Alguns Tópicos de Linguagem C

(parte I)

O que é a linguagem C

- Criada nos anos 70.
- Desenvolvida inicialmente para o Sistema Operativo Unix.
- Uma das linguagens de programação mais populares e influentes.

Por que aprender C?

- Base para outras linguagens como C++, C#, Java e Python.
- Manipulação direta de memória e otimização de recursos.
- Fundamental em áreas como: sistemas operativos, sistemas embarcados e outros.

Características da Linguagem C

- Linguagem de baixo nível, com manipulação direta da memória
- Eficiência em termos de desempenho
- Portabilidade: os programas escritos em C podem ser executados em diferentes sistemas operativos
- Linguagem estruturada com controle de fluxo (if/else, for/while, etc)
- Versatilidade: usada em sistemas operativos, compiladores e outros.

C x C++

```c
#include <stdio.h>

int main() {
    printf("Hello World!");
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int main() {
  cout << "Hello world!\n";
  return 0;
}
```

C x C++

```c
#include <stdio.h>

int main() {
    printf("Hello World!");
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int main() {
  cout << "Hello world!\n";
  return 0;
}
```

C x C++

```c
#include <stdio.h>

int main() {
    printf("Hello World!");
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int main() {
  cout << "Hello world!\n";
  return 0;
}
```

C x C++

```c
#include <stdio.h>

int main() {
    printf("Hello World!");
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int main() {
  cout << "Hello world!\n";
  return 0;
}
```

Compilação

```
$ g++ hello.cpp
$ ./a.out
Hello world!
```

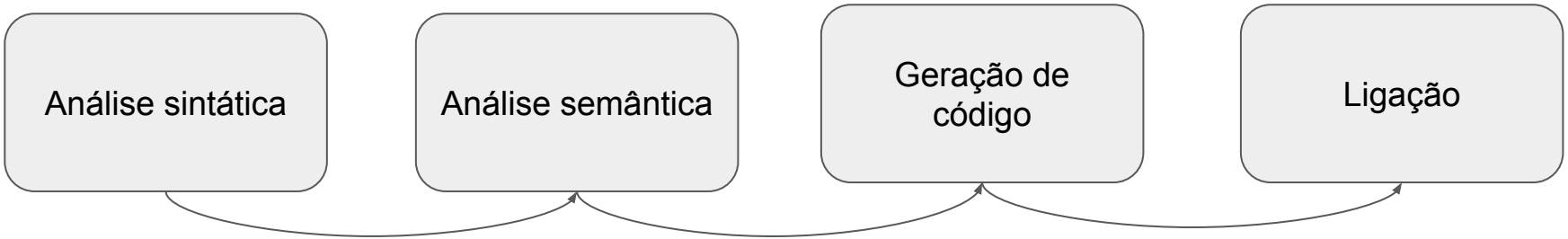
```
$ gcc hello.c
$ ./a.out
Hello world!
```

Compilação

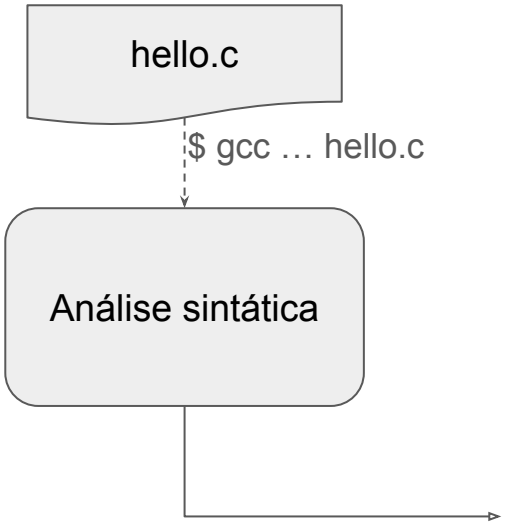
```
$ g++ -Wall -o hello hello.cpp
$ ./hello
Hello world!
```

```
$ gcc -Wall -o hello hello.c
$ ./hello
Hello world!
```

