UFV
Universidade Federal de Viçosa

INF 112 - Programação II

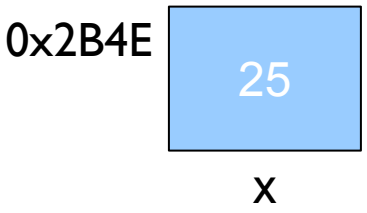
Revisão - Ponteiros

Ponteiros

- Uma variável ponteiro armazena um endereço de memória e normalmente é utilizada para referenciar um valor de forma indireta

Ponteiros

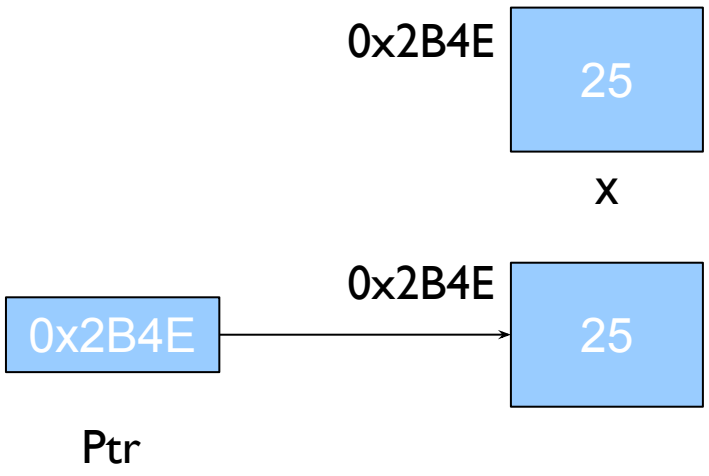
- Uma variável ponteiro armazena um endereço de memória e normalmente é utilizada para referenciar um valor de forma indireta



x é uma variável que fica na posição de memória 0x2B4E e armazena o número inteiro 25. A variável é referenciada de forma direta

Ponteiros

- Uma variável ponteiro armazena um endereço de memória e normalmente é utilizada para referenciar um valor de forma indireta



x é uma variável que fica na posição de memória 0x2B4E e armazena o número inteiro 25. A variável é referenciada de forma direta

xPtr é uma variável que armazena o endereço de memória 0x2B4E, onde o número inteiro 25 está armazenado. A variável é referenciada de forma indireta.

Ponteiros

- Assim como qualquer variável, um ponteiro precisa ser declarado
- Para isso, deve-se colocar um * antes do identificador da variável.

```
int *xPtr, x;   // Declara um apontador xPtr e uma
                // variável x
int *xPtr, *x; // tanto x quanto xPtr são ponteiros
                // para int
char *c;        // c é um ponteiro para char
float **x;      // O que é x?
```

Ponteiros

- Ponteiros podem ser inicializados com NULL
- Porém, na maioria das vezes, os ponteiros são inicializados com endereços
- Para se obter o endereço de uma variável, basta utilizar o operador &

```
int *xPtr, x;
x = 8;
xPtr = NULL; // xPtr aponta para “nada”
xPtr = &x;   // xPtr aponta para o endereço da
             // variável x
```

Ponteiros

- Exemplo de funcionamento de ponteiros.
- Vamos supor que char utiliza 1 byte na memória.

```
char *ptr, c1, c2;
ptr = NULL;
c1 = 2;
c2 = 5;
ptr = &c1;
ptr = &c2;
```

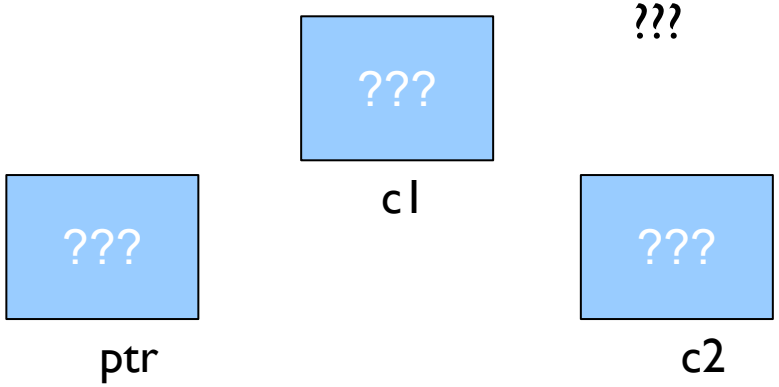
Variável associada	Endereço	Valor
???	00110	???
???	00111	???
???	01000	???
???	01001	???

