MC202 - Estruturas de Dados

Alexandre Xavier Falcão

Instituto de Computação - UNICAMP

afalcao@ic.unicamp.br

Indo de Python para C

Considere um script Python para calcular a área de um círculo.

```
PI = 3.141592653589793

def AreaCirculo(raio):
    area = 2*PI*raio**2
    return(area)

# Função principal

if __name__ == "__main__":

    raio = float(input("Entre com o raio (cm): "))
    area = AreaCirculo(raio)
    print("Área do círculo: {:.2f}cm2".format(area))
```

Temos variáveis globais e locais, funções de entrada e saída padrão, e uma função definida pelo programador.

Indo de Python para C

- Em C, todo **programa fonte** necessita de uma função principal.
- Constantes e variáveis globais podem ser definidas, mas as globais devem ser evitadas.
- Em vez de importar pacotes, inclui-se **bibliotecas** de funções.
- O programa fonte precisa ser **compilado** para gerar um códido executável.

```
> gcc area-circulo.c -o
> ./area-circulo
```

Indo de Python para C

O mesmo código em C fica.

```c
#include <stdio.h>

#define PI 3.141592653589793

float AreaCirculo(float raio)
{
  float area = 2*PI*raio*raio;
  return(area);
}

int main()
{
  float raio, area;

  printf("Entre com o raio (cm): ");
  scanf("%f",&raio);
  area = AreaCirculo(raio);
  printf("Área do círculo: %.2fcm2\n",area);
  return(0);
}
```

Vamos rever conceitos e aprender que variáveis possuem tipo, nome, endereço na memória principal, e ocupam espaço em memória.

Agenda

- Tipos de variáveis simples e espaço em memória.
- Apontadores.
- Quando passar o endereço ou o valor da variável para uma função.
- Como é o uso da memória durante a execução do código: `www.pythontutor.com`.

Variáveis simples e espaço em memória

Variáveis simples armazenam número ou letra, e podem ser de diversos tipos. Em um processador de 64 bits,

Variáveis simples e espaço em memória

Variáveis simples armazenam número ou letra, e podem ser de diversos tipos. Em um processador de 64 bits,

- o tipo **char** armazena um caracter alfanumérico ou valor inteiro em 1 byte.

Variáveis simples e espaço em memória

Variáveis simples armazenam número ou letra, e podem ser de diversos tipos. Em um processador de 64 bits,

- o tipo **char** armazena um caracter alfanumérico ou valor inteiro em 1 byte.
- **unsigned char** armazena um valor não negativo em 1 byte.

Variáveis simples e espaço em memória

Variáveis simples armazenam número ou letra, e podem ser de diversos tipos. Em um processador de 64 bits,

- o tipo **char** armazena um caracter alfanumérico ou valor inteiro em 1 byte.
- **unsigned char** armazena um valor não negativo em 1 byte.
- **short**, **int**, e **long** armazenam valores inteiros em 2, 4, e 8 bytes, respectivamente.

Variáveis simples e espaço em memória

Variáveis simples armazenam número ou letra, e podem ser de diversos tipos. Em um processador de 64 bits,

- o tipo **char** armazena um caracter alfanumérico ou valor inteiro em 1 byte.
- **unsigned char** armazena um valor não negativo em 1 byte.
- **short**, **int**, e **long** armazenam valores inteiros em 2, 4, e 8 bytes, respectivamente.
- **unsigned short**, **unsigned int**, e **unsigned long** armazenam valores inteiros não negativos em 2, 4, e 8 bytes, respectivamente.

Variáveis simples e espaço em memória

Variáveis simples armazenam número ou letra, e podem ser de diversos tipos. Em um processador de 64 bits,

- o tipo **char** armazena um caracter alfanumérico ou valor inteiro em 1 byte.
- **unsigned char** armazena um valor não negativo em 1 byte.
- **short**, **int**, e **long** armazenam valores inteiros em 2, 4, e 8 bytes, respectivamente.
- **unsigned short**, **unsigned int**, e **unsigned long** armazenam valores inteiros não negativos em 2, 4, e 8 bytes, respectivamente.
- **float**, **double**, e **long double** armazenam valores reais em 4, 8, e 16 bytes, respectivamente.

Variáveis simples e espaço em memória

- Variáveis inteiras de b bytes armazenam valores no intervalo $[\frac{-2^{8b}}{2}, \frac{2^{8b}-1}{2}]$.

  Ex: $b = 2$ bytes permite valores no intervalo $[-32768, 32767]$.
  Valores fora deste intervalo geram overflow.

Variáveis simples e espaço em memória

- Variáveis inteiras de b bytes armazenam valores no intervalo $[\frac{-2^{8b}}{2}, \frac{2^{8b}-1}{2}]$.

  Ex: $b = 2$ bytes permite valores no intervalo $[-32768, 32767]$. Valores fora deste intervalo geram overflow.

- Variáveis inteiras não sinalizadas (**unsigned**) de b bytes armazenam valores no intervalo $[0, 2^{8b} - 1]$.