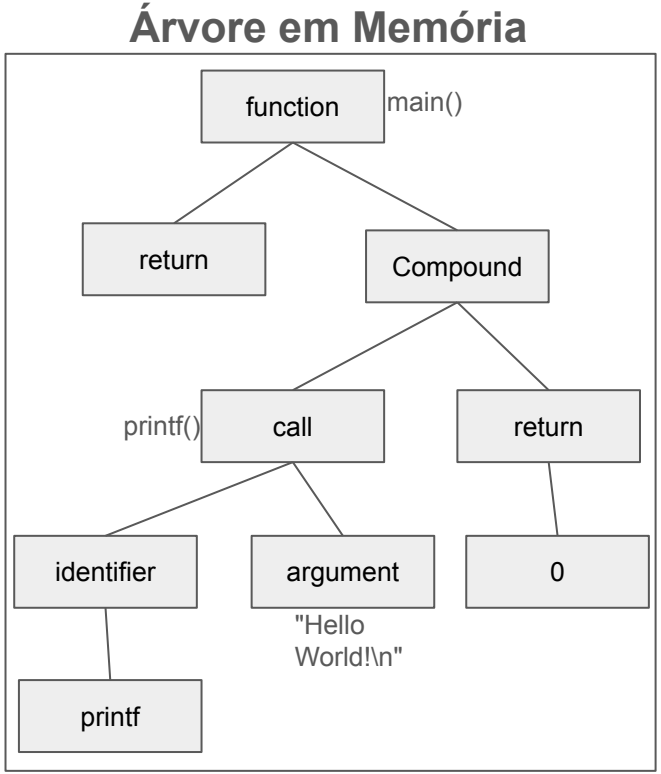
Compilação


Análise sintática
Análise semântica
Geração de código
Ligação

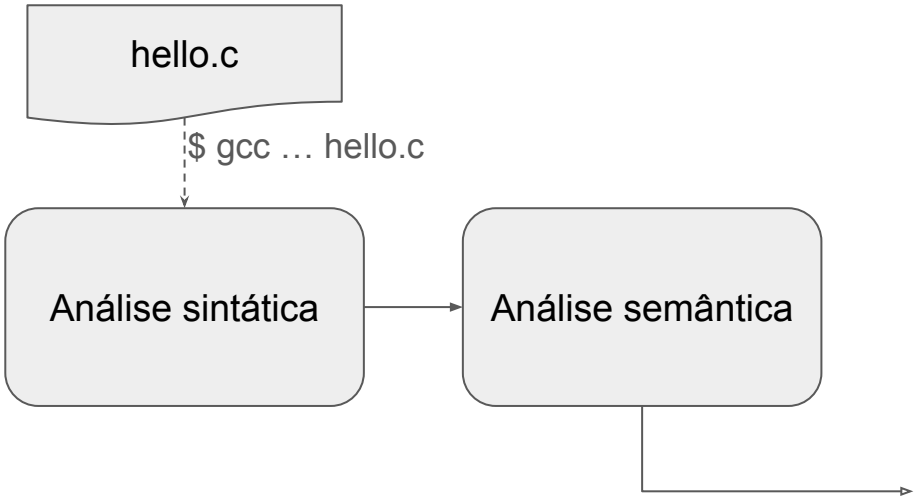
Compilação



Verifica se o programa está escrito segundo a **gramática** da linguagem C. Neste estágio, o programa é representado como uma **árvore em memória**.

