Alfredo's MAC0110 Journal

Alfredo Goldman

28 de junho de 2020

0.1 Aula 25 - C para quem programa em Julia

Como vocês devem lembrar, na primeira aula deixei claro que a linguagem de programação era apenas o meio de se traduzir algum conceito algorítmico para o computador, de forma a permitir a sua execução.

Como infelizmente, Julia ainda não é uma linguagem bastante conhecida, várias outras linguagens mais tradicionais farão compania para vocês no curso. Conhecer várias linguagens não é ruim, mas por outro lado Julia não ser conhecida é :)

Nessa aula, a ideia é apresentar a linguagem C para quem já programa em Julia. Há algumas diferenças básicas

0.1.1 Compilado versus Interpretada

Enquanto Julia é uma linguage interpretada, C é uma linguagem compilada, isso é, a partir de um código fonte, ao se passar pelo compilador, é gerado um código objeto, que se correto, pode ser executado na arquitetura para qual foi compilado.

Vejamos um primeiro código em C.

```
#include <stdio.h>
int main() {
  printf ("Hello World!\n");
}
```

Para compilar o código acima, usamos o comando

```
gcc um.c
```

Que vai gerar um arquivo executável a.out, que se chamado imprime a mensagem. O ponto de entrada inicial de um programa em C é único e é a função main().

Mas, dá para ver mais umas coisas no código acima que são diferenças em relação à Julia. Mesmo para coisas básicas como impressão é necessário incluir bibliotecas, no caso a stdio.h que possui a função printf().

Os blocos são definidos com chaves (abre e fecha) e o ponto e vírgula delimita os comandos.

Além disso, já dá para ver que C é fortemente tipado, isso é, é necessário dizer o tipo de tudo ou seja, a função main() acima, não devolve nada.

0.1.2 Linguagem Tipada

Para cada variável em C, é necessário definir o seu tipo, vamos a mais um exemplo:

```
#include <stdio.h>
 void main() {
   int a = 1;
   int b = 2;
   int c;
   c = a + b
   printf ("O valor de c é %d\n", c);
 }
```

Acima é possível ver que podemos dar o tipo e definir a variável na mesma linha, ou declarar e depois usar. Não é possível usar uma variável sem declarar explicitamente. Isso tem vantagens claras, pois possíveis erros podem ser encontrados já em tempo de compilação, antes da execução.

O comando de impressão também segue uma sintaxe diferente, recebendo primeiro uma string, e depois uma lista de parâmetros. Nessa string, para saber como imprimir cada um dos parâmetros e usar %, no caso %d para inteiros, %g para ponto flutuante e %s para string. O barra n no final é um indicativo para pular linha.

A declaração de funções é semelhante, só que para cada variável passada como parâmetro é necessário passar o seu tipo. Os tipos mais comuns em C são, int, char, float e double. Não há o tipo boolean em C, o que se faz é usar comparações, ou tipos inteiros, basicamente 0 equivale a falso e outros valores a verdadeiro.

```
#include <stdio.h>

int soma(int a, int b){
  return a + b;
}
void main() {
  int a = 1;
  int b = 2;
  printf ("O valor  é %d\n", soma(a, b));
  if (soma(a, b) == 3)
    printf(" Verdade = %d\n", soma(a, b) == 3);  // Bloco sem chaves
}
```

No código acima podemos ver que se o bloco tem apenas uma instrução, não precisa usar as chaves.
Assim como em Julia, a recursão também funciona bem em C.

0.1.3 Comandos diferentes

Já o for em C é composto por três parâmetros, todos opcionais, a inicialização, a condição e o passo.

```
#include <stdio.h>
void main(){
  for (int p = 1; p <= 512; p *= 2) {
    printf("%d\n", p);
  }
}
```

A sintaxe do if é um pouco diferente, principalmente no que se refere ao uso de elses. Vejamos um exemplo e aproveitemos para usar o comando de entrada de dados pelo teclado, o scanf
Comando switch

```
#include <stdio.h>
void main(){
  int n;

  printf("Entre com um número: ");
  scanf("%d", &n);
    if (n < 0)
      printf("Número negativo\n");
      else if (n > 0)
        printf("Número positivo\n");
        else
          printf("zero\n");
}
```