  Ex: $b = 2$ bytes permite valores não negativos no intervalo $[0, 65535]$.

Variáveis simples e espaço em memória

- Variáveis inteiras de b bytes armazenam valores no intervalo $[\frac{-2^{8b}}{2}, \frac{2^{8b}-1}{2}]$.

  Ex: $b = 2$ bytes permite valores no intervalo $[-32768, 32767]$. Valores fora deste intervalo geram overflow.

- Variáveis inteiras não sinalizadas (**unsigned**) de b bytes armazenam valores no intervalo $[0, 2^{8b} - 1]$.

  Ex: $b = 2$ bytes permite valores não negativos no intervalo $[0, 65535]$.

- Variáveis são declaradas no escopo das funções, especificando o **tipo** e o **nome**, com inicialização opcional.

Variáveis simples e espaço em memória

- Variáveis reais, float, double, e long double, armazenam valores nos intervalos $[-FLT\_MAX, FLT\_MAX]$, $[-DBL\_MAX, DBL\_MAX]$ e $[-LDBL\_MAX, LDBL\_MAX]$ definidos em float.h, respectivamente.

Variáveis simples e espaço em memória

- Variáveis reais, float, double, e long double, armazenam valores nos intervalos $[-FLT\_MAX, FLT\_MAX]$, $[-DBL\_MAX, DBL\_MAX]$ e $[-LDBL\_MAX, LDBL\_MAX]$ definidos em float.h, respectivamente.

- Cuidado: FLT_MIN, DBL_MIN, e LDBL_MIN são valores positivos muito próximos de zero.

Variáveis simples e espaço em memória

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

int main()
{
   printf("min: %E max: %E\n",DBL_MIN,DBL_MAX);
   return(0);
}
```

A saída será min: 2.225074E-308 max: 1.797693E+308.

Variável apontadora

- Toda variável possui um endereço na memória principal.

Variável apontadora

- Toda variável possui um endereço na memória principal.
- Uma variável **apontadora** (ponteiro) é uma "variável simples" de 4 bytes que armazena o endereço de outra variável (ou função).

Variável apontadora

- Toda variável possui um endereço na memória principal.
- Uma variável **apontadora** (ponteiro) é uma "variável simples" de 4 bytes que armazena o endereço de outra variável (ou função).
- O ponteiro permite mudar o conteúdo da variável apontada por ele.

float a=100, *b;	declaração do ponteiro b
b = &a;	b recebe o endereço de a
*b = 50;	variável apontada por b recebe 50
	e o conteúdo de a agora é 50

Passagem por endereço ou valor de uma variável

- Funções sempre recebem cópias dos seus argumentos, mas estes podem representar o valor de uma variável ou o seu endereço.

Passagem por endereço ou valor de uma variável

- Funções sempre recebem cópias dos seus argumentos, mas estes podem representar o valor de uma variável ou o seu endereço.

- No programa area-circulo.c, scanf recebeu a cópia do endereço da variável raio e AreaCirculo recebeu a cópia do seu valor.

Passagem por endereço ou valor de uma variável

- Funções sempre recebem cópias dos seus argumentos, mas estes podem representar o valor de uma variável ou o seu endereço.
- No programa area-circulo.c, scanf recebeu a cópia do endereço da variável raio e AreaCirculo recebeu a cópia do seu valor.
- A cópia do valor é armazenada em uma variável local da função e alterações nesta variável não vão afetar o conteúdo da variável original.

Passagem por endereço ou valor de uma variável

- Funções sempre recebem cópias dos seus argumentos, mas estes podem representar o valor de uma variável ou o seu endereço.
- No programa area-circulo.c, scanf recebeu a cópia do endereço da variável raio e AreaCirculo recebeu a cópia do seu valor.
- A cópia do valor é armazenada em uma variável local da função e alterações nesta variável não vão afetar o conteúdo da variável original.
- No caso de scanf, porém, a cópia do endereço da variável é usada para acessar e modificar o conteúdo da variável original.

Passagem por endereço ou valor de uma variável

```c
#include <stdio.h>

#define PI 3.141592653589793

float AreaCirculo(float raio)
{
  float area = 2*PI*raio*raio;
  return(area);
}

int main()
{
  float raio, area;

  printf("Entre com o raio (cm): ");
  scanf("%f",&raio);
  area = AreaCirculo(raio);
  printf("Área do círculo: %.2fcm2\n",area);
  return(0);
}
```

Passagem por endereço ou valor de uma variável

```c
#include <stdio.h>

void TrocaValor(int *a, int *b)
{
  int c = *a;
  *a = *b;
  *b = c;
}

int main()
{
  int a, b;

  printf("Entre com a e b: ");
  scanf("%d %d",&a,&b);
  TrocaValor(&a,&b);
  printf("Valores trocados: a=%d b=%d\n",a,b);
  return(0);
}
```

Note que, por falta de criatividade, * é usado para declarar o ponteiro e para acessar o conteúdo de memória apontado por ele.

Memória durante a execução do código

Para entender o fluxo de informações durante a execução de um código, vamos continuar a aula em `www.pythontutor.com`.