Introdução à Programação

Programação Imperativa
(Visão Geral e Ponteiros)

Tópicos da Aula

- Hoje aprenderemos fundamentos do paradigma imperativo
  - Conceito de Paradigma
  - Os grandes paradigmas
  - Paradigma Imperativo X Orientação a Objetos
- A Linguagem Imperativa C
  - Características e Críticas
  - Ponteiros
  - Funções de entrada e saída
  - Arrays e Ponteiros

Paradigmas de Programação

- Linguagens de programação são baseadas em vários conceitos:
  - Tipos de dados, variáveis e armazenamento, controle, abstração de dados, abstração procedural, sistema de tipos,etc
- A forma como estes conceitos são agrupados definem uma linguagem de programação
- Diferentes seleções dos conceitos suportados pela linguagem definem o estilo de programação ou Paradigma
  - Afeta a forma de pensar em como escrever um programa

Paradigmas de Programação

- Podemos encontrar 6 grandes paradigmas suportados pelas linguagens de programação:
  - Imperativo
    - —Uso de comandos, variáveis e procedimentos
    - —Primeiras linguagens de programação são deste paradigma
    - —Ex: Fortran, Pascal, C, etc
  - Orientação a Objetos
    - —Uso de classes, objetos, herança, polimorfismo, maior abstração de dados, encapsulamento(implementação)
    - —Evolução do paradigma imperativo
    - —Ex: Smalltalk, C++, Eiffel, Java, C#, etc

Paradigmas de Programação

- Funcional
  - Uso de expressões e funções, ausência de variáveis e comandos
  - Ex: ML, Haskell, Mercury
- Lógico
  - Relações lógicas entre entradas e saídas, asserções
  - Ex: Prolog, Mercury
- Concorrente
  - Execução concorrente de processos, abstração de controle de sincronização (inter-processos, acesso a recursos, etc)
  - Ex: Occam, Java, Concurrent C, etc
- Scripting
  - Sistema de tipos dinâmica, abstração de processamento de Strings, suporte a interfaces gráficas
  - Ex: Python, Java Script, Tcl, HTML

Na programação imperativa...

- Não temos classes nem objetos
- Não temos métodos nem atributos
- Ausência de mecanismos para restringir acesso (visibilidade) a variáveis e métodos
- Só temos
  - Funções ou procedimentos (correspondem aos métodos)
  - Variáveis (correspondem às variáveis locais e às variáveis estáticas)

Um programa imperativo...

Contém

- Uma função principal, por onde começa a execução do programa
- Várias funções auxiliares, para modularizar, dividir o código em partes
- Importação de bibliotecas de funções, que correspondem aos pacotes de Java

