FUNCIONAMENTO DA MEMÓRIA DO PROGRAMA

INTRODUÇÃO

Antes de trabalharmos com ponteiros, é essencial entendermos como funciona a memória de um programa, já que os ponteiros trabalham exatamente com isso. Então, vamos a alguns dados importantes:

BITS E BYTES

Um único bit assume dois valores: 0 ou 1.
1 byte é composto por 8 bits, assumindo valores de 0 a 255 (2^8). A partir disso, por combinatória, temos que 2 bytes resulta em 65.536 valores possíveis ($2^8 \cdot 2^8$ ou 256^2), 4 bytes resulta em 256^4 valores possíveis, etc.

TAMANHO DE CADA TIPO DE DADO

Cada tipo de dado ocupa um espaço específico na memória. Veja o código abaixo:

```c
int main() {
  char c; // Ocupa 1 byte na memória.
  int n; // Ocupa 4 bytes na memória.
  float f; // Ocupa 4 bytes na memória
  double d; // Ocupa 8 bytes na memória
}
```

Dessa forma, um `char` armazena até 256 caracteres diferentes (lembra da tabela ASCII?), um `int` valores de -2^{31} até $2^{31} - 1$, e por aí vai.

ENDEREÇO-BASE

Denominamos **endereço-base** o endereço inicial do espaço reservado para a variável na memória. Ou seja, se um `int` ocupa 4 bytes na memória, armazenados nos endereços `0xabc123`, `0xabc124`, `0xabc125`, `0xabc126`, o endereço-base vai ser o `0xabc123`.
Para saber o endereço-base de uma variável, basta usar o operador &, responsável por imprimir o endereço de memória e escrever `cout << &variavel`.

TIPOS DE MEMÓRIA

A `stack` e `heap` são as principais **áreas da memória do programa** nas quais vamos focar.

STACK

- Na `stack` estão os dados **estáticos** do seu programa, como variáveis locais e funções, os quais você sabe o tamanho. A `stack` tem área máxima de 8mb (aproximadamente), o que é pouco para uma aplicação grande;
- É gerenciada pelo sistema operacional de forma automática;
- É organizada de forma **linear** na memória, então os dados estão dispostos de forma consecutiva um do outro.

Ao executar o programa, a `stack` é criada na memória principal (RAM), sendo carregada com as declarações das funções e as variáveis da função `main`. No decorrer do programa, mais dados podem ser armazenados ou retirados da `stack` conforme a necessidade: ao entrar em uma função, uma região na `stack` é alocada para ela (`stack frame`) de forma que ao encerrar a função, todos os dados da `stack frame` criados nela são apagados automaticamente. Por isso não é permitido, no C++, que uma função retorne um vetor diretamente, já que o vetor seria nulo.

Um exemplo de como a `stack` cresce e decresce:

```
void func1() {
  int x = 10;
}
void func2() {
  int x = 20;
  func1();
}
int main() {
  func2();
  return 0;
}
```

Note que, quando cada função termina de executar, sua `stack frame` correspondente é apagada.

Para mexer com um volume de dados maior e fazer com que esses vetores e dados importantes persistam através das funções, precisamos da heap!

HEAP

- Na heap estão os dados **dinâmicos** do seu programa, ou seja, dados que serão criados durante a execução do programa e você não sabe o tamanho. A heap tem tamanho limitado somente pela quantidade da RAM disponível, ou seja, ultrapassa mais de 1gb de espaço fácil.
- É gerenciada por você!!! (e aí mora o problema)
- É organizada de forma **espalhada** na sua RAM, permitindo alocação e liberação de memória de forma fácil.

Pelo fato do programa acessar somente uma heap durante toda sua execução, torna-se mais fácil de manter os dados e evitar que eles sejam apagados ao fim de uma função. Contudo, por ser uma estrutura com dados espalhados, torna-se mais lento de capturá-los e isso reduz a performance do programa se o volume de dados for muito grande. Por fim, deve-se ter cautela ao manipular a heap corretamente.

PROBLEMAS FAMOSOS

- Stack Overflow: ocorre quando a `stack` estoura seu tamanho máximo, devido a uma recursão muito profunda, loops infinitos e etc.
- Memory Leak: ocorre quando a memória alocada na `heap` nunca é apagada, impedindo que o espaço seja usado novamente e o programa acaba consumindo recursos desnecessários no computador.

HEAP VS STACK

	Heap	Stack
Gerenciamento	Manual/Programador	Sistema Operacional
Tamanho	Grande	Pequeno
Tempo de vida	Enquanto não for liberado	Sair do escopo da função
Utilidade	Dados grandes e alocação dinâmica	Variáveis locais e chamadas de funções

Tá, entendi tudo! E agora, como faço pra mexer com ponteiros?

Speaker notes