que sabe!

Uma introdução à linguagem C

Felipe M. Megale
http://github.com/felipemegale/cqsabe

Breve história da linguagem

- Desenvolvida por Dennis Ritchie no período entre 1969 e 1973 na empresa Bell Labs

- Utilizada na re-implementação do sistema Unix

- Já teve várias versões
  - Hoje estamos na C11 (estável)

Primeiro programa

```
// first-1.c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
//ou    (int argc, char *argv[])
{
    printf(“hello world!\n”);
    return 0;
}
```

```
// first-2.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf(“hello world!\n”);
    return 0;
}
```

I/O

- Entrada
  - int scanf(const char* format, ...); ←
  - int getch(void);
  - char * gets(char * str);
  - char * fgets(char * str, int num, FILE * stream); ←
  - int fscanf(FILE * stream, const char * format, ...);
  - size_t fread(void * ptr, size_t size, size_t count, FILE * stream); ←

I/O

- Saída
  - int printf(const char * format, …); ←
  - int fprintf(FILE * stream, const char * format, …);
  - size_t fwrite(const void * ptr, size_t size, size_t count, FILE * stream); ←

I/O

- Arquivos


```
// file-1.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        FILE* arquivo;
        float pi = 3.14;
        arquivo = fopen(“filename.txt”, “w”);
        fprintf(arquivo, “%f”, pi);
        fclose(arquivo);
        return 0;
}
```

```
// file-2.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        FILE* arquivo;
        float pi;
        arquivo = fopen(“filename.txt”, “r”);
        fscanf(arquivo, “%f”, &pi);
        fclose(arquivo);
        printf(“Pi = %f\n”, pi);
        return 0;
}
```

I/O

- Arquivos


```
// file-3.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        FILE* arquivo;
        float pi = 3.14;
        arquivo = fopen(“filename.txt”, “w”);
        fwrite(&pi, sizeof(float), 1, arquivo);
        fclose(arquivo);
        return 0;
}
```

```
// file-4.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        FILE* arquivo;
        float pi;
        arquivo = fopen(“filename.txt”, “r”);
        fread(&pi, sizeof(float), 1, arquivo);
        fclose(arquivo);
        printf(“Pi = %f\n”, pi);
        return 0;
}
```

I/O

- Um ponto importante das funções de I/O:
  - A formatação deve ser condizente com o que se quer ler ou escrever
  - Se quero ler um int, devo pedir um int
  - Se quero escrever um int, devo esperar um int

```
int a;
scanf(“%i”, &a);

float a;
scanf(“%f”, &a);

double a;
scanf(“%lf”, &a);
```

```
int a;
a = 3;
printf(“%i”, a);

float a;
a = 3.14;
printf(“%f”, a);

double a;
a = 3.14;
printf(“%lf”, a);
```

Funções

- Da mesma forma que temos uma função principal, podemos criar qualquer outra função
- Basta colocá-la ou prototipar acima da função principal main. No segundo caso, devemos implementar a função abaixo do main

Funções - Exemplo

```
// function-1.c
#include <stdio.h>
#include <stdlib.h>

void mostraint(int a)
{
        printf(“Int: %i\n”, a);
}

int main()
{
        int a;
        printf(“Informe um int: “);
        scanf(“%i”, &a);
        mostraint(a);
        return 0;
}
```

```
// function-2.c
#include <stdio.h>
#include <stdlib.h>

void mostraint(int a);

int main()
{
        int a;
        printf(“Informe um int: “);
        scanf(“%i”, &a);
        mostraint(a);
        return 0;
}

void mostraint(int a)
{
        printf(“Int: %i\n”, a);
}
```