Mas sem controle de visibilidade e espaço de nomes

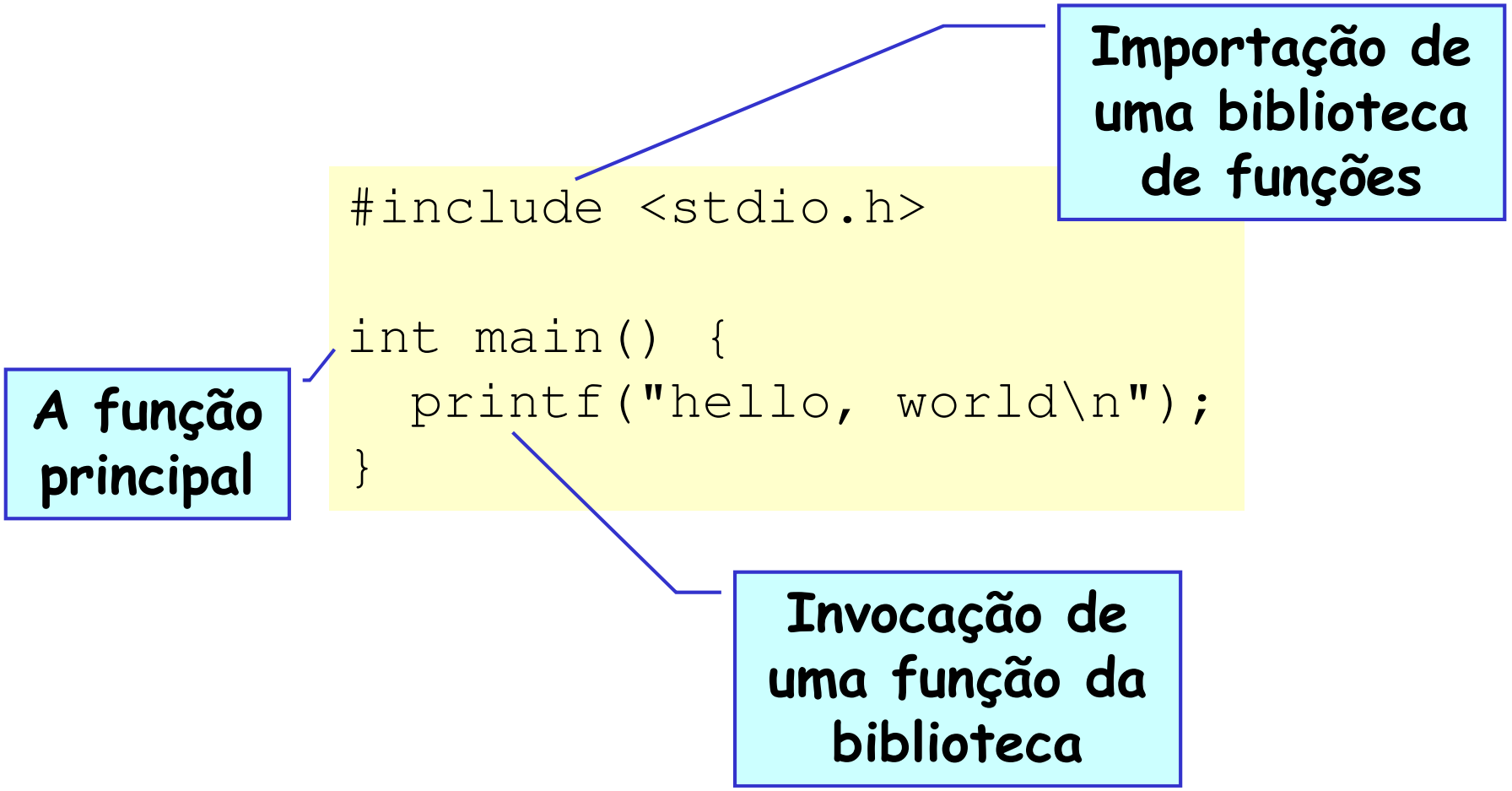
Apresentando a Linguagem C

- Linguagem de programação bastante popular
  - Muito da sintaxe de Java foi copiada de C
- Combina o *alto nível* com o *baixo nível*, permitindo a manipulação direta de bits, bytes e endereços (ponteiros)
- Possui apenas 32 palavras-chaves (reservadas)
- Possibilita alocação dinâmica de memória
- Permite economia de expressão e gera códigos reduzidos
- Permite estruturar o software em módulos, arquivos fontes, bibliotecas

Linguagem C - Críticas

- Dá-se muita liberdade ao programador
- Programas ininteligíveis, acesso direto à memória
- Não há verificação de tipos e nem de limites de *arrays*
  - Simplifica o projeto do compilador
- Tamanho dos tipos pode variar de acordo com o compilador e o computador
- Uso de ponteiros pode se tornar muito confuso
- Mensagens de erro muito vagas (limitação do compilador)
- Ausência de suporte ao tratamento de exceções

Um programa em C



Definindo e chamando funções auxiliares

```
#include <stdio.h>

void printString(char s[]) {
  printf(s);
}

main() {
  printString("hello,");
  printString("world");
  printString("\n");
}
```

Função auxiliar, recebendo um array de **char**, o que corresponde a **String** em C

Termina com '\0'

Nas funções podemos ter...

- Variáveis locais
  - usando tipos correspondentes aos tipos primitivos de Java, com diferenças mínimas
- Atribuições, como em Java
- Estruturas de controle
  - praticamente as mesmas de Java
- Chamadas de funções
- ...

