



BC1424

Algoritmos e Estruturas de Dados I

Aula 04:

Ponteiros e estruturas

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

1Q-2015



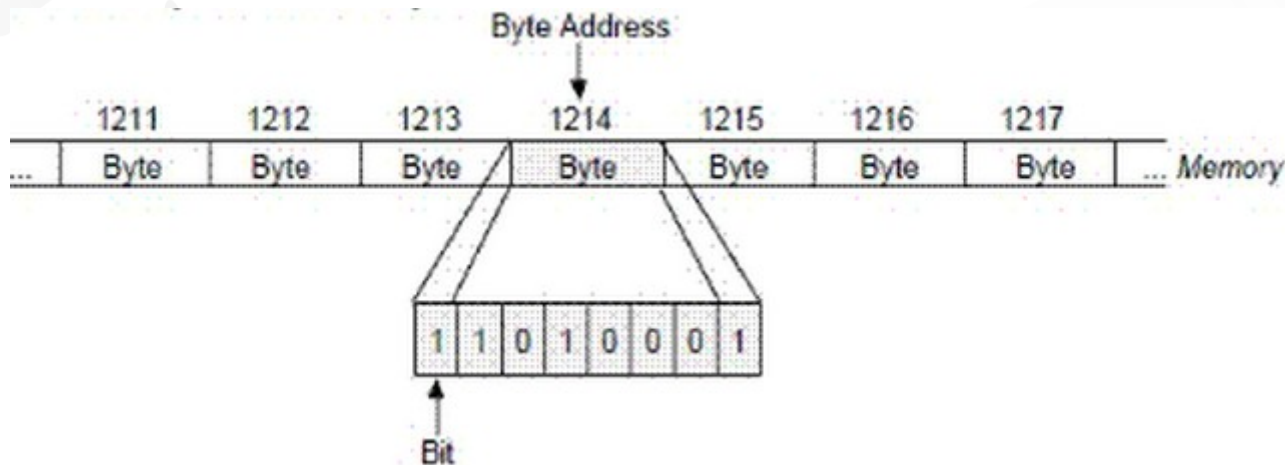
Endereços e ponteiros

Endereços e ponteiros

- **Os conceitos de endereço e ponteiro são fundamentais em qualquer linguagem de programação.**
- Na linguagem C é mais visível este conceito.
- Requer um esforço para usar os ponteiros.

Endereços

- A memória de qualquer computador (arquitetura de von Neumann) **é uma sequência de bytes**.
- Cada byte armazena um de 256 possíveis valores
- Os bytes são numerados sequencialmente e o número de um byte é o seu endereço

