



Algoritmos e Programação de Computadores

Tipos primitivos de dados

Prof. Lucas Boaventura
lucasxboaventura18@gmail.com





Introdução

- Durante a programação, comumente é necessário armazenar valores
- Em um navegador, precisamos do endereço da página a ser visitada
- Uma calculadora, precisa dos números que fará a operação





Introdução

- Toda informação contida está armazenada na memória do computador

Endereço

Valor

...

...

200	0
204	8355
208	4
212	-1366160

...

...





Introdução

- Para que o nosso programa possa utilizar essa memória, utilizamos variáveis
- No código, declaramos:
- **tipo_da_variável** nome_da_variável;
- O tipo da variável deve ser um dos tipos suportados na linguagem e no seu código
- O nome da variável deve seguir as regras de declaração





Regras de nome da variável

- Uma variável pode conter letras, números ou underscore (_)
- Sempre deve iniciar com letras ou underscore
- A linguagem C é case-sensitive, sensível a maiúsculas e minúsculas
- Ou seja, a variável “Soma” e “soma” são variáveis diferentes





Regras de nome da variável

- Além disso, C possui 32 palavras-chave reservadas que não podem ser usadas como nome de variáveis

auto	const	double	float	int	short	struct	unsigned
break	continue	else	for	long	signed	switch	void
case	default	enum	goto	register	sizeof	typedef	volatile
char	do	extern	if	return	static	union	while





Regras de nome da variável

- Não se preocupe em decorar de todas as palavras-chave
- Utilizaremos um editor de texto que colore as palavras quando são reservadas, desta forma você irá detectar que usou uma
- Apenas se preocupe em saber que elas existem e que não podem ser usadas como nome





Tipos de variáveis

- Alguns tipos básicos de variáveis:

Tipo	Uso
char	Armazenar letras
int	Armazenar números inteiros
float	Armazenar números com casas decimal
double	Armazenar números com casas decimal





Tipos de variáveis

- Alguns tipos básicos de variáveis:

Tipo	Bits	Valor
char	8	-128 a 127
int	32	-2.147.483.648 a 2.147.483.647
float	32	complexo
double	64	complexo





Int

- Uma variável int armazena um número inteiro, ou seja, sem casas decimais
- **int** n = 10;
- Neste caso, ele irá armazenar um valor decimal





Imprimindo as variáveis

- Cada uma dessas variáveis podem ser utilizadas de diversas formas
- Uma das formas mais básica é imprimir na tela com a função “printf”, por exemplo:
- `printf(“%d\n”, i);`
- Imprime a variável “i” e uma nova linha na tela





Imprimindo as variáveis

```
#include <stdio.h>

int main()
{
    int i = 1951;

    printf("%d\n", i);
    return 0;
}
```

- Código imprime 1951 na tela





Float

- Para a usar casas decimais, podemos declarar variáveis float (precisão simples) ou double (precisão dupla)
- `float f = 3.14;`
- `double d = 1.84;`
- Float possui 32 bits, double 64 bits. Ou seja, double consegue trabalhar com mais casas decimais. Nenhum deles é igual a número real





Float

- Para imprimir esses tipos, utilizamos “%f” no printf:

```
#include <stdio.h>

int main()
{
    float f = 3.14;
    double d = 1.81;

    printf("%f %lf\n", f, d);
    return 0;
}
```

Saída: 3.140000 1.810000





Float

- Podemos alterar o número de casas decimais:

```
#include <stdio.h>

int main()
{
    float f = 3.14;
    double d = 1.81;

    printf("%.5f %.3lf\n", f, d);
    return 0;
}
```

Saída: 3.14000 1.810





Char

- Também queremos representar letras nos programas de computador
- Para isso, utilizamos o tipo “char” para representar um caractere
- As letras são atribuídas com aspas simples





Char

- Exemplo:

```
#include <stdio.h>

int main()
{
    char c = 'A';

    printf("%c\n", c);
    return 0;
}
```

Saída: A





Char

- Na verdade, as letras são representadas como números em um computador
- Uma tabela padrão conhecida como Tabela ASCII ilustra quais números representam quais caracteres





Char

```
[user@station]$ ascii -d
```

0	NUL	16	DLE	32		48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL



Char

- Exemplo, imprimindo 2 caracteres:

```
#include <stdio.h>

int main()
{
    char c1 = 'A';
    char c2 = 80;

    printf("%c%c\n", c1, c2);
    return 0;
}
```