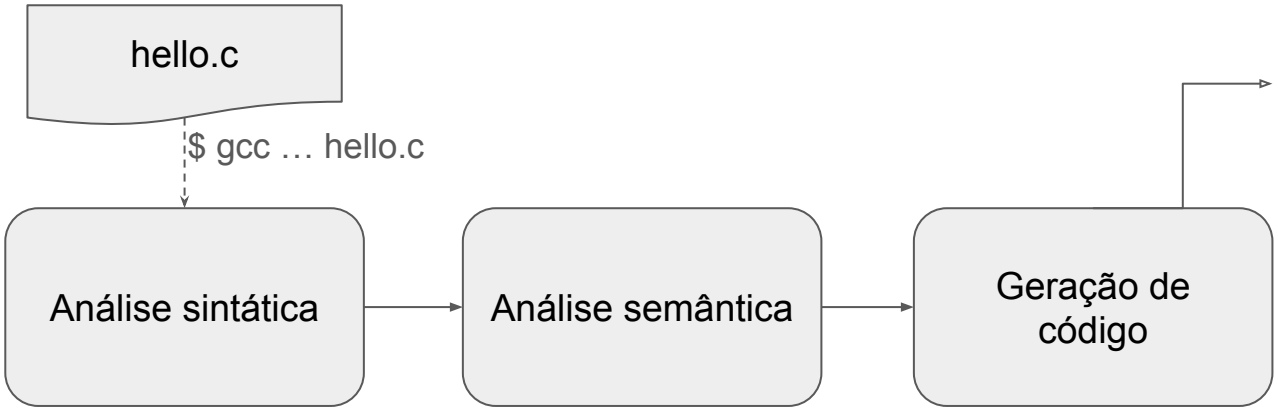


Compilação



Verifica se o programa é bem formado (se os tipos estão corretos) e recolhe informações para a geração de código. O programa é representado como uma árvore com anotações em memória.