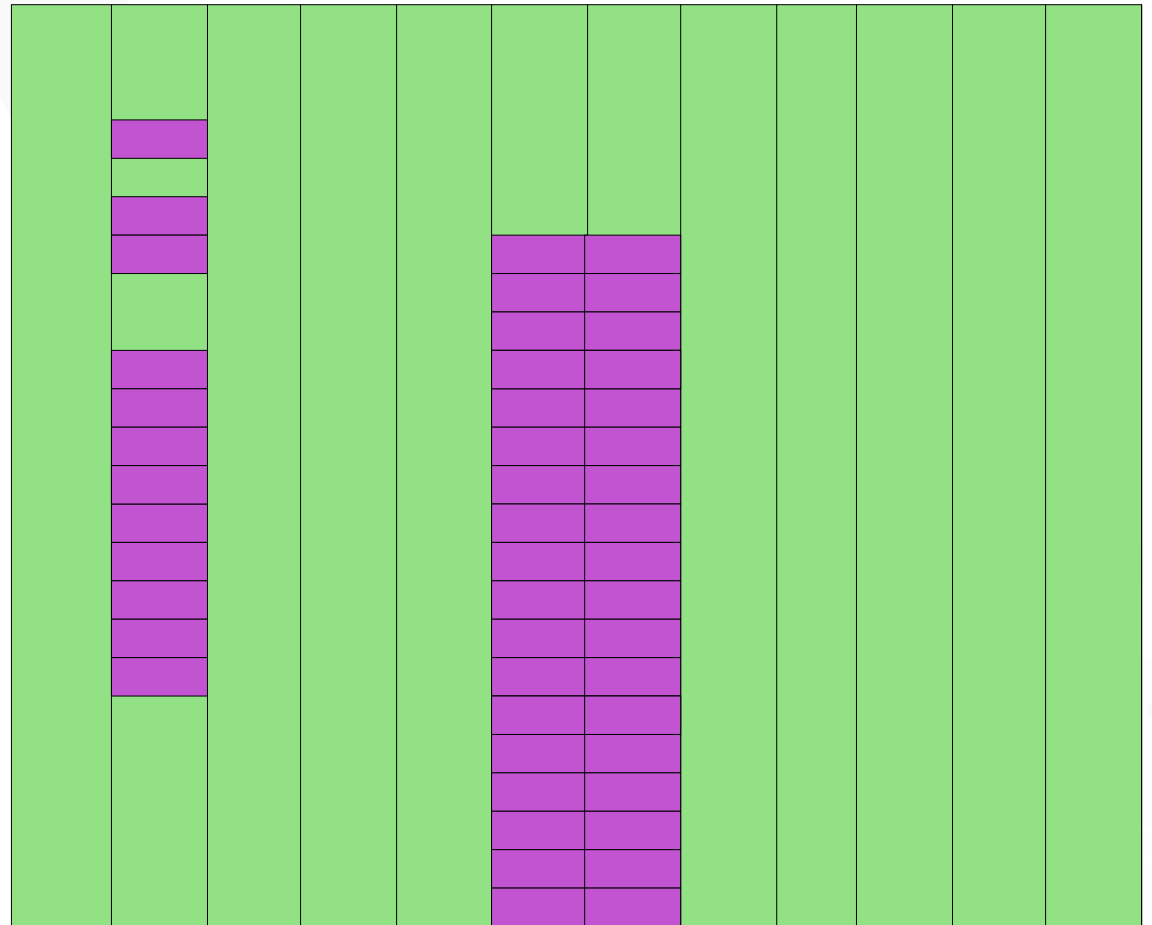


Endereços

	...
37FD00	01010111
37FD01	11000011
37FD02	01100100
37FD03	11100010
	...

Endereços

	...
37FD00	01010111
37FD01	11000011
37FD02	01100100
37FD03	11100010
	...



Cada objeto na memória do computador tem um endereço.

Tipos de datos

```
1 #include<stdio.h>
2
3 int main() {
4     printf("Size of char is      %ld bytes\n",sizeof(char));
5     printf("Size of short is     %ld bytes\n",sizeof(short));
6     printf("Size of int is       %ld bytes\n",sizeof(int));
7     printf("Size of long is      %ld bytes\n",sizeof(long));
8     printf("Size of float is     %ld bytes\n",sizeof(float));
9     printf("Size of double is    %ld bytes\n",sizeof(double));
10    printf("Size of long double is %ld bytes\n",sizeof(long double));
11    return 0;
12 }
```

```
jmenac@aed1:~/workspace/aed1-01 $ gcc tiposDeDatos.c -o tiposDeDatos && ./tiposDeDatos
```

```
Size of char is      1 bytes
Size of short is     2 bytes
Size of int is       4 bytes
Size of long is      8 bytes
Size of float is     4 bytes
Size of double is    8 bytes
Size of long double is 16 bytes
```

```
jmenac@aed1:~/workspace/aed1-01 $ uname -a
```

```
Linux jmenac-aed1-1261073 3.14.13-c9 #1 SMP Tue Aug 19 16:13:44 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
```