Ponteiros

- Exemplo de funcionamento de ponteiros.
- Vamos supor que char utiliza 1 byte na memória.

```
char *ptr, c1, c2;
ptr = NULL;
c1 = 2;
c2 = 5;
ptr = &c1;
ptr = &c2;
```

Variável associada	Endereço	Valor
ptr	00110	???
c1	00111	???
c2	01000	???
???	01001	???

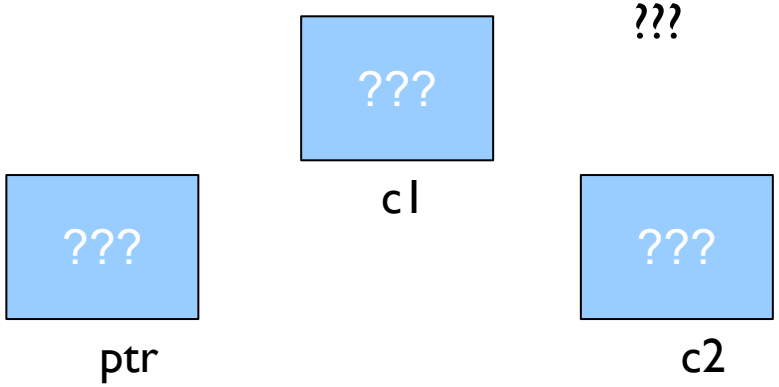


Ponteiros

- Exemplo de funcionamento de ponteiros.
- Vamos supor que char utiliza 1 byte na memória.

```
char *ptr, c1, c2;
ptr = NULL;
c1 = 2;
c2 = 5;
ptr = &c1;
ptr = &c2;
```

Variável associada	Endereço	Valor
ptr	00110	00000000
c1	00111	???
c2	01000	???
???	01001	???

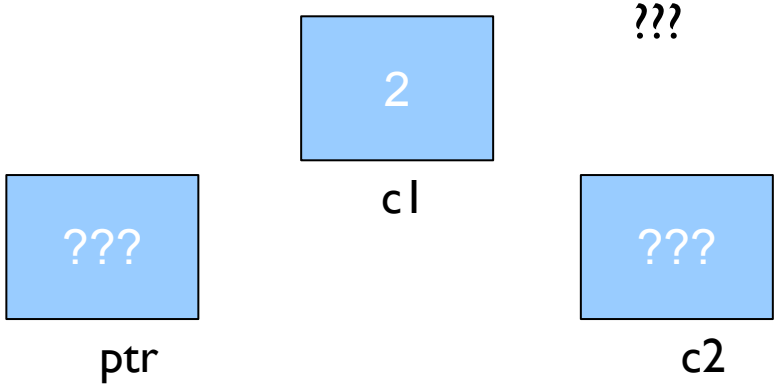


Ponteiros

- Exemplo de funcionamento de ponteiros.
- Vamos supor que char utiliza 1 byte na memória.

```
char *ptr, c1, c2;
ptr = NULL;
c1 = 2;
c2 = 5;
ptr = &c1;
ptr = &c2;
```

Variável associada	Endereço	Valor
ptr	00110	00000000
c1	00111	00000010
c2	01000	???
???	01001	???

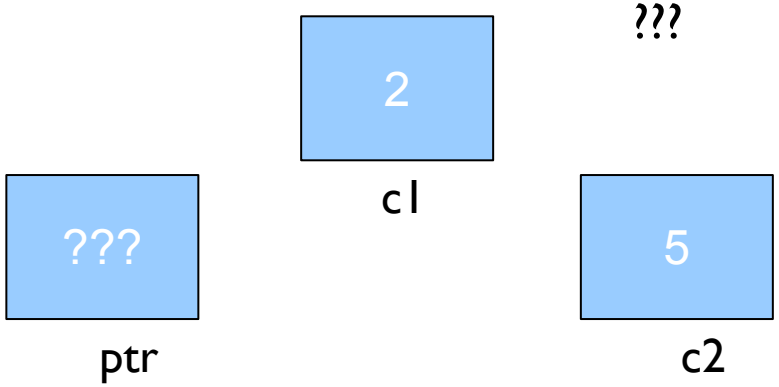


Ponteiros

- Exemplo de funcionamento de ponteiros.
- Vamos supor que char utiliza 1 byte na memória.

```
char *ptr, c1, c2;
ptr = NULL;
c1 = 2;
c2 = 5;
ptr = &c1;
ptr = &c2;
```

Variável associada	Endereço	Valor
ptr	00110	00000000
c1	00111	00000010
c2	01000	00000101
???	01001	???

