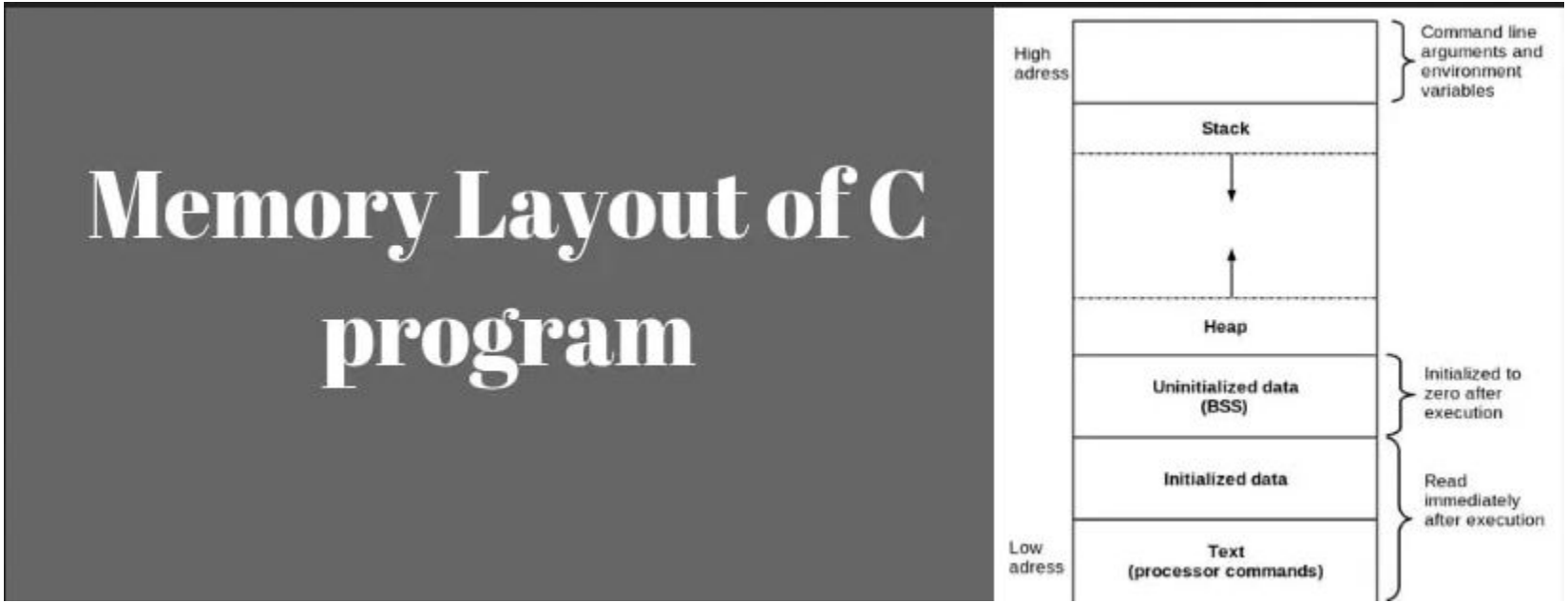
Funções e Endereços de Memória

Prof. Denio Duarte
duarte@uffs.edu.br
Prof. Claunir Pavan
claunir.pavan@uffs.edu.br

Endereços de Memória

- A Linguagem C trabalha com um conceito essencial em programação:
  - Acesso a endereços para armazenamento de dados na memória Principal