Endereços

37FD00

01010111

37FD01

11000011

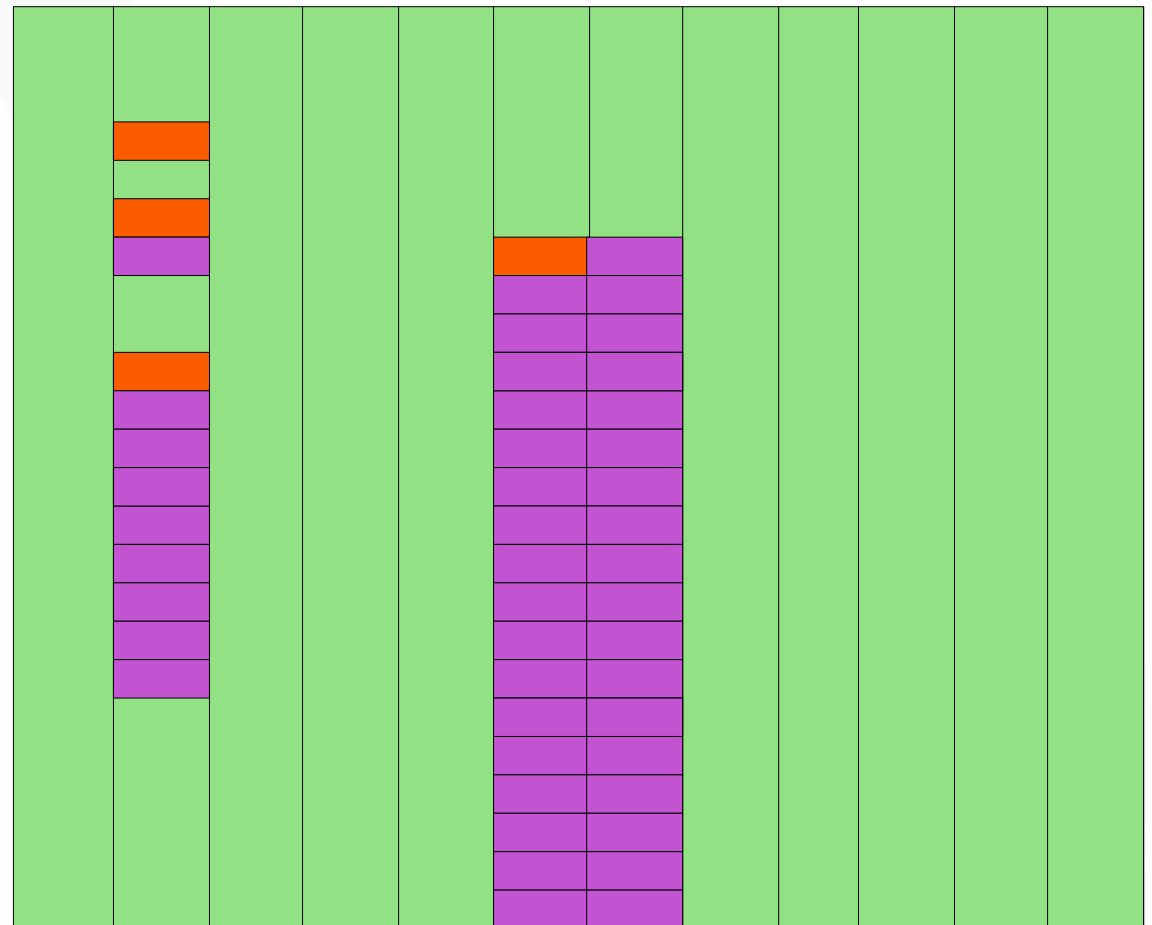
37FD02

01100100

37FD03

11100010

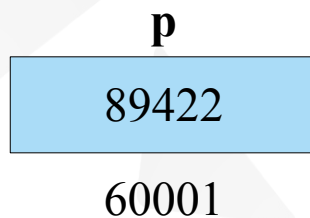
...
01010111
11000011
01100100
11100010
...



Geralmente o endereço do objeto é o endereço do 1ro byte.

