Módulo 3

Ponteiros

Algoritmos e Estruturas de Dados II

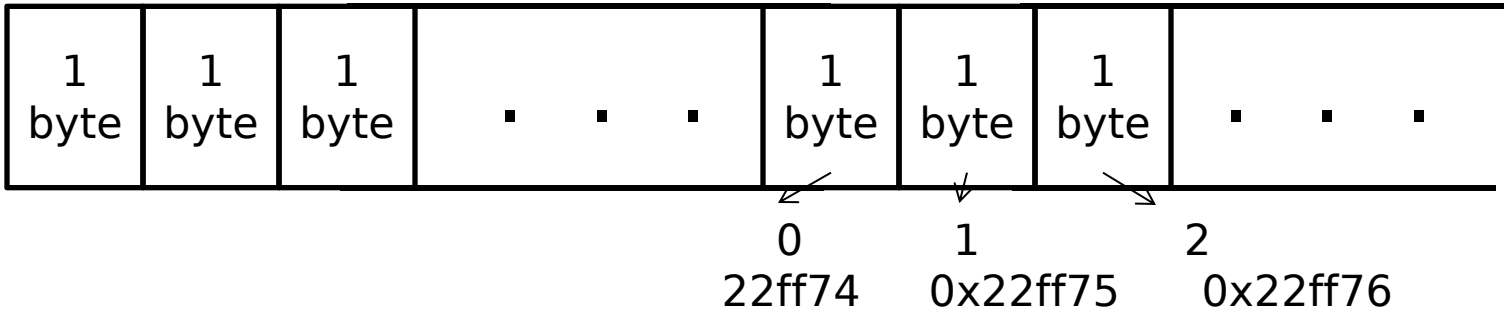
C++

(Rone Ilídio)

Memória do Computador

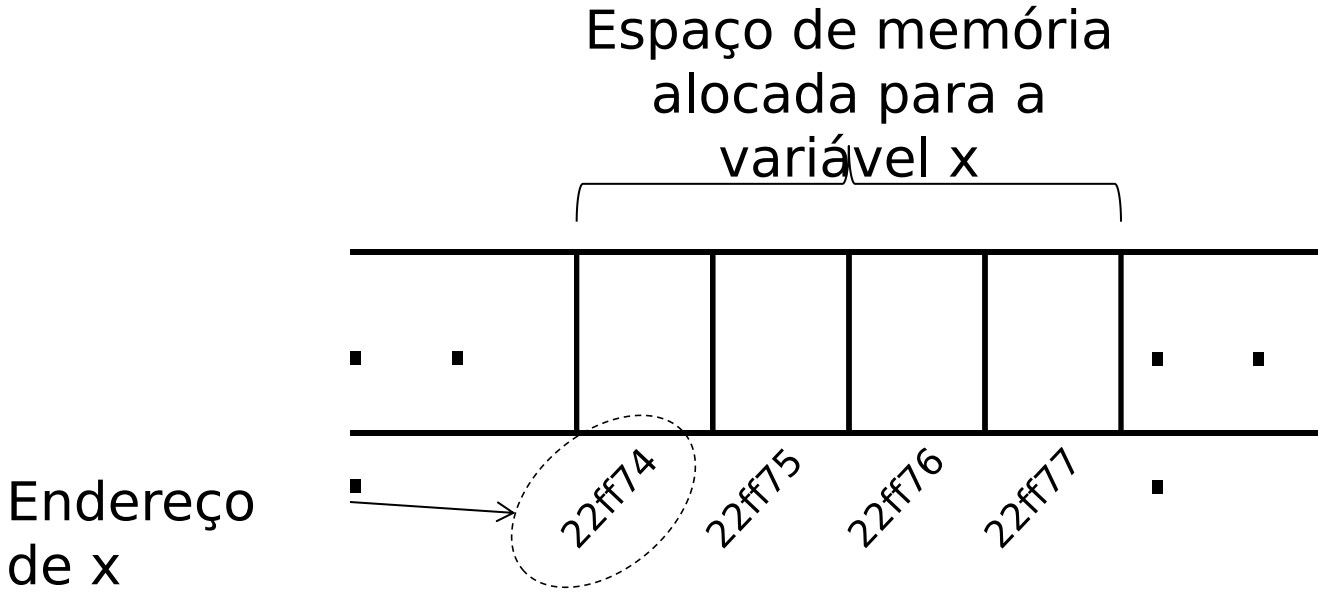
- Um vetor de bytes
- Cada byte com um número que é chamado de **endereço**
- Representação em hexadecimal

