    -
- `*` é o valor de onde ele aponta
  - `&` pega o endereço de uma variável
  - `a = &b; b == *a; // TRUE`
- Diferente na declaração
  - `int *a = &b; // o * é apenas indicador de tipo`



# Como evitar erros

- Para toda alocação, uma função equivalente de liberação
  - Construa funções construtoras e destrutoras
  - Lembre de esvaziar toda sua estrutura no destrutor
- Sempre inicialize os ponteiros
  - A linguagem C não checa se seus ponteiros são válidos
  - Segmentation Fault quando o programa tenta escrever onde não pode

# Bibliotecas

- Arquivos de código autocontido
- Encapsula o funcionamento interno de funções complicadas
- Pode ser redistribuído sem o código original
- Pode ser usada por vários programas
- Exemplos: `stdio`, `stdlib`, `math`, `vector` (C++)

# Ligação

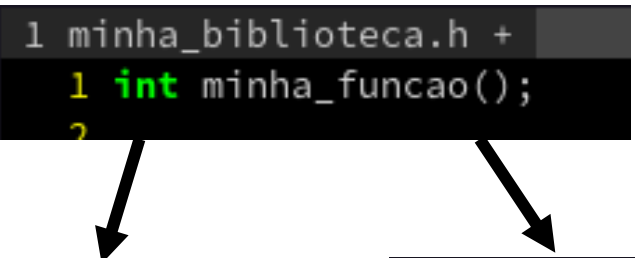
- A biblioteca é “ligada” ao seu código
  - Isso pode acontecer durante a compilação (estática)
  - Ou pode ser ligada durante a execução (dinâmica)
- As bibliotecas padrão mencionadas são todas dinâmicas
  - Os arquivos estão em alguma pasta do seu sistema operacional
- As que vocês farão em ED são estáticas

# Como fazer uma biblioteca (estática)

- Basta criar um arquivo de código (.c) sem uma função main
- Para expor funções, você vai precisar declarar elas
  - Para isso serve o arquivo de cabeçalho (.h)
  - Coloque lá as declarações de funções que outros podem usar
- Quando for usar a biblioteca, inclua esse cabeçalho
- “Compile” esses dois arquivos de código (.c) juntos

# Um exemplo

```
1 minha_biblioteca.h +
1 int minha_funcao();
2
```



```
1 NERD_tree_1 -
1 #include "minha_biblioteca.h"
2 int main(){
3     minha_funcao();
4 }
```

```
1 minha_biblioteca.c
1 #include "minha_biblioteca.h"
2 int minha_funcao(){
3     return 1;
4 }
```

```
gcc main.c minha_biblioteca.c -o test_biblioteca
```

# O Compilador

- “Traduz” o seu código de humano para código de máquina
- Não é uma tradução 1:1
  - Pode, por exemplo, realizar otimizações ou eliminar código “inútil”
- Não é muito bem padronizado
- Possui muitas “flags” de configuração

# Compilador vs Ligador

- O seu compilador (gcc) provavelmente faz os dois
  - Se você quiser usar apenas o ligador, existe o `ld`
- O compilador na verdade apenas gera o binário para **1** arquivo
  - O ligador une todos esses binários em **executável**
- Para apenas compilar, usa-se a flag `-c`

```
gcc -c main.c -o main.o
```

# Porque separar os processo?

- Projetos muito grandes demoram para compilar
  - Partes sem modificação não precisam ser recompiladas
- Facilita redistribuição
- Interage bem com o programa Make
- Principalmente por motivos técnicos



# Flags úteis do compilador

- Compiladores modernos tem várias flags muito úteis
- Flags de aviso geralmente começam com -W
  - -Wall gera vários avisos para possíveis erros
  - -Wextra gera ainda mais avisos
- Flags de otimização (demoram mais para compilar)
  - -O1, -O2, -O3
- -std força o compilador a usar um padrão mais moderno de C
  - -std=c99/ -std=c11/ -std=c17

# Fim

# Fim