Endereços

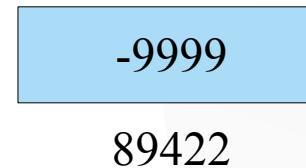
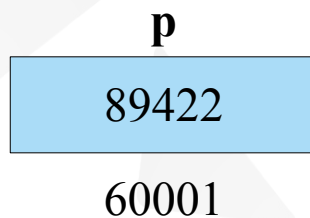
- Em **c** o endereço de um objeto é dado pelo operador **&**
- Se **x** é uma variável, então **&x** é o seu endereço



$p = 89422$
 $\&p = 60001$

Endereços

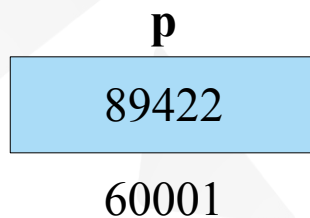
- Em **c** o endereço de um objeto é dado pelo operador **&**
- Se **x** é uma variável, então **&x** é o seu endereço



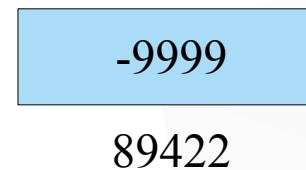
$p = 89422$
 $\&p = 60001$

Endereços

- Em **c** o endereço de um objeto é dado pelo operador **&**
- Se **x** é uma variável, então **&x** é o seu endereço



$p = 89422$
 $\&p = 60001$



$*p = -9999$

Endereços

- Em **c** o endereço de um objeto é dado pelo operador **&**
- Se **x** é uma variável, então **&x** é o seu endereço



“p aponta para a s”
“p é o endereço de s”
“p aponta a s”

$p = 89422$
 $\&p = 60001$

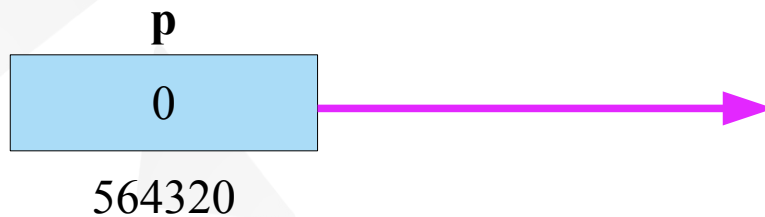
$*p = -9999$

$*p$ é o mesmo que
escrever “s”

Endereços

- Todo ponteiro pode ter o valor NULL.
- NULL é uma constante, geralmente vale 0 (definida no arquivo interface stdlib)

```
int *p = NULL;
```



Endereços

Há vários tipos de ponteiros:

- P. para caracteres
- P. para inteiros
- P. para registros
- P. para ponteiros para inteiros

`int* p ;`

← Um tipo de dado novo `int*` (conceitualmente correto)

`int *p;`

← O “*” modifica a variável e não o `int` (mais aceito)

`int * p;`

*O compilador C aceita
qualquer das formas.*

exemploPonteiro.c

```
1 #include<stdio.h>
2
3 int main() {
4     int x;
5     int i = 100;
6
7     int *p;           /* p é um ponteiro para um inteiro */
8     p = &i;           /* p aponta para i*/
9
10    x = *p+900;        /* o mesmo que x = i+900 */
11
12    printf("O valor de i      : %d\n", i);
13    printf("O endereco de i: %d\n", &i);
14    printf("O valor de p      : %d\n", p);
15    printf("O valor de x      : %d\n", x);
16
17 }
```

exemploPonteiro.c

```
Running /home/ubuntu/workspace/aed1-04/exemploPonteiro.c
/home/ubuntu/workspace/aed1-04/exemploPonteiro.c: In function 'main'
/home/ubuntu/workspace/aed1-04/exemploPonteiro.c:13:5: warning: form
    printf("O endereco de i: %d\n", &i);
    ^
/home/ubuntu/workspace/aed1-04/exemploPonteiro.c:14:5: warning: form
    printf("O valor de p    : %d\n", p);
    ^
O valor de i      : 100
O endereco de i: 1808239072
O valor de p      : 1808239072
O valor de x      : 1000
```


