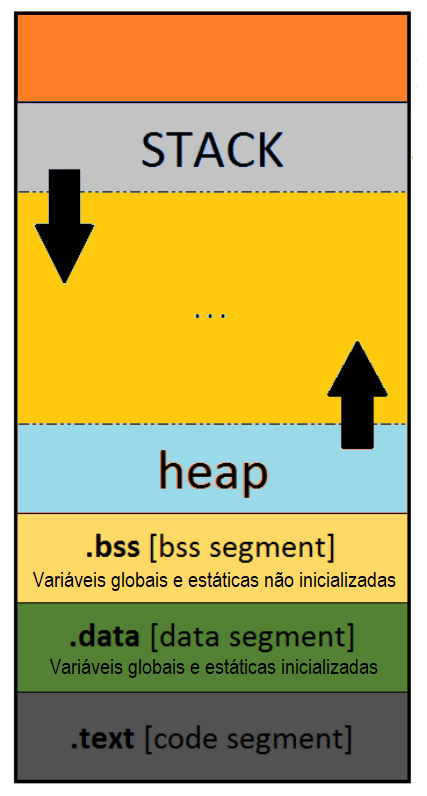
**Artigo: Quais as regiões do espaço de endereçamento dos programas?**

A divisão de memória para um processo de uma aplicação é mais complexa. Estamos usando stack e heap para simplificar.

O sistema operacional ao carregar um programa(.exe) na memória disponibiliza ao programa um espaço de endereçamento. Esse espaço é a memória disponível para aquele programa.Existe no geral a seguinte divisão deste espaço de endereçamento de um programa(ou seus processos)



**1. Segmento de código (Code Segment) - TEXT**

O segmento de código, também conhecido como segmento de texto, é a área da memória que contém o código executável(binário) de seu programa. Ou seja, é onde está armazenado as instruções de seu programa. O segmento de código geralmente é somente leitura para evitar o riscos de ataques como buffer overflow, sobreposição de memória etc. O segmento de código **não contém** variáveis ​​de programa como variável local, variáveis ​​globais etc. Contém apenas  no geral o código executável do programa que está sendo executado.

Depois temos outra região de memória chamada de **Segmento de dados**: onde são armazenados as variáveis ​​globais e variáveis ​​estáticas de dados. Não é uma memória somente leitura e é dividida em .bss(**Segmento de dados não inicializado)** e .data(**Segmento de dados inicializado)**

**2. Segmento de dados não inicializados(bss** ou Block Started by Symbol ou bloco iniciado pelo símbolo)

Este segmento também é conhecido como **bss** .Esta é a parte da memória que contém:**Variáveis ​​globais não inicializadas*(incluindo variáveis ​​de ponteiro)*, Variáveis ​​globais constantes não inicializadas** .**Variáveis ​​estáticas locais não inicializadas**. Qualquer variável local global ou estática que não seja inicializada será armazenada no segmento de dados não inicializado(bss). Por exemplo: variável global int VarGlobal;ou variável local estática static int VarLocalStatic;será armazenada no segmento de dados não inicializado. Se você declarar uma variável global e a inicializar como 0ou NULLainda assim, ela será direcionada para o segmento de dados não inicializado ou bss.

Ou seja, quaisquer variáveis**não-​​inicializadas**e que não são definidas dentro de uma função (e, portanto, podem ser acessadas de qualquer lugar) ou que são definidas em uma função, mas são definidas como *estáticas,(static)* para que mantenham seu endereço nas chamadas subsequentes estarão armazenadas aqui na região de .bss

**3. Segmento de dados inicializados(.data)**

Este segmento armazena:**Variáveis ​​globais inicializadas*(incluindo variáveis ​​de ponteiro)*, Variáveis ​​globais constantes  inicializadas** .**Variáveis ​​estáticas locais inicializadas** . Por exemplo: variável global int VarGlobalVar = 1;ou variável local estática static int VarLocalStatic = 1;será armazenada no segmento de dados inicializado. ***O tamanho desse segmento é determinado pelo tamanho dos valores no código-fonte do programa e não é alterado no tempo de execução***.

Ou seja, quaisquer variáveis**​​inicializadas**e que não são definidas em uma função (e, portanto, podem ser acessadas de qualquer lugar) ou que são definidas em uma função, mas são definidas como *estáticas,* para que mantenham seu endereço nas chamadas subsequentes estãrão armazenadas aqui na região de .data segmente.