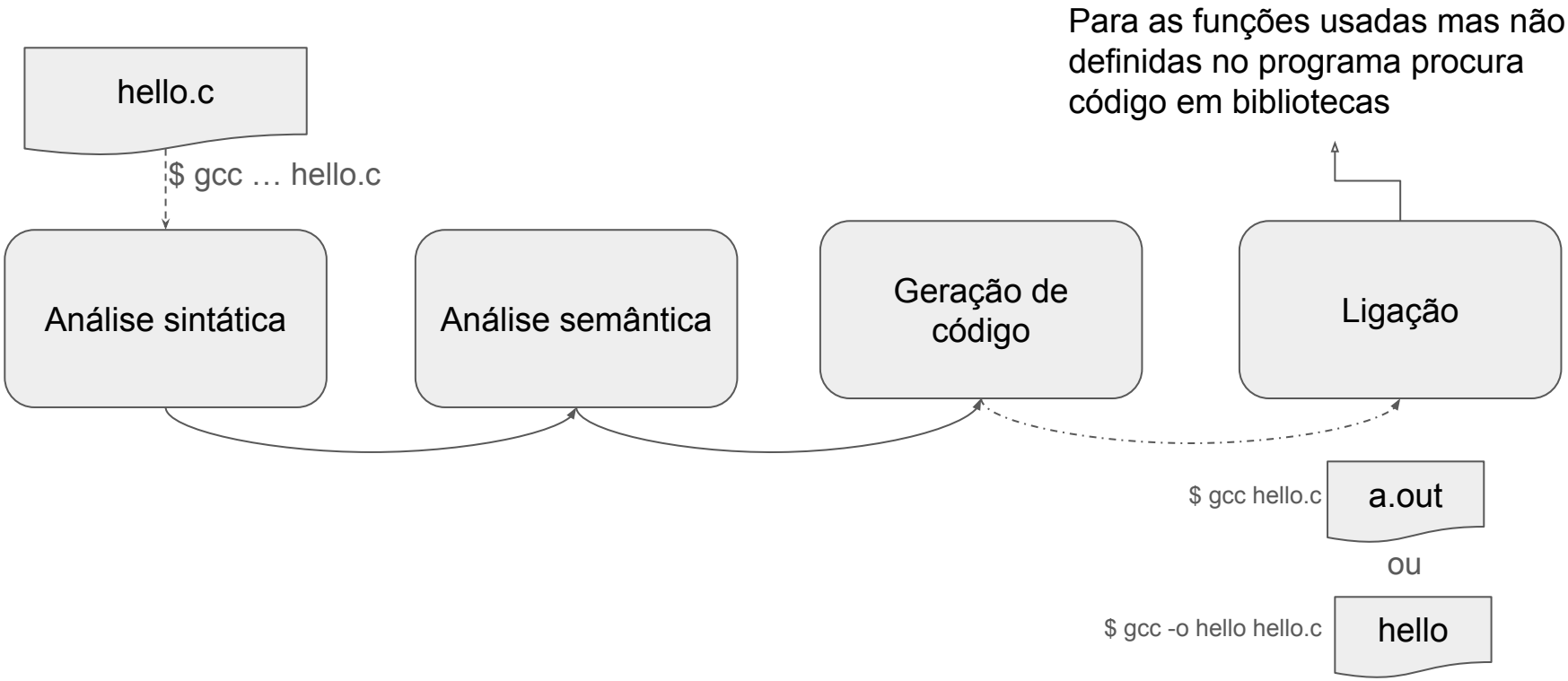
Compilação



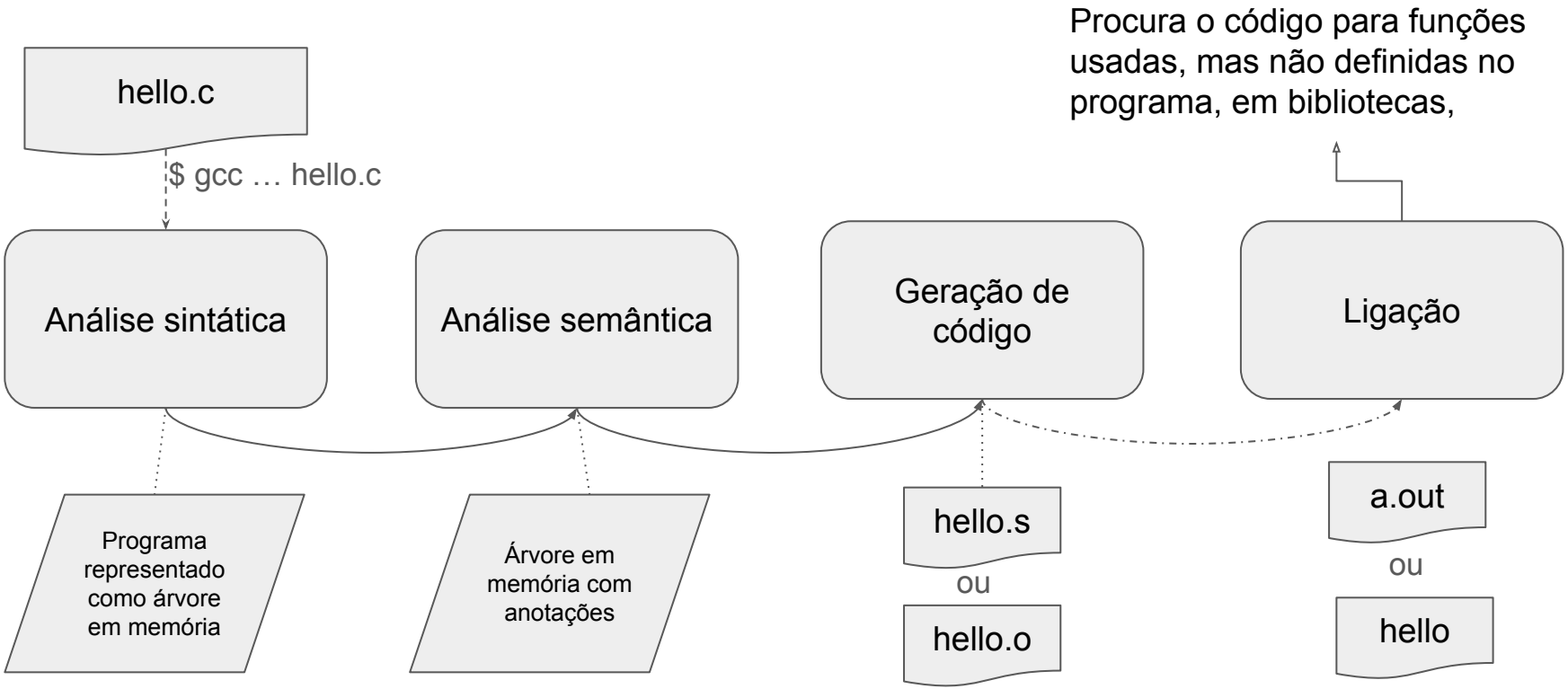
$ gcc -S hello.c ——— Para aqui e gera assembly `hello.s`