exemploPonteiro.c

```
printf("O endereco de i: %p\n", &i);  
printf("O valor de p    : %p\n", p);
```

```
Running /home/ubuntu/workspace/aed1-04/exemploPonteiro.c  
O valor de i    : 100  
O endereco de i: 0x7fffc8aff3e0  
O valor de p    : 0x7fffc8aff3e0  
O valor de x    : 1000
```

exemploPonteiro.c

%d	%i	Decimal signed integer.
%o		Octal integer.
%X	%X	Hex integer.
%u		Unsigned integer.
%c		Character.
%s		String. See below.
%f		double
%e	%E	double.
%g	%G	double.
%p		pointer.

```
• int num_students;  
  scanf("%d", &num_students);
```

Specifier for
"reading an integer value"

VERY IMPORTANT
special symbol

"Place value into this
variable"

exemploPonteiro2.c

```
1 #include<stdio.h>
2
3 int main() {
4     int i;
5     i = 100;
6
7     printf("%d\n", *&i);
8
9 }
```

ControlString

```
printf("a = %d\nb = %d\n", a, b);
```

1st Line 2nd Line *arg1* *arg2*

New Line New Line

The diagram illustrates the execution of the `printf` function. The format string `"a = %d\nb = %d\n"` is shown with annotations. A dashed orange line labeled *ControlString* spans the entire string. A red arrow points from the first `\n` to the first argument `a`, and a blue arrow points from the second `\n` to the second argument `b`. The first argument `a` is labeled *arg1* and the second `b` is labeled *arg2*. The string is divided into two parts by the first `\n`: `"a = %d"` is underlined in red and labeled 1st Line, and `"b = %d\n"` is underlined in blue and labeled 2nd Line. Two purple arrows point upwards to the `\n` characters, each labeled "New Line".

exemploPonteiro3.c

```
1 #include<stdio.h>
2
3 void troca(int i, int j) {
4     int temp;
5     temp = i;
6     i = j;
7     j = temp;
8 }
9
10 int main() {
11     int a=1;
12     int b=10;
13
14     troca(a,b);
15
16     printf("\ta=%d\n\tb=%d", a, b);
17 }
```

exemploPonteiro3.c

```
1 #include<stdio.h>
2
3 void troca(int *i, int *j) {
4     int temp;
5     temp = *i;
6     *i = *j;
7     *j = temp;
8 }
9
10 int main() {
11     int a=1;
12     int b=10;
13
14     troca(&a,&b);
15
16     printf("\ta=%d\n\tb=%d", a, b);
17 }
```

exemploPonteiro3.c

```
Running /home/ubuntu/workspace/aed1-04/exemploPonteiro3.c  
a=10  
b=1
```


exemploPonteiro3.c

Por que o código abaixo está errado?