Saída: AP





Char

- Nota: Apesar de ser aceito na linguagem C, é preferível que se inicialize caracteres com aspas simples
- Dessa forma, o código fica mais legível para humanos
- Isso é considerado uma “boa prática de programação”





Char

- Em APC, não trabalharemos com caracteres acentuados: ã, ç, ó, etc...
- Recomendação: nunca utilize acentos no código da nossa disciplina. Apenas caracteres ASCII
- Para mais informações, veja seção 2.3 do livro <https://riscv-programming.org/book/riscv-book.html#pf1f>





Lendo dados do teclado

- A leitura de dados do teclado pode ser realizada de diversas formas
- Para ler apenas um dado do teclado, é possível utilizar a função “getchar”





Lendo dados do teclado

- A função `getchar` não recebe parâmetros e retornar o caractere lido (ou um erro, que vamos aprender a tratar mais na frente)

```
#include <stdio.h>

int main()
{
    char c1;

    c1 = getchar();
    printf("%c\n", c1);
    return 0;
}
```





Lendo dados do teclado

- É possível realizar a leitura de dados de uma forma semelhante ao printf, usando a função scanf
- No entanto, as variáveis que são passadas como argumento precisam receber o operador &
- Iremos entender o uso desse operador quando aprenderemos passagem por referência e parâmetro...





Lendo dados do teclado

- Uso de scanf:

```
#include <stdio.h>

int main()
{
    int i;
    char c;

    scanf("%c", &c);
    scanf("%d", &i);
    printf("%c\n", c);
    printf("%d\n", i);

    return 0;
}
```





Lixo de memória

- Note que é possível declarar variáveis sem inicializar
- Também é possível USAR variáveis sem inicializar
- Nesse caso, estamos lendo “lixo de memória” e seu programa pode imprimir diferentes resultados





Lixo de memória

- Imprimindo lixo de memória:

```
#include <stdio.h>

int main()
{
    int i;

    printf("%d\n", i);
    return 0;
}
```





Lixo de memória

- Usar variáveis não inicializadas é um dos erros mais comuns que os programadores iniciantes cometem
- É um problema difícil de detectar, afinal o comportamento é não-determinístico





Reutilizando Variáveis

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

```
int main()  
{
```

```
    int i = 0;  
    int y;
```

```
    printf("Valor inicial %d\n", i);
```

```
    y = i;
```

```
    printf("Valor intermediario %d %d\n", i, y);
```

```
    i = 6;
```

```
    printf("Valor final %d %d\n", i, y);
```

```
    return 0;
```

```
}
```

Endereço
de
Memória

0x7f..6
8

0x7f..6
0

Valor

LIXO

LIXO

Variável

Saída:






Reutilizando Variáveis

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

```
int main()
```

```
{
```



```
int i = 0;  
int y;
```

```
printf("Valor inicial %d\n", i);
```

```
y = i;
```

```
printf("Valor intermediario %d %d\n", i, y);
```

```
i = 6;
```

```
printf("Valor final %d %d\n", i, y);
```

```
return 0;
```

```
}
```

Endereço
de
Memória

0x7f..6
8

0x7f..6
0

Valor

0

LIXO

Variável

i

Saída:



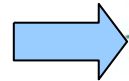


Reutilizando Variáveis

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

```
int main()  
{
```

```
    int i = 0;  
    int y;
```



```
    printf("Valor inicial %d\n", i);
```

```
    y = i;
```

```
    printf("Valor intermediario %d %d\n", i, y);
```

```
    i = 6;
```

```
    printf("Valor final %d %d\n", i, y);
```

```
    return 0;
```

```
}
```

Endereço
de
Memória

0x7f..6
8

0x7f..6
0

Valor

0

LIXO

Variável

i

y

Saída:





Reutilizando Variáveis

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

```
int main()  
{
```

```
    int i = 0;  
    int y;
```

→ `printf("Valor inicial %d\n", i);`

```
    y = i;
```

```
    printf("Valor intermediario %d %d\n", i, y);
```

```
    i = 6;
```

```
    printf("Valor final %d %d\n", i, y);
```

```
    return 0;
```

```
}
```

Endereço
de
Memória

0x7f..6 8
0x7f..6 0

Valor

0
LIXO

Variável

i
y

Saída:
Valor inicial 0





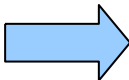
Reutilizando Variáveis

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

```
int main()  
{
```

```
    int i = 0;  
    int y;
```

```
    printf("Valor inicial %d\n", i);
```



```
    y = i;
```

```
    printf("Valor intermediario %d %d\n", i, y);
```

```
    i = 6;
```

```
    printf("Valor final %d %d\n", i, y);
```

```
    return 0;
```

```
}
```