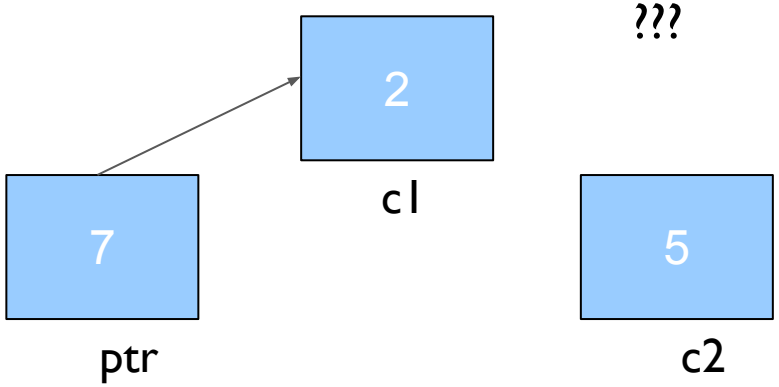


Ponteiros

- Exemplo de funcionamento de ponteiros.
- Vamos supor que char utiliza 1 byte na memória.

```
char *ptr, c1, c2;
ptr = NULL;
c1 = 2;
c2 = 5;
ptr = &c1;
ptr = &c2;
```

Variável associada	Endereço	Valor
ptr	00110	00000111
c1	00111	00000010
c2	01000	00000101
???	01001	???

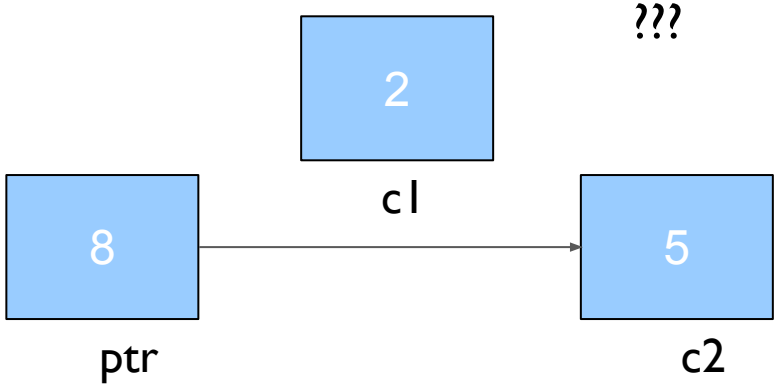


Ponteiros

- Exemplo de funcionamento de ponteiros.
- Vamos supor que char utiliza 1 byte na memória.

```
char *ptr, c1, c2;
ptr = NULL;
c1 = 2;
c2 = 5;
ptr = &c1;
ptr = &c2;
```

Variável associada	Endereço	Valor
ptr	00110	00001000
c1	00111	00000010
c2	01000	00000101
???	01001	???



Ponteiros

- Neste exemplo utilizamos endereços com 1 byte (por isso o ponteiro utiliza 1 byte de memória). Porém, o tamanho do ponteiro NÃO é relacionado com o tamanho do dado para o qual ele aponta (neste caso, char)!
- Normalmente ponteiros utilizam 4 ou 8 bytes de memória
- **Exercício**: Um ponteiro que ocupa 4 bytes consegue "apontar" para quantas posições distintas da memória?