**4. Segmento de pilha(stack)**

O segmento de pilha é usado para armazenar variáveis ​​criadas dentro de funções (a *função pode ser a principal ou definida pelo usuário* ), variáveis ​​como **Variáveis ​​locais** da função ***(incluindo variáveis ​​de ponteiro),*Argumentos passados ​​para a função,**As variáveis ​​armazenadas na pilha serão removidas assim que a execução da função terminar.  Cada função terá uma região de memória para ela dentro da STACK denominada de STACK FRAME

**5. Segmento de heap**

Este segmento é para oferecer suporte à **alocação dinâmica de memória.** Uma variável ponteiro estará na pilha(STACK) ou no segmento de dados(.BSS ou .Data), dependendo da maneira como é declarada usada. O heap é o segmento de memória que não está definido para um tamanho constante antes da compilação e pode ser controlado dinamicamente pelo programador. Usamos a memória do heap quando não sabemos quanto espaço uma estrutura de dados ocupará em nosso programa, quando precisamos alocar mais memória do que a disponível na pilha ou quando precisamos criar variáveis ​​que durem no tempo de duração da nossa aplicação.

Então veja que acima temos 3(três) possibilidades para variáveis ponteiros serem armazenadas nas regiões de memória...

Se for uma variável ponteiro global **não inicializada** será alocada na **.bss**

Se for uma variável ponteiro global **inicializada** será alocada na **data segment**

Se for uma variável ponteiro local a uma função será alocada na stack, precisamente na stack frame da função que a variável ponteiro foi declarada

Para simplificar consideramos apenas stack e heap e neste caso a stack conterá bss e data segment, pois isso as variáveis ponteiros estarão sempre na stack. Porém as variáveis que o ponteiro aponta estarão ou na stack ou quando forem alocadas dinamicamente estão na Heap

**O que ocorre quando o programa chamar/executar uma função?**

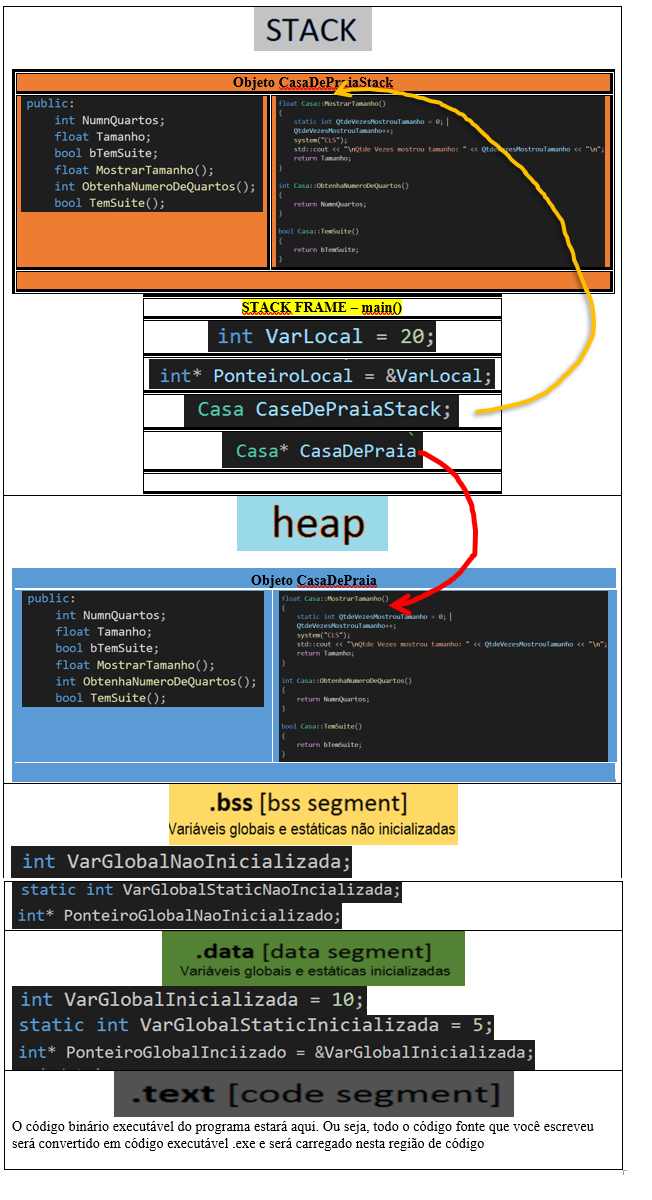
Quando você executa uma função em seu código é criado dentro da região stack  chamado STACK FRAME ou Quadro/Bloco de Pilha. É uma região que guarda as informações da função, incluindo variáveis para os parâmetros passados para a função, o endereço de retorno (para onde a instrução *return* aponta).  variáveis criadas dentro da função (variáveis locais). E é importante lembrar que se uma ponteiro é criado dentro de uma função ele irá pertencer a esta região de stack dentro da stack frame de sua função...