Obs: o endereço de uma variável é sempre o primeiro byte alocado para ela



Variável na memória

- Exemplo: int x = 10;
  - Inteiro ocupa 4 bytes
  - Endereço de x, ou &x = 22ff74



```
#include<iostream>
using namespace std;
int main(){
  int a=0, b=1, c=2;
  cout << "\nContudo de a:" <<a<< "  endereco:"<<&a;
  cout << "\nContudo de b:" <<b<< "  endereco:"<<&b;
  cout << "\nContudo de c:" <<c<< "  endereco:"<<&c;
}
```

Resultado

```
Contudo de a:0
endereco:0x22ff74
Contudo de b:1
endereco:0x22ff70
Contudo de c:2
endereco:0x22ff6c
```

Ponteiros

- Tipo especial de variável que armazena endereços de memória
- Criadas com o operador *
- Cada tipo possui um ponteiro diferente
  - int *p1 --> só pode receber endereços de memória onde foi armazenado um inteiro
  - double *p2 --> só pode receber endereços de memória onde foi armazenado um double
- Essa regra é válida para todos os tipos, primitivos ou não

Ponteiros

```
#include<iostream>
using namespace std;
int main(){
  int a=0;
  int *p;
  p = &a;
  cout << "\nContudo de a:" <<a<< "  endereco:"<<&a
    <<" contudo de p:"<<p;
}
```

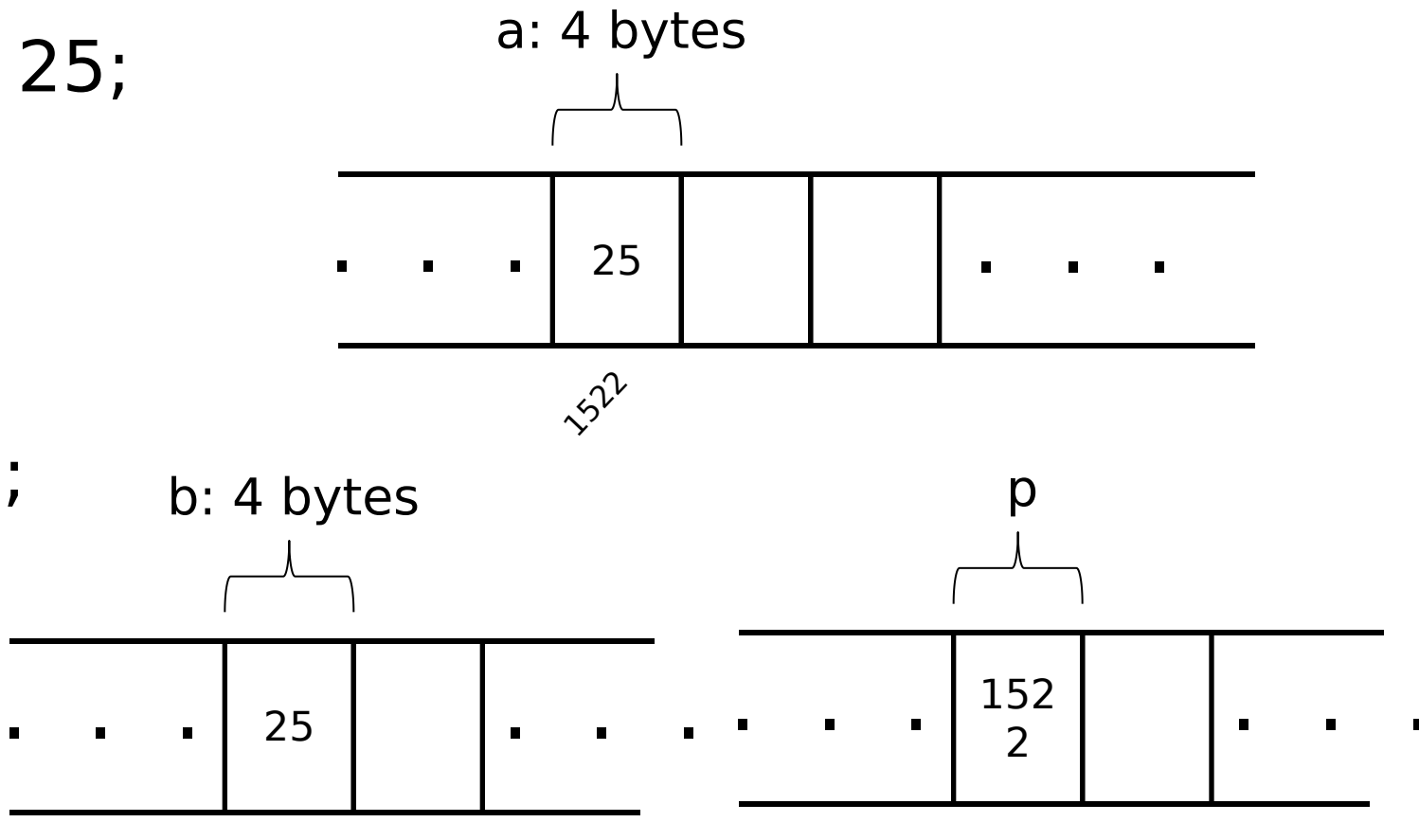
Contudo de a:0 endereco:0x22ff74 contudo de p:0x22ff74