Ponteiros

- Outro operador importante é o operador *, que é utilizado para acessar a variável apontada por um ponteiro.
- Essa operação normalmente é chamada de "derreferência".

```
int *ptr, c1;
c1 = 2;
ptr = &c1;
*ptr = 8;
cout << ptr << endl;
cout << *ptr << endl;
cout << c1 << endl;
```

Ponteiros

- Outro operador importante é o operador *, que é utilizado para acessar a variável apontada por um ponteiro.
- Essa operação normalmente é chamada de "derreferência".

```
int *ptr, c1;
c1 = 2;
ptr = &c1;
*ptr = 8; //grava 8 na variável apontada por ptr (c1)
cout << ptr << endl; //Imprime o valor de ptr (endereço)
cout << *ptr << endl; //Imprime o valor apontado por ptr
cout << c1 << endl; //Imprime o valor de c1
```

```
0x7fff02db139c
8
8
```

Ponteiros

- Atenção: o símbolo * pode ser utilizado em várias situações: multiplicação, declaração de apontador, derreferenciação de variável.

```
int *ptr;
int x = 9;
ptr = &x;
*ptr = *ptr**ptr; /* comentário com * */
```

Ponteiros

- Atenção: o símbolo & também pode ser utilizado em várias situações: obtenção de endereço e declaração de referência.

```
int *ptr;
int x = 9;
ptr = &x;    //Obtenção de endereço
int &z = x; //z é um referência para x
```

Ponteiros

- Qual é o problema nos trechos de código abaixo?

```
int *ptr;
*ptr = 10;
cout << *ptr << endl;
```

```
int *ptr;
ptr = NULL;
cout << *ptr << endl;
```

Ponteiros

- O que o trecho de código abaixo faz?

```
int a, b;
int *x;
int **y;
int ***z;
a = 1;
b = 2;
x = &a;
y = &x;
z = &y;
cout << ***z << endl;
*y = &b;
cout << ***z << endl;
```

Ponteiros

- Ponteiros podem ser utilizados para acessar variáveis declaradas em outra função.
- Há algum problema no código abaixo? Como corrigi-lo?

```
void troca(int a, int b) {
    int aux = a;
    a = b;
    b = aux;
}
…
int a;
int b;
…
troca(a, b);
```

Ponteiros

- Ponteiros podem ser utilizados para acessar variáveis declaradas em outra função.
- Há algum problema no código abaixo? Como corrigi-lo?

```
void troca(int &a, int &b) {
    int aux = a;
    a = b;
    b = aux;
}
…
int a;
int b;
…
troca(a, b);
```

Ponteiros

- Ponteiros podem ser utilizados para acessar variáveis declaradas em outra função.
- Há algum problema no código abaixo? Como corrigi-lo?

```
void troca(int *a, int *b) {
    int aux = *a;
    *a = *b;
    *b = aux;
}
…
int a;
int b;
…
troca(&a, &b); // <---- Atenção!
```

Ponteiros

- Ponteiros podem ser utilizados para acessar variáveis declaradas em outra função.

- **Exercício:** implemente uma função "quadrado", que eleva o argumento ao quadrado (pense em várias possibilidades de implementação).

Ponteiros

- Como trabalhar com ponteiros para structs?

```
struct Jogador {
    int pontos;
    int x,y;
}


…
Jogador j;
Jogador *ptr = &j;
(*ptr).x = 8;
(*ptr).y = 2;
//ou...
ptr->x = 1;
ptr->y = 9;
```

Ponteiros e Arranjos

- Quando declaramos um arranjo, o identificador desse arranjo é um apontador (**constante**) para a primeira posição do arranjo.

```
int vetor[5];
int *ptr = &(vetor[0]);
cout << vetor << endl;
cout << ptr << endl;


…
int z;
vetor = &z;
```

Ponteiros e Arranjos

- Quando declaramos um arranjo, o identificador desse arranjo é um apontador (**constante**) para a primeira posição do arranjo.

```
int vetor[5];
int *ptr = &(vetor[0]);
cout << vetor << endl;
cout << ptr << endl;

…
int z;
vetor = &z; //Errro! o apontador é constante!
```

Ponteiros e Arranjos

- Podemos aplicar determinadas operações aritméticas a ponteiros.
- O que o código abaixo imprime??

```
int vetor[5] = {10,20,30,40,50};
int *ptr = &(vetor[0]);
cout << vetor[0] << endl;
cout << *ptr << endl;
cout << *(ptr+1) << endl;
ptr+= 2;
cout << *ptr << endl;
ptr++;
cout << *ptr << endl;
```