$ gcc -c hello.c ——— Para aqui e gera binário `hello.o`

Compilação


Para as funções usadas mas não definidas no programa procura código em bibliotecas
hello.c
$ gcc … hello.c
Análise sintática
Análise semântica
Geração de código
Ligação
$ gcc hello.c
a.out
ou
$ gcc -o hello hello.c
hello

Compilação


Procura o código para funções usadas, mas não definidas no programa, em bibliotecas,
hello.c
$ gcc … hello.c
Análise sintática
Análise semântica
Geração de código
Ligação
Programa representado como árvore em memória
Árvore em memória com anotações
hello.s
ou
hello.o
a.out
ou
hello

Debugger

$ gcc **-g** -o hello hello.c

incluir informações de depuração no executável, necessárias para o GDB (depurador) entender o código-fonte, permitindo a depuração em nível de linha de código (e não apenas em código binário).

Degugger

$ gcc **-g** -o hello hello.c

$ **gdb** hello

Inicia o depurador (*debugger*)

gdb> break main

Define um ponto de interrupção (breakpoint) na função `main`. Um breakpoint é um local no código onde a execução será pausada.

gdb> run

O programa executará até chegar à primeira linha de `main`

gdb> next

Faz o programa avançar para a próxima linha no código-fonte.

gdb> …

gdb> quit

quit ou *q* ou *CTRL+D* encerra a sessão de depuração e retorna ao terminal do Linux.

Degugger (lldb - macOs)

$ gcc **-g** -o hello hello.c

$ lldb ./hello

gdb> break set –name main

gdb> run

gdb> next

gdb> exit

$

Bibliotecas

- Uma biblioteca em C é um conjunto de funções e recursos pré-compilados que podem ser reutilizados em programas.
- Encapsulam código que realiza tarefas específicas, como operações matemáticas ou entrada/saída de dados, permitindo que os programadores utilizem essas funções sem precisar reescrevê-las.
- São compiladas separadamente do programa principal e são vinculadas (ou "linkadas") ao código durante a fase de compilação ou execução.

Exemplos:

➔ libm: Contém funções matemáticas como sin(), cos(), sqrt(), entre outras.
➔ Standard I/O Library: funções de entrada e saída.

Cabeçalhos

- Um cabeçalho em C é um ficheiro com extensão **.h** que contém declarações de funções (mas não sua implementação), macros, tipos de dados e constantes.
- Age como uma **interface** entre o código do usuário e as bibliotecas, permitindo que o compilador saiba sobre as funções e variáveis antes de usá-las.

Exemplos:

➔ `<math.h>`
➔ `<stdio.h>`

Bibliotecas

seno.c

```
#include <stdio.h>

int main() {
    double n, seno;

    n = 0.7854; // corresponde a 45 graus
    seno = sin(n);

    printf("sin(45) = %f\n", seno);

    return 0;
}
```

Bibliotecas

seno.c

```
#include <stdio.h>

int main() {
    double n, seno;

    n = 0.7854; // corresponde
a 45°

    seno = sin(n);

    printf("sin(45) = %f\n",
seno);

    return 0;
}
```

Bibliotecas

seno.c

```
#include <stdio.h>

int main() {
    double n, seno;

    n = 0.7854; // corresponde
a 45°

    seno = sin(n);

    printf("sin(45) = %f\n",
seno);

    return 0;
}
```