Nomenclatura

- int y; --> declaração de uma variável inteira
- &y --> endereço de y
- int *p; --> declaração de um ponteiro para inteiros
- p = &y; --> o ponteiro p recebeu o endereço da variável y

Exemplo

```
int a = 25;
int b;
b = a;
int *p;
p = &a;
```



Valor apontado

- O operador * também é utilizado para exibir o valor que está sendo apontado.

```
#include<iostream>
using namespace std;
int main(){
  int x=10;
  int *p;
  p = &x;
  cout << "\nO que está sendo apontado por p:"
    <<*p;
}
```

Operações Com Ponteiros

- Ponteiros são valores que representam endereços de memória, por isso podem ser utilizados em operações aritméticas, como:
  - Atribuição
  - Adição
  - Subtração
  - Comparação
  - Incremento
  - Decremento
- As operações dependem do tipo apontado pelo ponteiro

Operações Com Ponteiros

- Suponha que a variável a (int) aloque a região de memória 1522 (o sistema operacional define o endereço, não o programador), então:

```
int *p = &a;  //p recebe o valor 1522
p = p + 1;    //o valor de p passa para 1526
                    pois ele aponta para um inteiro
                    e inteiros utilizam 4 bytes

cout << p;
```

Operações Com Ponteiros

```
#include <iostream>
using namespace std;
int main(){
 int x = 5;
 int *p = &x;
 cout << p << "  " << p+1;
}
```

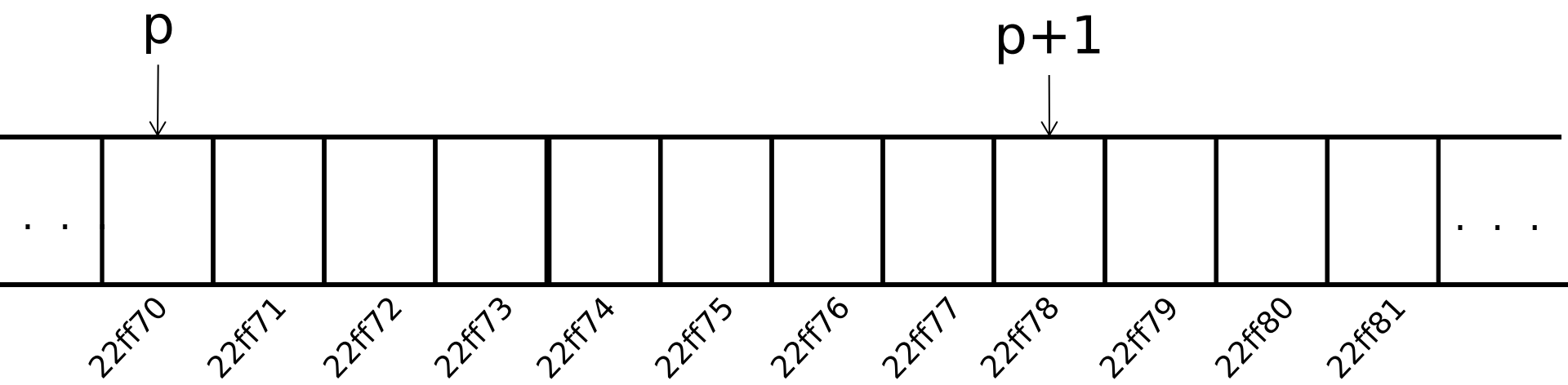
Exemplo de execução:

```
0x28ff44
0x28ff48
```