A função para multiplicação

```
int multiply (int a, int b) {
    int count = abs(a);
    int n = abs(b);
    int mult = 0; int i;
    for (i = 0;i < n; ++i)
        mult = mult + count;
    if ((a > 0 &&  b < 0) ||
        (a < 0 && b > 0)) {
      mult = mult * (-1);
    }
    return mult;
 }
```

Variável não pode ser declarada no cabeçalho do for

Função da biblioteca padrão, importada automaticamente, que devolve o valor absoluto

Nas funções **não** podemos ter...

- `new`
- `this`
- `super`
- `throw`
- `instanceof`
- `boolean`
- `...`

Também não temos overloading, polimorfismo, dynamic binding, etc. **(veremos depois)**

Ao invés de referências para objetos...

Nas funções podemos ter acesso a referências para variáveis

Variáveis e endereços

- Memória abstrata:

  {x→5, y→9, z→'a'} (Id→Valor)

- Memória concreta:
  - Associações:

    {x→13, y→72, z→00} (Id→Ref)

  - Memória de fato:

    {00→'a',...,13→5, (Ref→Valor)

    72→9,...,99→*undefined*}

Ponteiros

- Toda variável tem um endereço e uma posição associados na memória
- Este endereço é visto como um ponteiro, uma referência, para o conteúdo da variável, da posição de memória
- Este endereço pode ser armazenado em uma variável do tipo ponteiro

Ponteiros em C

- Variáveis do Tipo Ponteiro

```
int  b ;   /* declara uma variável de nome b que pode
              armazenar valores inteiros (4 bytes)  */

int  *p ;  /* declara uma variável de nome p que pode
              armazenar um endereço de memória para um
              inteiro*/
```

- Operadores unários
  - & → resulta no endereço da posição de memória reservada para a variável. Forma: &p
  - * → resulta no conteúdo do endereço de memória armazenado pela variável ponteiro. Forma: *p

Ponteiros em C

```
int i, j;

int *ip;

i = 12;

ip = &i;

j = *ip;

*ip = 21;
```

A variável **ip** armazena um ponteiro para um inteiro

O endereço de **i** é armazenado em **ip**

O conteúdo da posição apontada por **ip** é armazenado em **j**

O conteúdo da posição apontada por **ip** passa a ser **21**

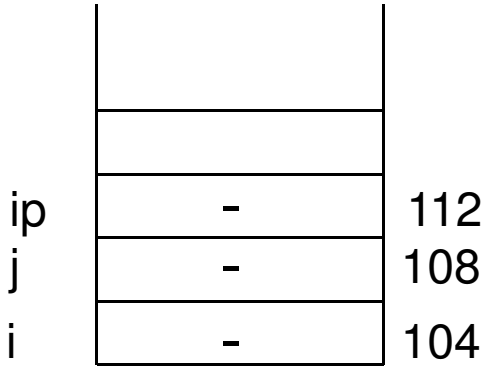
Ponteiros em C

- Variáveis do Tipo Ponteiro

1)

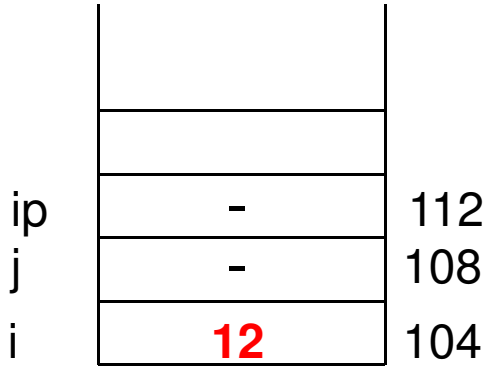
```
int  i,j ;

int  *ip ;
```



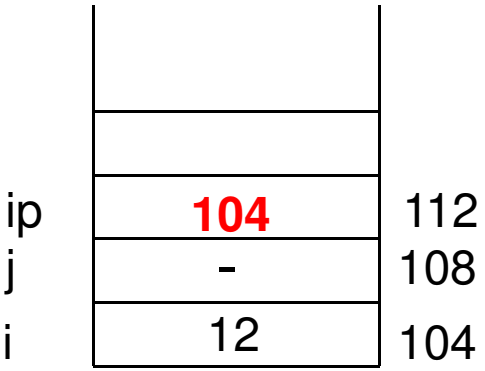
2)

```
i = 12 ;
```



3)