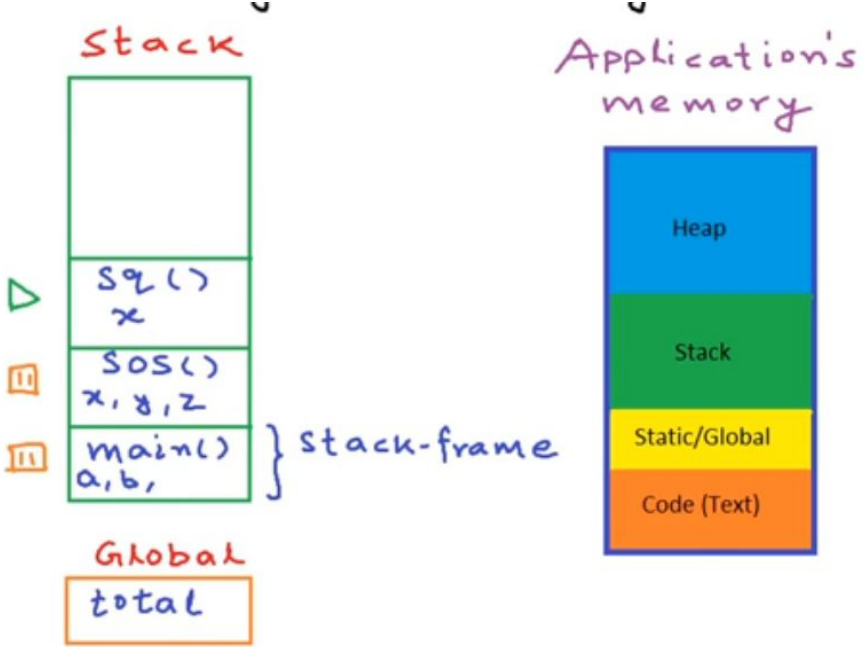


Fonte: https://aticleworld.com/memory-layout-of-c-program/

Endereços de Memória

- A Linguagem C trabalha com um conceito essencial em programação:
  - Acesso a endereços para armazenamento de dados na memória Principal

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return x*x;  // x^2
}
int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return z;  // (x+y)^2
}
int main()
{
    int a = 4, b = 8;
    total = SquareOfSum(a,b);
    printf("output = %d",total);
}
```



Fonte: https://www.youtube.com/watch?v=_8-ht2AKyH4

Endereços de Memória

- A Linguagem C trabalha com um conceito essencial em programação:
  - Acesso a endereços para armazenamento de dados na memória Principal
- Endereços em C são acessados através do operador &
  - printf(“Valor de a=**%d**”,a); printf(“Endereço de a=**%p**”,&a);
- Pode-se criar uma variável que armazene apenas endereços (respeitando o tipo da variável)
  - int a, *b; // **a** é uma variável “normal” e **b** armazena endereços (pois foi criada com ***** na frente)
  - a=10;
  - b=&a; // **b** agora aponta para o endereço de **a**
  - printf(“Endereço de a=%p e de b=%p”,&a,&b);

Fonte: https://www.youtube.com/watch?v=_8-ht2AKyH4

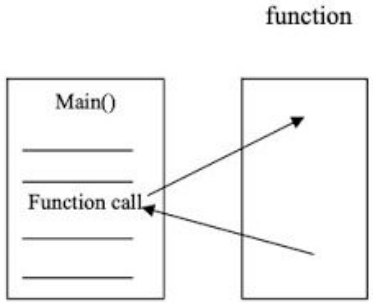
Ponteiros

- O conceito de uma variável armazenar o endereço de outra é chamado **ponteiro**
- Então, uma variável **aponta** para outra variável (ou para o endereço dela)
  - b=**&**a; // **b** aponta para **a** (ou **b** aponta para o endereço de **a**)
  - // b é chamada de variável ponteiro
- Uma variável ponteiro pode “alterar” o conteúdo da variável que aponta?
  - Sim, utilizando o operador * (CUIDADO: não é multiplicação, é *dereferencing*). * também é utilizado para criar a variável ponteiro

```
int a=5,*b;
b=&a;
*b=1;
print(“Valor de a=%d\n”,a); // aparecerá Valor de a=1
```

Funções

- Uma função é um conjunto de comandos (bloco de comandos) associado a um nome
  - O uso deste nome é uma chamada da função
- Chamada da função
  - Quando o programa realiza a chamada de uma função, esses dados são empilhados na memória, o bloco é executado, após seu término o programa continua a execução da próxima instrução do chamador.
    - Quando o bloco de execução de uma função termina é chamado de retorno da função.



Funções

- A chamada de uma função, geralmente, passa informações (argumentos) para o processamento da função
  - Lista pode ser vazia
  - Lista aparece entre ( ) junto ao nome da função
  - A declaração da função é chamada de protótipo da função
    - Por exemplo, o protótipo da função abaixo é `int addition(int, int);`

```python
def addition( a , b ):
    return a + b
```

Python x C

```c
int addition (int a, int b){
    return a + b;
}
```

Funções

- Uma função pode retornar resultados ao programa que a chamou
  - O tipo de retorno é definido no protótipo da função

função Python

```python
def addition( a , b ):
    return a + b
```

função C

```c
int addition (int a, int b){
    return a + b;
}
```

Funções

- Uma função pode retornar resultados ao programa que a chamou
  - O tipo de retorno é definido na definição da função
  - Uma função void não retorna nada

função Python

```
1  def funcTeste( a , b ):
2      print("Recebeu A: %s e B: %s"% (a,b));
```

função C

```
1  void funcTeste (int a, int b){
2      printf("Recebeu A: %d e B: %d", a,b);
3  }
```

Funções

- Definição de função
  - Sintaxe