Nossa agora ficou confuso de vez professor!

Ok, vamos ver um exemplo de código para facilitar Leia os comentários do código....

1. #include <iostream>
3. /\*Relembrando, uma variável é global quando ela for declarada FORA de qualquer função do código\*/
4. int VarGlobalNaoInicializada;//<----Alocada no .bss Segmento de dados não inicializados
5. int VarGlobalInicializada = 10;//<----Alocada no Segmento de dados inicializados(.data)
6. static int VarGlobalStaticNaoIncializada;//<----Alocada no.bss Segmento de dados não inicializados
7. static int VarGlobalStaticInicializada = 5;//<----Alocada no Segmento de dados inicializados(.data)
8. int\* PonteiroGlobalNaoInicializado;//<----Alocada no .bss Segmento de dados não inicializados
9. int\* PonteiroGlobalInciizado = &VarGlobalInicializada;//<----Alocada no Segmento de dados inicializados(.data)
10. /\*Observe que este acima são ponteiros globais e por isso estarão em .bss ou .data segment\*/
12. class Casa
13. {
14. public:
15. int NumnQuartos;
16. float Tamanho;
17. bool bTemSuite;
18. float MostrarTamanho();
19. int ObtenhaNumeroDeQuartos();
20. bool TemSuite();
21. };
23. /\*Agora temos uma função que é a função principal main. O que for declarada dentro dela será uma variável local a esta função main. Mais atenção! new Casa() está criando dinâmicamente um objeto e por isso este objeto vai ficar na região de memória onde residem os recursos alocados dinâmicamente que é a HEAP\*/
24. int main()
25. {
26. int VarLocal = 20;//<---Estará na stack dentro da STACK FRAME da função main
27. int\* PonteiroLocal = &VarLocal;//<---Estará na stack dentro da STACK FRAME da função main
28. /\*Aqui criamos um objeto na Stack e um ponteiro do tipo Casa de nome CaseDePraiaStack. Então CaseDePraiaStack irá apontar para um objeto criado na STACK que teve como molde a classe Casa. É muito comum achar que CasaDePraiaStack seja o objeto. Não é! ele apenas é um ponteiro para um objeto do tipo da classe Casa que será alocado neste caso na STACK\*/
29. Casa CaseDePraiaStack;
31. /\*Aqui CasaDePraia é um ponteiro local a função main e portanto estará aramazenado na stack dentro da stack frama da função main(). Este ponteiro CasaDePraia irá apontar para um objeto criado dinâmicamente (new Casa()) e que portanto este objeto estará na Heap\*/
32. Casa\* CasaDePraia = new Casa();
34. CaseDePraiaStack.Tamanho = 100;
35. CasaDePraia->MostrarTamanho();
37. CasaDePraia->Tamanho = 50;
38. CasaDePraia->MostrarTamanho();
40. system("PAUSE");
41. return 0;
42. }
44. float Casa::MostrarTamanho()
45. {
46. static int QtdeVezesMostrouTamanho = 0; // Aqui apenas desta variável estar dentro de uma função
47. //ela estará armazenada na região de memória data segment(Segmento de dados inicializados(.data))
48. QtdeVezesMostrouTamanho++;
49. system("CLS");
50. std::cout << "\nQtde Vezes mostrou tamanho: " << QtdeVezesMostrouTamanho << "\n";
51. return Tamanho;
52. }
54. int Casa::ObtenhaNumeroDeQuartos()
55. {
56. return NumnQuartos;
57. }
59. bool Casa::TemSuite()
60. {
61. return bTemSuite;
62. }

Veja abaixo como ficaria dividido  a região de memória deste programa com o código acima



Na figura acima eu simplifiquei e não coloquei o STACK FRAME das outras funções, só coloquei o da função  main()

Se tivéssemos por exemplo o seguinte:  float \*Ptr=&Num;

Onde Ptr e Num estariam alocados?

Neste caso se isso estiver dentro da função main ou outra função tanto a variável Ptr quanto Num estariam na STACK dentro de um STACK FRAME da função em que estas variáveis residem.

Caso contrário se forem globais/estáticas estarão na .bss ou data segment a depender se foram inicializadas ou não...

É bom lembrar que esta divisão que coloquei acima na primeira figura não é de toda a memória RAM

o que é dividido desta forma é o espaço de endereçamento fornecido ao seu programa quando ele executa., As vezes um programa é dividido em vários processos e então cada processo executado terá este espaço de endereçamento.

Abraço