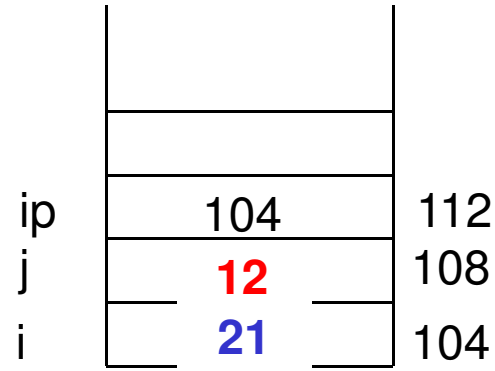
```
ip = &i ;
```



4)

```
j  = *ip;

*ip  = 21;
```



Ponteiros em C

```
int main  () {  /* função  principal */
   int   a , *p ;
   p = &a ;
   *p = 2 ;
   printf (a);
   return  0 ;
}
```

Imprime o valor 2

```
int  main  () {      /* função principal*/
   int a,b,*p ;
   a = 2 ;
   *p = 3 ;
   b = a +( *p );
   printf(b);
   return 0;
}
```

Erro típico de manipulação de ponteiros - ponteiro não inicializado!

A passagem de parâmetros em C é por valor...

```
void swap(int x, int y){
   int temp;
   temp = x;
   x = y;
   y = temp;
}
```

```
int a,b;
a = 8;
b = 12;
swap(a,b);
```

A chamada da função não afeta os valores de **a** e **b**

Mas temos o equivalente à passagem por referência

```
void swap(int *px, int *py){
   int temp;
   temp = *px;
   *px = *py;
   *py = temp;
}
```

```
int a,b;
a = 8;
b = 12;
swap(&a,&b);
```

A chamada da função **afeta** os valores de **a** e **b**

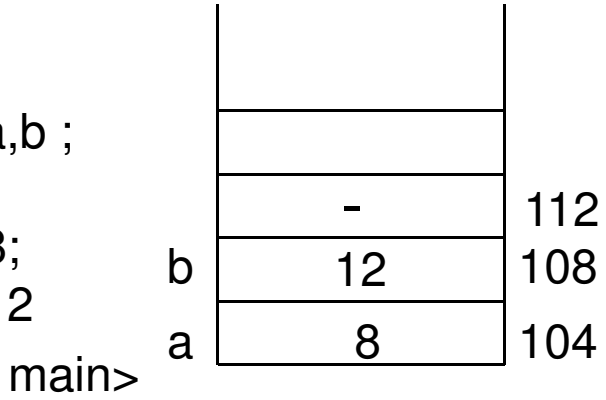
Passagem por Referência em C

- Passo a Passo do programa que chama o swap

1)

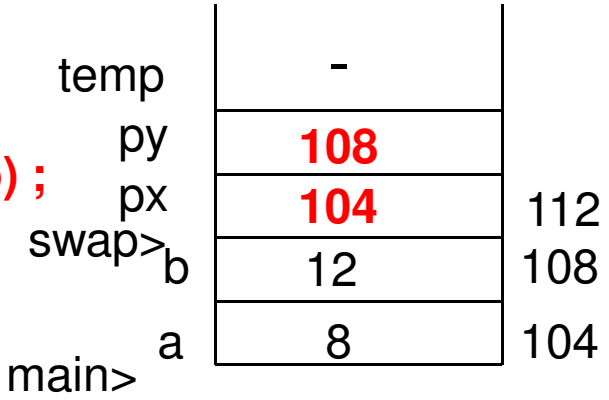
```
int  a,b ;

a = 8;
b = 12
```



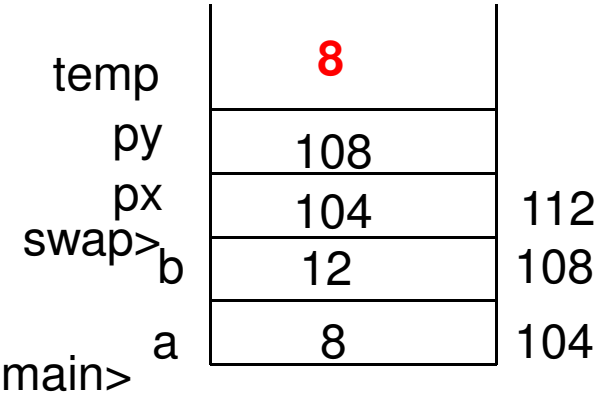
2)

swap(&a,&b) ;