Funções

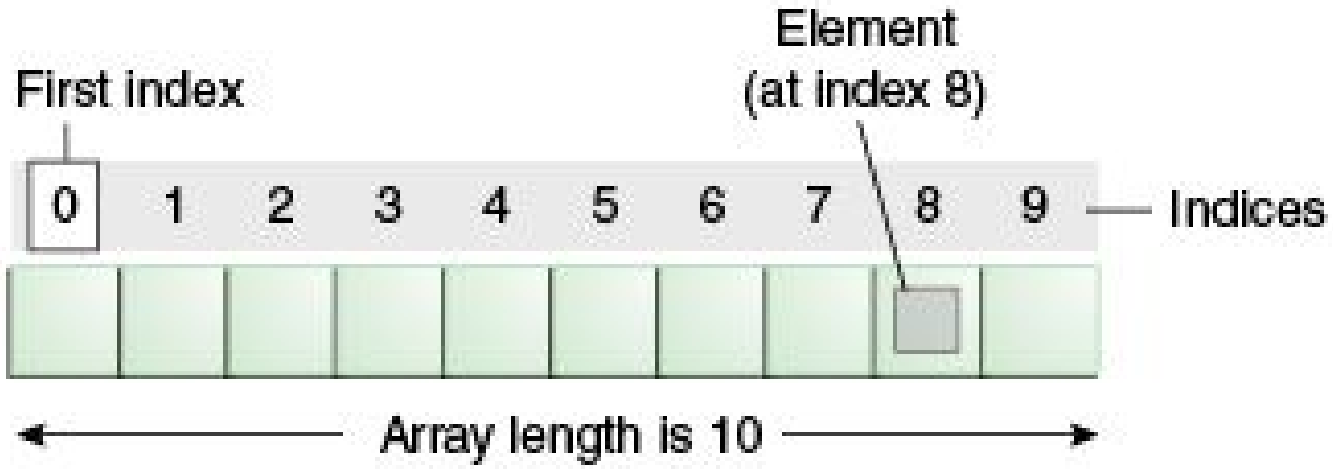
- Podem podem retornar qualquer tipo de dado, desde que a sintaxe esteja correta

```
// function-3.c
#include <stdio.h>
#include <stdlib.h>

int voltadobro(int a)
{
        int resp = 2*a;
        return resp;
}
int main()
{
        int a = 3;
        int b = voltadobro(a);
        printf(“B = %i\n”, b);
        return 0;
}
```

```
// function-4.c
#include <stdio.h>
#include <stdlib.h>

float voltaquadrado(float a)
{
        float resp = a*a;
        return resp;
}
int main()
{
        float a = 3.14;
        float b = voltaquadrado(a);
        printf(“B = %f\n”, b);
        return 0;
}
```

Arrays

- São espaços contínuos de memória, separados para um tipo de dado



Arrays

- Podem ser declarados quase igual ao Java

```
// array-1.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        int a[10], i;

        for(i = 0; i < 10; i++)
                a[i] = i+1;

        for(i = 0; i < 10; i++)
                printf(“a[%i]: %i\n”, i, a[i]);

        return 0;
}
```

Arrays

- É possível instanciar todo o vetor no momento da declaração!

```
// array-2.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        int a[10] = {1,2,3,4,5,6,7,8,9,10}, i;

        for(i = 0; i < 10; i++)
                a[i] = i+1;

        for(i = 0; i < 10; i++)
                printf("a[%i]: %i\n", i, a[i]);

        return 0;
}
```

Arrays - Um uso da diretiva define

- A diretiva define nos permite criar uma constante simbólica para nosso programa. O pré-processador troca tudo antes de compilar o código

```
// array-3.c
#include <stdio.h>
#include <stdlib.h>

#define TAM 10

int main()
{
        int a[TAM], i;

        for(i = 0; i < TAM; i++)
                a[i] = i+1;

        for(i = 0; i < TAM; i++)
                printf("a[%i]: %i\n", i, a[i]);

        return 0;
}
```

Arrays Bidimensionais (Matrizes)

- Semelhantemente a um array, podemos ter uma matriz

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Arrays Bidimensionais (Matrizes)