Ponteiros e Arranjos

- Podemos aplicar determinadas operações aritméticas a ponteiros.
- Note que, após adicionar 1 a um ponteiro, a posição da memória apontada por ele não é, necessariamente, deslocada em 1 byte (ela é deslocada em sizeof(tipo apontado pelo ponteiro) bytes).

```
int vetor[5] = {10,20,30,40,50};
int *ptr = &(vetor[0]);
cout << vetor[0] << endl;
cout << *ptr << endl;
cout << *(ptr+1) << endl;
ptr+= 2;
cout << *ptr << endl;
ptr++;
cout << *ptr << endl;
```

```
10
10
20
30
40
```

Ponteiros e Arranjos

- Podemos aplicar determinadas operações aritméticas a ponteiros.

```
int vetor[5] = {10,20,30,40,50};
for(int i=0;i<5;i++)
    cout << vetor[i] << endl;

for(int i=0;i<5;i++)
    cout << *(vetor+i) << endl;
```

Ponteiros e Arranjos

- Podemos aplicar determinadas operações aritméticas a ponteiros.

```
int vetor[5] = {10,20,30,40,50};
for(int i=0;i<5;i++)
    cout << vetor[i] << endl;

for(int i=0;i<5;i++)
    cout << *(vetor+i) << endl;
```

```
10
20
30
40
50
10
20
30
40
50
```

Ponteiros e Arranjos

- **Exercício:** com base no que você viu, o que o código abaixo imprime?

```
int a[10];

cout << (long)(a+1) - (long)(a) << endl;
```

Ponteiros e Arranjos

- **Exercício:** mostre diferentes formas de se varrer o vetor abaixo e imprimi-lo (pelo menos duas das formas devem utilizar ponteiros).

```
int a[10];
```

Ponteiros e Arranjos

- **Exercício:** mostre diferentes formas de se varrer o vetor abaixo e imprimi-lo (pelo menos duas das formas devem utilizar ponteiros).

```
int a[10];
for(int i=0;i<10;i++)
    cout << a[i] << endl;
```

Ponteiros e Arranjos

- **Exercício:** mostre diferentes formas de se varrer o vetor abaixo e imprimi-lo (pelo menos duas das formas devem utilizar ponteiros).

```
int a[10];
for(int i=0;i<10;i++)
    cout << a[i] << endl;
for(int i=0;i<10;i++)
    cout << *(a+i) << endl;
```

Ponteiros e Arranjos

- **Exercício:** mostre diferentes formas de se varrer o vetor abaixo e imprimi-lo (pelo menos duas das formas devem utilizar ponteiros).

```
int a[10];
for(int i=0;i<10;i++)
    cout << a[i] << endl;
for(int i=0;i<10;i++)
    cout << *(a+i) << endl;
for(int *ptr = a;ptr!=(a+10);ptr++)
    cout << *ptr << endl;
```

Ponteiros e Arranjos

- No caso de “strings” em C, como elas terminam com um '\0' o processo de varrê-las utilizando um for pode ser simplificado.

```
char teste[] = "testando";
```

Ponteiros e Arranjos

- No caso de “strings” em C, como elas terminam com um '\0' o processo de varrê-las utilizando um for pode ser simplificado.

```
char teste[] = "testando";
for(char *c = teste; *c ; c++)
    cout << *c ;
cout << endl;
```

Ponteiros e Arranjos

- **Exercício**: para que serve a função abaixo? como ela funciona?

```
bool сравнение(char *c1, char *c2) {
   for(; *c1 && *c1 == *c2; c1++, c2++);
   return *c1 == *c2;
}
```

Ponteiros e Arranjos

- **Exercício**: encontre erros no código abaixo.

```
int *zptr;
int *aptr;
int number;
int z[5] = {5,8,7,9,6};
aptr = z;
number = aptr;
```

Ponteiros e Arranjos

- **Exercício**: encontre erros no código abaixo.

```
int *zptr;
int *aptr;
int number;
int z[5] = {5,8,7,9,6};
aptr = z;
number = *zptr;
```

Ponteiros e Arranjos

- **Exercício**: encontre erros no código abaixo.

```
int *zptr;
int *aptr;
int number;
int z[5] = {5,8,7,9,6};
aptr = z;
for(int i=0;i<=5;i++)
   cout << *(aptr+i) << endl;
```

Ponteiros e Arranjos

- **Exercício**: encontre erros no código abaixo.

```
int *zptr;
int *aptr;
int number;
int z[5] = {5,8,7,9,6};
int *end = &(z + 5);
for(; z != end; z++)
    cout << *z << endl;
```