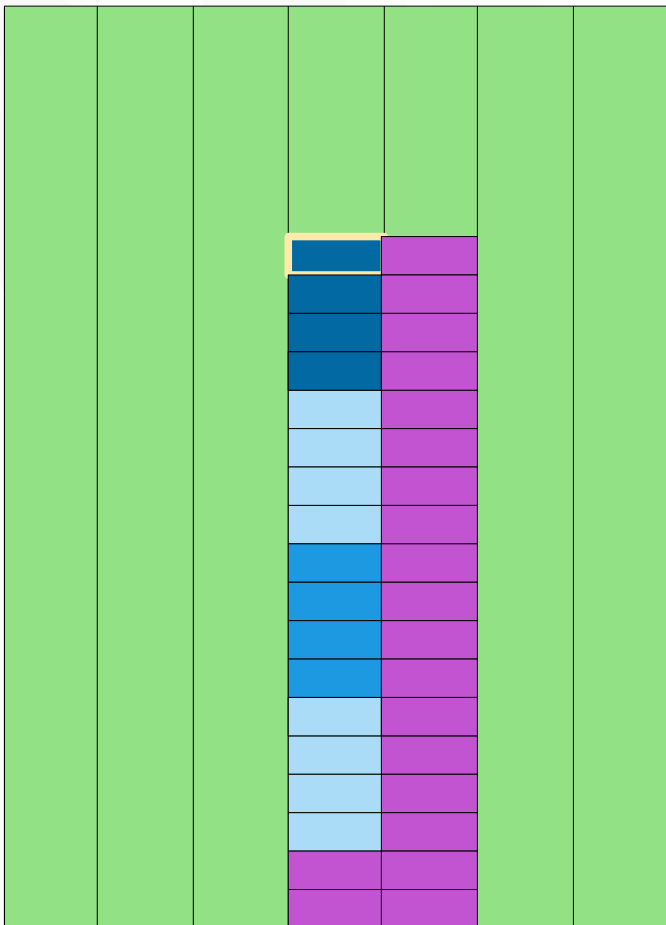
```
void troca(int *i, int *j) {  
    int *temp;  
    *temp = *i;  
    *i = *j;  
    *j = *temp;  
}
```



Vetores e endereços

Vetores

Os elementos de um vetor são alocados consecutivamente na memória do computador.

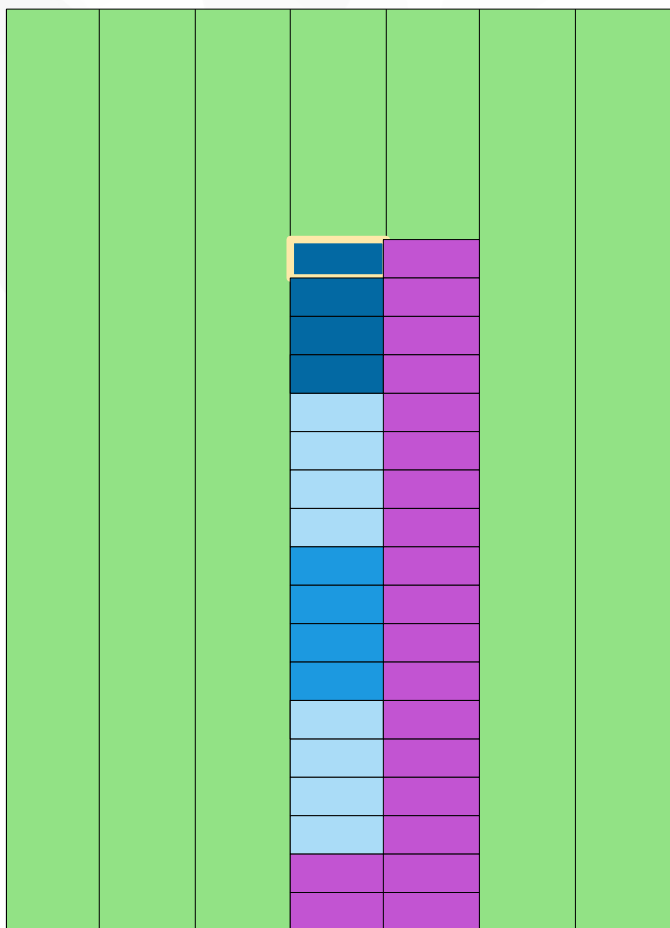


Se cada elemento ocupa **b** bytes, a diferença entre os endereços de dois elementos consecutivos será de **b**.

(ex. inteiros ocupam 4 bytes, em uma plataforma de 64 bits)

Vetores

Os elementos de um vetor são alocados consecutivamente na memória do computador.



Se cada elemento ocupa **b** bytes, a diferença entre os endereços de dois elementos consecutivos será de **b**.