- Podem ser declarados semelhantemente aos arrays unidimensionais

```
// matriz-1.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        int a[2][2], i, j;
        for(i = 0; i < 2; i++)
                for(j = 0; j < 2; j++)
                        a[i][j] = i+j;
        for(i = 0; i < 2; i++)
                for(j = 0; j < 2; j++)
                        printf("a[%i][%i]: %i\n", i, j, a[i][j]);
}
```

Arrays Bidimensionais (Matrizes)

- Podem ser declarados e inicializados semelhantemente aos vetores

```
// matriz-2.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
	int a[2][2] = {{1,2}, {3,4}}, i, j;

	for(i = 0; i < 2; i++)
		for(j = 0; j < 2; j++)
			printf("a[%i][%i]: %i\t", i, j, a[i][j]);

	return 0;
}
```

Te deixa mais poderoso que o Dr. Manhattan!

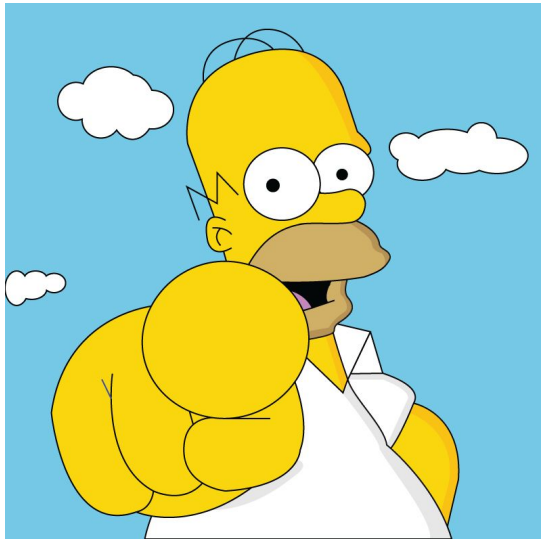
- A linguagem permite ao programador grande acesso à memória

- Precisamos ter muito cuidado ao mexer com isso por questões de segurança, estabilidade e desempenho dos nossos programas

Como ter esse poder em minhas mãos??

A resposta é simples:

Ponteiros!



O que é um ponteiro?

- Variável que guarda endereço de memória!
- Representamos ponteiros com *
- Endereço é representado por &
- Conteúdo do ponteiro é representado com * também!

```
// pointer-1.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        int a;
        int* b;
        a = 6;
        b = &a;
        printf(“A: %i\n”, a);
        printf(“B: %p\n”, b);
        return 0;
}
```

E eu consigo apontar pra quê?

- Basicamente, pra tudo que você quiser e que esteja disponível na memória!

- Também conseguimos apontar para arquivos (como visto na parte de entrada e saída)

Pegar um valor a partir de um ponteiro

- Como pegar o valor que está guardado numa variável usando um ponteiro??

- Esta técnica se chama desreferência

```
// pointer-2.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        int a = 42;
        int* a_ptr = &a;

        printf(“Valor de a: %i\n”, a);
        printf(“Endereco de a: %p\n, a_ptr);
        printf(“Valor de a pelo endereco: %i\n”, *a_ptr);

        return 0;
}
```

Ponteiros - malloc/free

- Podem ser usados como vetores pois apontam para o primeiro bloco do vetor

```
// pointer-3.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        int* a, i;

        a = (int*) malloc(sizeof(int));

        for(i = 0; i < sizeof(int); i++)
                a[i] = i+1;

        for(i = 0; i < sizeof(int); i++)
                printf("a[%i]: %i\n", i, a[i]);

        free(a);

        return 0;
}
```

Ponteiros - malloc/free

- Analogamente, podemos usar ponteiros como uma matriz!

```
// pointer-4.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
	int** a = (int**) malloc(sizeof(int));
	int i, j;

	for(i = 0; i < sizeof(int); i++)
		a[i] = (int*)malloc(sizeof(int));

	for(i = 0; i < sizeof(int); i++)
		for(j = 0; j < sizeof(int); j++)
			a[i][j] = i+j;

	for(i = 0; i < sizeof(int); i++)
	{
		for(j = 0; j < sizeof(int); j++)
			printf("a[%i][%i]: %i\t", i, j, a[i][j]);
		printf("\n");
	}

	return 0;
}
```