Obs: o endereço de x pode variar de execução para execução

Operações Com Ponteiros

- No exemplo anterior, se o ponteiro for double, o endereço varia de 8 em 8, pois double ocupa 8 bytes
- double *p; --> p=22ff70--> p+1=22ff78



Endereço de um ponteiro

- Como um ponteiro também é uma variável, ele também tem um endereço de memória
- Exemplo:
  - int *p;
  - cout << &p; //Imprime o endereço de p
  - cout << p; //Imprime o endereço armazenado por p
  - cout << *p; //Imprime o valor apontado por p

```
#include<iostream>
using namespace std;
int main(){
    int x=5, y=6;
    int *px, *py;
    px = &x;   //atribuição
    py = &y;

    if(px<py)
        cout << "py-px= " << (py-px) <<"\n";
    else
        cout << "px-py= " << (px-py) <<"\n";

    cout << "px= " << px;
    cout << " , *px= " << *px;
    cout << ", &px= " << &px << endl;

    cout << "py= " << py;
    cout << " , *py= " << *py;
    cout << ", &py= " << &py << endl;

    py++;

    cout << "\npy++ \npy= " << py;
    cout << " , *py= " << *py;
    cout << ", &py= " << &py << endl;

    px = py + 5;

    cout << "\npx = py + 5\npx= " << px;
    cout << " , *px= " << *px;
    cout << ", &px= " << &px << endl;

    cout << "\npx-py= " << (px-py) <<
     endl;


}
```

Resultado

px-py= 1
px= 0x28ff44 , *px= 5, &px= 0x28ff3c
py= 0x28ff40 , *py= 6, &py= 0x28ff38

py++
py= 0x28ff44 , *py= 5, &py= 0x28ff38

px = py + 5
px= 0x28ff58 , *px= 7281160, &px= 0x28ff3c

px-py= 5

Ponteiro de Ponteiro

- Pode-se criar um ponteiro para um ponteiro
  - Podem ser criados quantos níveis forem necessários(ponteiro para ponteiro para ponteiro ...)
- Na declaração, cada nível é expresso por um *
- Ex:

```
#include<iostream>
using namespace std;
int main(){
    int **pp;
    int *p;
    int x  = 10;
    p = &x;
    pp = &p;
    cout << "x: " << x << "  &x: "<< &x;
    cout << "\np: " << p << "  &p: "<< &p << "  *p: " << *p;
    cout << "\npp: " << pp <<"  *pp: "<< *pp << "  **p:" << **pp;

}
```

Resultado

- x: 10 &x: 0x28ff3c
- p: 0x28ff3c &p: 0x28ff40 *p: 10
- pp: 0x28ff40 *pp: 0x28ff3c **p:10

```
#include <iostream>
using namespace std;
int main () {
    char letra='a';
    char *ptrChar;
    char **ptrPtrChar;
    char ***ptrPtr;
    ptrChar = &letra;
    ptrPtrChar = &ptrChar;
    ptrPtr = &ptrPtrChar;
    cout << "O valor final de ptrPtr e " << ***ptrPtr << endl;
}
```

Ponteiro para Ponteiro

Comando new

- Retorna aloca e retorna o endereço de uma área de memória
- Pode ser utilizado para qualquer tipo de dados

```
#include<iostream>
using namespace std
int main(){
  double x = *(new double);
  x = 10;

  cout<<x;
}
```

```
#include<iostream>
using namespace std;
int main(){
  double *x = new double;
  double y = 5;
  *x = y;

  cout<<*x;
}
```

Ponteiros e Matrizes

- Uma matriz é sempre um ponteiro para seu primeiro elemento

```
#include<iostream>
using namespace std;
int main(){
  double x[] = {1,2,3,4,5};
  cout << "x= " << x<<"\n\n";

  for(int i=0; i<5; i++)
    cout << *(x+i)<<"\n";
}
```

```
#include<iostream>
using namespace std;
int main(){
  cout << "Informe o tamanho do vetor:";
  int t;
  cin >> t;
  int *m = new int[t];
  for(int i=0;i<t;i++){
    cout << "Informe o " << i+1 << "o elemento:";
    cin >> m[i];
  }
  for(int i=0;i<t;i++){
    cout << " " << m[i];
  }
}
```