Endereço
de
Memória

0x7f..6 8
0x7f..6 0

Valor

0
0

Variável

i
y

Saída:
Valor inicial 0





Reutilizando Variáveis

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

```
int main()  
{
```

```
    int i = 0;  
    int y;
```

```
    printf("Valor inicial %d\n", i);
```

```
    y = i;
```

```
    printf("Valor intermediario %d %d\n", i, y);
```

```
    i = 6;
```

```
    printf("Valor final %d %d\n", i, y);
```

```
    return 0;
```

```
}
```

Endereço
de
Memória

0x7f..6 8
0x7f..6 0

Valor

0
0

Variável

i
y

Saída:

Valor inicial 0

Valor intermediario 0 0



Reutilizando Variáveis

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

```
int main()  
{
```

```
    int i = 0;  
    int y;
```

```
    printf("Valor inicial %d\n", i);
```

```
    y = i;
```

```
    printf("Valor intermediario %d %d\n", i, y);
```

```
    i = 6;
```

```
    printf("Valor final %d %d\n", i, y);
```

```
    return 0;
```

```
}
```

Endereço
de
Memória

0x7f..6 8
0x7f..6 0

Valor

6
0

Variável

i
y

Saída:

Valor inicial 0

Valor intermediario 0 0



Reutilizando Variáveis

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

```
int main()  
{
```

```
    int i = 0;  
    int y;
```

```
    printf("Valor inicial %d\n", i);
```

```
    y = i;
```

```
    printf("Valor intermediario %d %d\n", i, y);
```

```
    i = 6;
```

```
    printf("Valor final %d %d\n", i, y);
```

```
    return 0;
```

```
}
```

Endereço
de
Memória

0x7f..6
8

0x7f..6
0

Valor

6

0

Variável

i

y

Saída:

Valor inicial 0

Valor intermediario 0 0

Valor final 6 0





Reutilizando Variáveis

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

```
int main()  
{
```

```
    int i = 0;  
    int y;
```

```
    printf("Valor inicial %d\n", i);
```

```
    y = i;
```

```
    printf("Valor intermediario %d %d\n", i, y);
```

```
    i = 6;
```

```
    printf("Valor final %d %d\n", i, y);
```

```
    return 0;
```

```
}
```

Endereço
de
Memória

0x7f..6
8

0x7f..6
0

Valor

6

0

Variável

i

y

Saída:

Valor inicial 0

Valor intermediario 0 0

Valor final 6 0





Modificadores

- Também é possível modificar as variáveis utilizando as palavras chaves:
 - signed ou unsigned;
 - short ou long.





Signed e Unsigned

- O modificador “signed” é o modo padrão da linguagem e por isso não é utilizado com frequência
- O modificador “unsigned” determina que a variável é utilizada apenas para valores positivos
- Utiliza-se o operador “u” no lugar do “d” para imprimir variáveis unsigned





Signed e Unsigned

- Os valores capazes de serem armazenados são alterados

Tipo	Bits	Valor
char	8	-128 a 127
unsigned char	8	0 a 255
int	32	-2.147.483.648 a 2.147.483.647
unsigned int	32	0 a 4.294.967.295



Short e Long

- O modificador “short” é utilizado para reduzir o número de bits que a variável possui
- Assim a capacidade e o uso da memória são reduzidos!
- O operador “long” faz o contrário: aumenta o número de bits e uso de memória





Short e Long

Tipo	Bits	Valor
short int	16	-32.768 a 32.767
long int	32 ou 64*	Se 64 bits: -9.223.372.036.854.775.808 9.223.372.036.854.775.807

Nota: algumas versões mais antigas podem usar 32

A versão que usaremos em aula é 64





Short e Long

- Para imprimir variáveis long usa-se o modificar “l” no printf:

```
#include <stdio.h>

int main()
{
    long int i = 9123456789012345678;

    printf("%ld\n", i);
    return 0;
}
```





Constantes

- Por último, podemos utilizar variáveis constantes, “const”
- Variáveis constantes devem ser obrigatoriamente inicializadas e não podem ser modificadas ao longo do código
- Não podemos ler dados do teclado, nem fazer atribuições nela





Constantes

```
#include <stdio.h>

int main()
{
    const float pi = 3.141593;

    printf("%f\n", pi);
    return 0;
}
```





Dúvidas?

- lucasxboaventura18@gmail.com