O compilador C cria a ilusão de que **b** vale 1 qualquer que seja o tipo dos elementos do vetor.

exemploVetor.c

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main() {
5     int i;
6     int *V;
7     V = malloc(100*sizeof(int));
8
9     V[0] = 10;
10    V[1] = 20;
11    V[2] = 30;
12
13    for(i=0; i<5; i++)
14        printf("%d\n", V[i]);
15
16 }
```

exemploVetor.c

```
Running /home/ubuntu/workspace/aed1-04/exemploVetor.c
/home/ubuntu/workspace/aed1-04/exemploVetor.c: In function 'main':
/home/ubuntu/workspace/aed1-04/exemploVetor.c:14:9: warning: format '%d' expects an int argument but 1 has type 'char*'
    printf("%d\n", V+i);
    ^
10903568
10903572
10903576
10903580
10903584
```

exemploVetor.c

```
for(i=0; i<5; i++)  
    printf("%d\n", *(V+i));    // v[i]
```

```
Running /home/ubuntu/workspace/aed1-04/exemploVetor.c
```

```
10
```

```
20
```


```
30
```

```
0
```

```
0
```

exemploVetor2.c

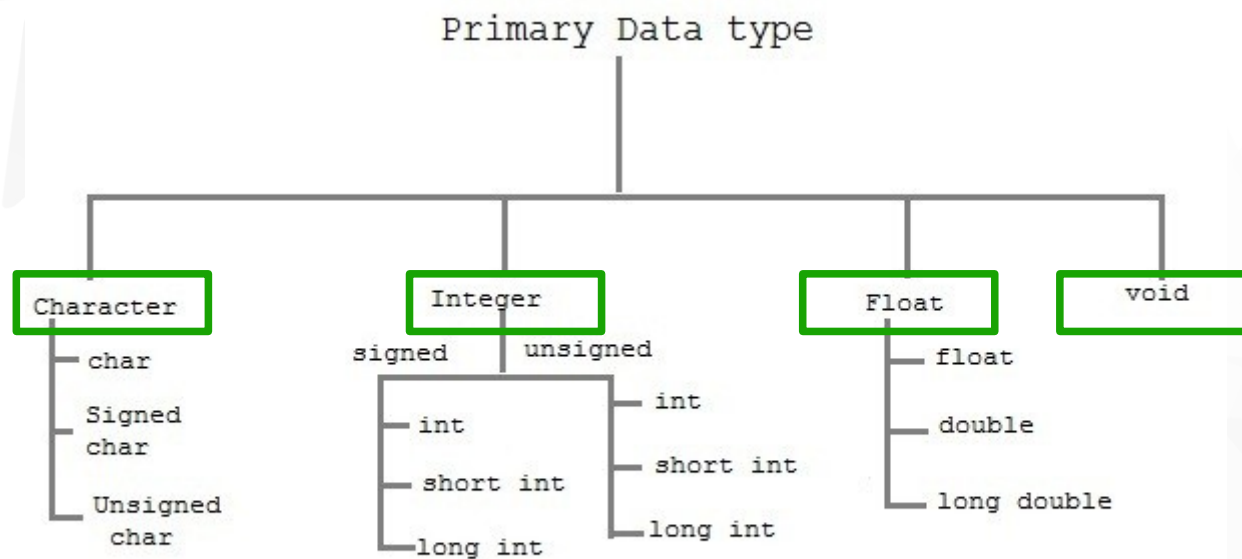
```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main() {
5     int i;
6     int *V;
7     V = malloc(100*sizeof(int));
8
9     for(i=0; i<5; i++)
10         scanf("%d", V+i);    // &V[i]
11
12     for(i=0; i<5; i++)
13         printf("%d\n", *(V+i));
14
15 }
```

Estruturas (=registros)

Linguagem C: Tipos de dados

- **Tipos de dados primários.**
- Tipos de dados derivados.
- Tipos definidos pelo usuário.



Estruturas

```
1 #include<stdio.h>
2
3 int main()
4 {
5     struct ponto3D {
6         double x;
7         double y;
8         double z;
9     };
10
11     struct ponto3D p1;    /*um registro p1 do tipo ponto3D*/
12
13     printf("%ld\n", sizeof(p1));
14     printf("%f %f %f\n", p1.x, p1.y, p1.z);
15
16 }
```

Estruturas