$ gcc -Wall -o seno seno.c

seno.c:7:12: error: call to **undeclared library function 'sin'** with type 'double (double)'; ISO C99 and later do not support implicit function declarations [-Wimplicit-function-declaration]

```
      seno = sin(n);
      ^
```

seno.c:7:12: note: **include the header <math.h> or explicitly provide a declaration for 'sin'**

Bibliotecas

seno.c

```
#include <stdio.h>
#include <math.h>

int main() {
    double n, seno;

    n = 0.7854; // corresponde a
45°

    seno = sin(n);

    printf("sin(45) = %f\n",
seno);

    return 0;
}
```

$ gcc -Wall -o seno seno.c
undefined reference to `sin'

Bibliotecas

seno.c

```
#include <stdio.h>
#include <math.h>

int main() {
    double n, seno;

    n = 0.7854; // corresponde a
45°

    seno = sin(n);

    printf("sin(45) = %f\n",
seno);

    return 0;
}
```

```
$ gcc -Wall -o seno seno.c
```

undefined reference to `sin'

```
$ gcc -Wall -o seno seno.c -lm
```

Bibliotecas

seno.c

```
#include <stdio.h>
#include <math.h>

int main() {
    double n, seno;

    n = 0.7854; // corresponde a
45°

    seno = sin(n);

    printf("sin(45) = %f\n",
seno);

    return 0;
}
```

```
$ gcc -Wall -o seno seno.c
undefined reference to `sin'
$ gcc -Wall -o seno seno.c -lm
```

Informa o compilador para **linkar (-l)** com a biblioteca matemática **libm (m)**

Endereços

- A memória RAM (= random access memory) de qualquer computador é uma sequência de bytes. A posição (0, 1, 2, 3, etc.) que um byte ocupa na sequência é o endereço (= address) do byte.
- Cada variável de um programa ocupa um certo número de bytes consecutivos na memória do computador. Uma variável do tipo **char ocupa 1 byte**. Uma variável do tipo **int ocupa 4 bytes** e um **double ocupa 8 bytes** em muitos computadores. O número exato de bytes de uma variável é dado pelo operador `sizeof`. A expressão `sizeof (char)`, por exemplo, vale 1 e a expressão `sizeof (int)` vale 4.
- O endereço de uma variável é dado pelo operador `&`. Assim, se `i` é uma variável então `&i` é o seu endereço.

Apontadores

- Um apontador (= ponteiro = pointer) é um tipo especial de variável que armazena um endereço.
- Um apontador pode ter o valor NULL, que é um endereço "inválido".
- Há vários tipos de apontadores: para inteiros, para apontadores para inteiros, e outros. O computador precisa saber o tipo do apontador. Para declarar um apontador `p` para um inteiro, escreva:

```
int *p;  ou  int* p;
```

Alocação Dinâmica de Memória

- Gestão de memória dinâmica em C é feita através de um conjunto de funções na biblioteca standard cujo cabeçalho é `stdlib.h`
  - `malloc`
  - `free`
  - `realloc`

Alocação Malloc

`malloc` retorna um apontador para um espaço de memória que podemos utilizar da forma que pretendermos.

```
void *malloc(int size)
```

Como utilizar:

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

main()
{
   int *p;
   p = (int*)malloc(sizeof(int));
   *p = 10;
   printf("%d\n", *p);
}
```

Free

`free` liberta o espaço de memória apontado pelo apontador `p`

```
void free(void *p)
```

Como utilizar:

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

main()
{
   int *p;
   p = (int*)malloc(sizeof(int));
   *p = 10;
   printf("%d\n", *p);
   free(p);
}
```

Stack e Memória Heap

- Memória local (stack): As variáveis locais de uma função são armazenadas na pilha, e essa área de memória é automaticamente liberada quando a função termina. Retornar um apontador para essa memória resulta em um dangling pointer (ponteiro pendente), que aponta para uma região de memória não válida.
- Memória dinâmica (heap): A memória alocada com `malloc` é armazenada no **heap**, e essa memória permanece alocada até que seja liberada explicitamente com `free`. Assim, o apontador retornado pela função continua válido enquanto a memória não for liberada.

Alocação Dinâmica de Memória