```
1   tipo_retorno nomeFuncao (tipo paramentro, tipo parametro2){
2       //bloco
3   }
```

Qualquer tipo que quiser: int, float, double, char, etc.
Se **declarar** um **tipo** precisa de **retorno**
Se colocar **void** não **retorna nada**

```
1   int nomeFuncao (tipo paramentro, tipo parametro2){
2       //bloco
3       return valore;
4   }
```

Funções

- Definição de função
  - Sintaxe

```
1   tipo_retorno nomeFuncao (tipo paramentro, tipo parametro2){
2       //bloco
3   }
```

Pode colocar o nome que quiser, mas ele não pode conter caracteres especiais.

Funções

- Definição de função
  - Sintaxe

```
1   tipo_retorno nomeFuncao (tipo paramentro, tipo parametro2){
2       //bloco
3   }
```

Aceita quantos parâmetros forem necessários. Sempre seguindo o padrão:
tipo **nomeVar**
Deixar vazio se não tiver parâmetros.

```
1   void nomeFuncao (){
2       //bloco
3       printf("função sem parametro");
4       return; //opcional
5   }
```

Funções

- Exemplo Programa com função
  - Lê um valor inteiro e o eleva ao quadrado

```c
#include <stdio.h>

int elevaAoQuadrado (int val){
    return val*val;
}

int main(){
    int a, resultado;

    scanf("%d", &a); //lê valor interiro

    resultado = elevaAoQuadrado(a); //chamda de função

    printf("%d ^ 2 = %d\n", a,resultado);

    return (0);
}
```

```c
#include <stdio.h>
// protótipo da função
int elevaAoQuadrado(int);
int main() {
    int a, resultado;
    scanf("%d",&a);
    resultado=elevaAoQuadrado(a);
    printf("%d^2=%d\n",a,resultado);
    return 0;
}
// implementação da função
int elevaAoQuadrado(int val) {
    return val*val;
}
```

Funções

- Exemplo
  - Lê valores inteiros maiores que 0 e diz se é par ou ímpar, finaliza quando recebe um valor negativo ou zero

```c
#include <stdio.h>

void parOuImpar(int x){
    if(x % 2 == 0){
        printf("Par\n");
    }else{
        printf("Impar\n");
    }
}

int main(){

    int a = 1;

    while(a > 0){
        scanf("%d", &a);
        parOuImpar(a);
    }

    return 0;
}
```

Funções Exercícios

1. Faça uma função que lê dois inteiros e apresenta a diferença entre os dois.
2. Crie uma função que receba 2 números e retorne o maior valor entre eles (se forem iguais, retorna o segundo).
3. Crie uma função que receba 3 números e retorne o maior valor, utilizando uma chamada para função anterior.
4. Crie um aplicativo de conversão entre as temperaturas Celsius e Fahrenheit.
    a. Primeiro o usuário deve escolher se vai entrar com a temperatura em Celsius ou Fahrenheit, depois a conversão escolhida é realizada.
    b. Se C é a temperatura em Celsius e F em Fahrenheit, as fórmulas de conversão são:
        i. C= 5.(F-32)/9
        ii. F= (9.C/5) + 32

Funções

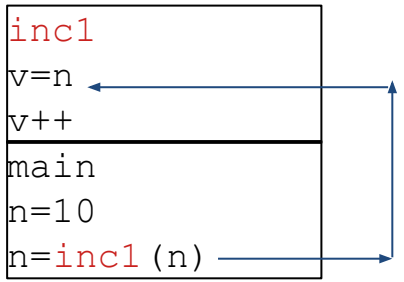
- Quando passamos valores para uma função através de parâmetros, podemos fazer de duas formas
  - Por cópia
    - É a forma padrão, quando criamos o valor é passado uma cópia dos parâmetros para serem utilizados no escopo da função.
    - Os valores das variáveis originais não podem ser alterados pela função
  - Por referência
    - Passamos o endereço de memória que armazena o dado, no escopo da função o dado é acessado através do endereço e qualquer alteração é vista no exterior.
    - Ou seja, os valores originais podem ser alterados pela função

Funções

- Parâmetro por cópia

```
int inc1 (int v){
    return v++;
}
//
int main() {
    int n=10;
    n=inc1(n);
    printf("%d",n);
    return 0;
}
```

v será uma cópia da variável passada para ela.
No exemplo, seria como se fizéssemos
v=n



Funções

- Parâmetro por referência/endereço

```
void inc1 (int *v){
    (*v)++;
}
int main() {
    int n=10;
    inc1(&n);
    printf("%d",n);
    return 0;
}
```

v recebe o endereço da variável passada para ela.
No exemplo, n e v estão apontando para o mesmo lugar na memória