Ponteiros - malloc/free

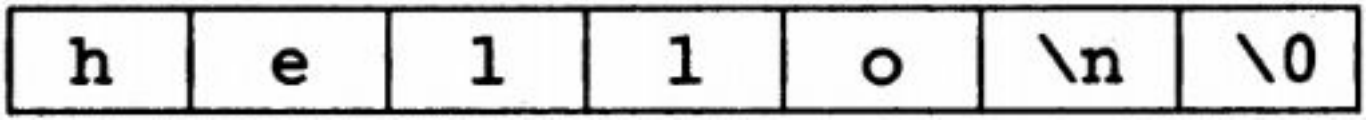
- Da mesma forma que alocamos memória para matrizes e vetores, temos de desalocar também, para evitar consumo excessivo e inútil de memória útil!

Um caso particular de arrays

- Bom, não vimos tudo aquilo à toa, né?

- A linguagem C tem uma peculiaridade que é: não tem string!

- Ela conta com arrays de caracteres, em que cada array termina, **SEMPRE** com um \0 na última posição do array



string.h

- Facilita a manipulação de strings, por exemplo...

- Nos fornece meios práticos de saber o tamanho de uma string (size_t strlen(const char* str))

- Também nos permite copiar uma string para outra (char* strcpy(char* destination, const char* origin))

string.h

```
// string-1.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
        char* str;
        int size;
        printf(“Digite uma string: “);
        fgets(str, 1024, stdin);
        size = strlen(str);
        printf(“Tamanho da string: %i\n”, size);
}
```

string.h

- Por que o resultado não foi o esperado?

Para finalizar

- Podemos criar nossos tipos personalizados de dados usando struct

```
// struct-1.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct pessoa
{
        char* name;
        char* surname;
};

int main()
{
        struct pessoa a;
        struct pessoa* aPtr;

        a.name = (char*) mallor(sizeof(char)*6);
        a.surname = (char*) mallor(sizeof(char)*6);

        strcpy(a.name, “Felipe”);
        strcpy(a.surname, “Megale”);
        aPtr = &a;

        printf(“%s %s\n”, a.name, a.surname);
        printf(“%s %s\n”, aPtr->name, aPtr->surname);
        printf(“%s, %s\n”, (*aPtr).name, (*aPtr).surname);

        return 0;
}
```

Para finalizar

- Podemos aliar o struct ao typedef para suprimir a necessidade de declarar struct <struct_name> <var_name>;

- typedef também é usado para representar dados iguais, mas de diferentes nomes em cada processador

Freqüentemente, **typedef** é usado para criar sinônimos de tipos básicos de dados. Por exemplo um programa que exija inteiros de 4 bytes pode usar o tipo **int** em um sistema e o tipo **long e, outro.** Os programas que devem apresentar portabilidade usam freqüentemente **typedef** para criar um alias para inteiros de 4 bytes como **Integer.** O alias **Integer** pode ser modificado uma vez no programa para fazer com que ele funcione em ambos os sistemas.

Para finalizar

```
// struct-2.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct pessoa
{
        char* name;
        char* surname;
}Person;

int main()
{
        Person a;
        Person* aPtr;

        a.name = (char*) mallor(sizeof(char)*6);
        a.surname = (char*) mallor(sizeof(char)*6);

        strcpy(a.name, “Felipe”);
        strcpy(a.surname, “Megale”);
        aPtr = &a;

        printf(“%s %s\n”, a.name, a.surname);
        printf(“%s %s\n”, aPtr->name, aPtr->surname);
        printf(“%s, %s\n”, (*aPtr).name, (*aPtr).surname);

        return 0;
}
```

Obrigado!

Dúvidas??

Referências

C: Como Programar, Deitel 6ª edição

The C Programming Language, Brian Kernighan & Dennis Ritchie
ANSI C

cplusplus.com ←