Observem que quanto mais elses, mais iríamos para a direita, logo isso se escreve de uma forma alternativa:

```
#include <stdio.h>
void main(){
  int n;

  printf("Entre com um número: ");
  scanf("%d", &n);
    if (n < 0)
      printf("Número negativo\n");
    else if (n > 0)
      printf("Número positivo\n");
    else
      printf("zero\n");
}
```

Mas, o principal acima é o operador &, que obtém o endereço de uma variável, ou seja a sua posição na memória, podemos ver o efeito disso na seção abaixo.

Mas, antes um exemplo do uso de switch.

```
#include <stdio.h>
void main(){
    int n;

    printf("Qual a sua carta (1-13)? ");
    scanf("%d", &n);
    switch (n) \{
      case  1: printf("Ace\n"); break;
      case 11: printf("Jack\n"); break;
      case 12: printf("Queen\n"); break;
      case 13: printf("King\n"); break;
      default: printf("%d\n", n);
    }
}
```

0.1.4 Vetores e ponteiros

Vejamos o exemplo abaixo, o primeiro a lidar com ponteiros de forma mais explícita.

```
#include <stdio.h>
void naoModifica(int a){
  a = 2;
}
void Modifica(int *a){
  *a = 2;
}

void main(){
  int n = 3;

  printf("A variável n vale %d\n", n);
  naoModifica(n);
  printf("A variável n vale %d\n", n);
  Modifica(&n);
  printf("A variável n vale \%d\n", n);
}
```

O mais próximo que vimos de ponteiros em Julia foi o conceito de vetores, onde um vetor também é um ponteiro, mas que também guarda o seu tamanho.

Vamos a um exemplo de vetores em C.

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

void imprimeVetor(int *v, int tam){
  for (int i = 0; i < tam; i++)
    printf("v[%d] = %d  ",i, v[i]);
  printf("\n");
}

void inicializaVetor(int *v, int tam){
  for (int i = 0; i < tam; i++)
    v[i] = rand() % 100;
}

void vezes2Vetor(int *v, int tam){
  for (int i = 0; i < tam; i++)
    v[i] = 2 * v[i];
```

```
}



void main()
{
  int vetor[10];
  inicializaVetor(vetor, 10);
  imprimeVetor(vetor, 10);
  vezes2Vetor(vetor, 10);
  imprimeVetor(vetor, 10);
}
```

Mas, com grandes poderes vem grandes responsabilidades, veja o programa abaixo com uma pequena modificação e um erro inserido.

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

void imprimeVetor(int *v, int tam){
  for (int i = 0; i < tam; i++)
    printf("v[%d] = %d  ",i, v[i]);
  printf("\n");
}

void inicializaVetor(int *v, int tam){
  for (int i = 0; i < tam; i++)
    v[i] = rand() % 100;
}

void vezes2Vetor(int *v, int tam){
  for (int i = 0; i < tam; i++)
    v[i] = 2 * v[i];
}



void main()
{
  int vetor[10];
  int *ptr;

  ptr = vetor;
  inicializaVetor(ptr, 10);
  imprimeVetor(ptr, 10);
  vezes2Vetor(ptr, 10);
  ptr++;
  imprimeVetor(ptr, 10);

}
```

ou

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

void imprimeVetor(int *v, int tam){
  for (int i = 0; i < tam; i++)
    printf("v[%d] = %d  ",i, v[i]);
  printf("\n");
}
```

```
void inicializaVetor(int *v, int tam){
  for (int i = 0; i < tam; i++)
    v[i] = rand() % 100;
}

void vezes2Vetor(int *v, int tam){
  for (int i = 0; i < tam; i++)
    v[i] = 2 * v[i];
}



void main()
{
  int *ptr;


  ptr = malloc(10 * sizeof(int));
  inicializaVetor(ptr, 10);
  imprimeVetor(ptr, 11);
  vezes2Vetor(ptr, 11);
  imprimeVetor(ptr, 11);
}
```