Ponteiro NULL

- O endereço 000000 é denominado NULL
- Quando um ponteiro possui esse valor, diz-se que ele está apontando para o nada
- NULL é utilizado para informar que o ponteiro está vazio

Exercicios

Dado o programa abaixo, preencha as tabelas

```
main() {
int i, j, *p_1, *p_2, **p_p_1, **p_p_2;
i = 4;
j = 5;
p_1 = &i;
p_2 = &j;
p_p_1 = &p_2;
p_p_2 = &p_1;
}
```

Nome Variável	i	j	p_1	p_2	p_p_1	p_p_2
Conteúdo	4	5	1000	1007	1053	1030
Endereço	1000	1007	1030	1053	1071	1079

Expressão	i	*p_2	&i	&p_2	**p_p_1	*p_p_2	&*p_1	j	*p_1	*&p_2
Resultado	4	5	1000	1053	5	1079	1000	5	4	1007

Passagem por Referência com Ponteiros

- Passa-se para a função o ponteiro para a variável
- Dentro da função trabalha-se com o valor apontado pelo ponteiro (ex: *x)
- Utilizado em C

```
#include <iostream>
using namespace std;
void sem_referencia(int x, int y){
     x = x * x;
     y = y * y;
}
void referencia_tradicional(int &x, int &y){
    x = x * x;
    y = y * y;
}
void referencia_ponteiros(int *x, int *y){
    *x = *x * *x;
    *y = *y * *y;
}
int main(){
    int a=1, b=2, c=3, d=4, e=5, f=6;
    sem_referencia(a,b);
    referencia_tradicional(c,d);
    referencia_ponteiros(&e,&f);
    cout << "Sem referencia:          a=" << a << "  b=" << b;
    cout << "\nReferencia tradicional    c="<< c << "  d=" << d;
    cout << "\nReferencia com ponteiros: e="<< e << " f=" << f
      << endl;


}
```

Resultado:

```
Sem referencia:          a=1  b=2
Referencia tradicional    c=9  d=16
Referencia com ponteiros: e=25 f=36
Pressione qualquer tecla para
continuar. . .
```

Exercícios

- Crie uma função recursiva que calcule o mdc de dois número (a,b) da seguinte forma:

  se a%b ==0 --> mdc(a,b) =b

  senão mdc(a,b) = mdc(b,(a%b))

- Faça um programa que receba do usuário o tamanho de um vetor e o valor de cada um de seus elementos. Imprima o vetor na tela. Exigência: não utilize v[i] para referenciar o vetor, utilize *v.
- Crie um programa que leia um arquivo texto contendo números (float) separados por espaço. O primeiro número informa quantos número contém o arquivo. Utilize esse valor para criar um vetor nesse tamanho. Exiba para o usuário a média, o maior e o menor desses números.

```
//MDC recursivo (número 1)
#include<iostream>
using namespace std;
int mdc(int a, int b){
   if (a%b==0)
      return b;
   else
      return mdc(b,(a%b));
}
int main(){
    cout << mdc(80,60)<< endl;
}
```

```
//Tamanho vetor (número 2)
#include<iostream>
using namespace std;
int main(){
    int t;
    cout << "Tamanho do vetor:";
    cin >> t;
    int *v = new int[t];
    for(int i=0; i<t; i++){
            cout << "\nDigite um número:";
            cin >> *(v+i);
    }
    cout << endl << endl;
    for(int i=0; i<t; i++){
            cout << *(v+i) << "  ";
    }
    cout << endl;
}
```

```
//Arquivo vetor (número 3)
#include<iostream>
#include<fstream>
using namespace std;
int main(){
    ifstream fin("numeros.txt");
    int t;
    fin >> t;
    double *v = new double[t];
    for(int i=0; i<t; i++){
            fin >> v[i];
    }
    double max = v[0];
    double min = v[0];
    double soma = 0;
    for(int i=0; i<t; i++){
            if(max < v[i]) max = v[i];
            if(min > v[i]) min = v[i];
            soma = soma + v[i];
    }
    cout << "\nMaior: " << max << "  Menor:"<< min << "  Media:" << (soma/t)<<
     endl;
}
```