```
Running /home/ubuntu/workspace/aed1-04/estrutura.c
```

```
24
```

```
0.000000 0.000000 0.000000
```

```
1 #include <stdio.h>
2 #include <math.h>
3
4 struct ponto3D {
5     double x;
6     double y;
7     double z;
8 };
9
10 double dEuclidiana(struct ponto3D p, struct ponto3D q) {
11     return sqrt(pow(p.x-q.x,2) + pow(p.y-q.y,2) + pow(p.z-q.z,2));
12 }
13
14 int main() {
15     struct ponto3D p1, p2;
16
17     p1.x = p1.y = p1.z = 0;
18     p2.x = p2.y = p2.z = 10;
19
20     printf("%f\n", dEuclidiana(p1,p2) );
21 }
```

```
Running /home/ubuntu/workspace/aed1-04/estrutura.c
/tmp/ccW0ohTu.o: In function `dEuclidiana':
estrutura.c:(.text+0x2a): undefined reference to `pow'
estrutura.c:(.text+0x55): undefined reference to `pow'
estrutura.c:(.text+0x89): undefined reference to `pow'
estrutura.c:(.text+0x93): undefined reference to `sqrt'
collect2: error: ld returned 1 exit status
```





```
jmenac@aed1:~/workspace/aed1-04 $ gcc estrutura.c -o estrutura -lm && ./estrutura
17.320508
```

Estruturas



```
1 #include <stdio.h>
2
3 struct ponto3D {
4     double x;
5     double y;
6     double z;
7 };
8
9 int main() {
10     struct ponto3D *ptr; /* ptr é um ponteiro para registros ponto3D */
11     struct ponto3D p1;
12
13     ptr = &p1; /* ptr aponta a p1 */
14
15     (*ptr).x = 10; /* mesmo efeito que p1.x=10 */
16
17     ptr->x = 10; /* ptr->x é uma abreviatura (*ptr).x */
18 }
```



Sobre a lista de exercícios

0 ✓ 0.02s : Success  (sample)
1 ✓ 0.02s : Success 
2 ✓ 0.02s : Success 
3 ✓ 0.02s : Success 

4 ✓ 0.02s : Success 
5 ✓ 0.02s : Success 
6 ✓ 0.02s : Success 
7 ✓ 0.02s : Success 

8 ✓ 0.02s : Success 
9 ✓ 0.02s : Success 

```
6 unsigned long int flip(unsigned long int n) {
7     int V[32] = {0};
8     int pos=31, r;
9     unsigned long int n2=0, base=1;
10
11     do {
12         r = n%2;
13         n = n/2;
14         V[pos--] = r;
15     } while(n>=1);
16
17     for (pos=31; pos>=0; pos--) {
18         n2 += (1-V[pos])*base;
19         base *= 2;
20     }
21     return n2;
22
23 }
```

```
25 int main()
26 {
27     int t, i;
28     unsigned long int item;
29
30     scanf("%d",&t);
31
32     for(i=0; i<t; i++) {
33         scanf("%ld", &item);
34         printf("%ld\n", flip(item));
35     }
36
37     return 0;
38 }
```

Lista 02: <https://www.hackerrank.com>

- Utopian tree
- Max Min (antigo Angry Children)
- Halloween party
- Is Fibo

Será utilizado um programa de detecção de plágio em todas as submissões!
Plágio → reprovação

Data: 01/Março (domingo) até às 23h50.

Envio: Através do Tidia.

Arquivos:

Para cada exercício-problema deverá submeter:

- O código fonte: nome do arquivo → RA_nomeDoProblema.c
- O comprovante de aceitação (screenshot) → RA_nomeDoProblema.pdf

Exemplo: 10123456_solveMeFirst.c
10123456_solveMeFirst.pdf