3)

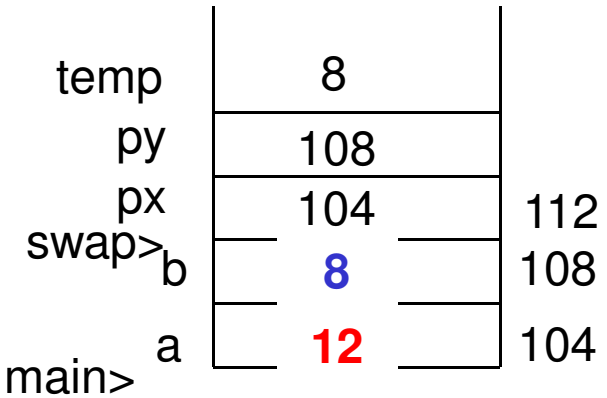
temp = *px;



4)

***px = *py;**

***py = temp;**



Leitura de dados com `scanf`

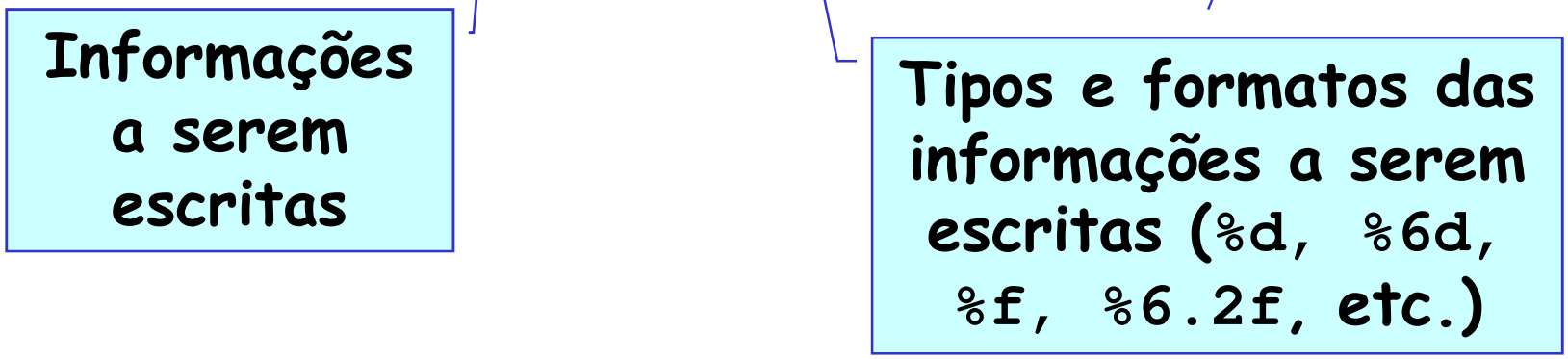
```
int numero;
printf("Digite um decimal:\n")
scanf("%d",&numero);
```

Formato e tipo da informação a ser lida (`c`, `s`, `f`, etc.)

Endereço da variável aonde a informação lida deve ser armazenada depois de convertida

Escrita de dados com `printf`

```
printf("numero: %s, saldo: %.2f",
        n, v);
```



Arrays

```
int numeros[10];
numeros[0] = 5;
char numero[11];
```

```
void printStr(char s[]){
   printf(s);
}
```

O tamanho tem que ser especificado na declaração de uma variável, mas não na declaração de um parâmetro

Ponteiros e arrays

Arrays podem ser tratados como ponteiros em C!

```
int a[10];      →  *pa == a[0] == pa[0]
int *pa;        →  *(pa+i) == a[i] ==
pa = &a[0];        pa[i] == *(a+i)
pa = a;         →  a+i == &a[i]
```

Equivalentes!

~~a = pa~~

~~a++~~

Resumindo

- Fundamentos do paradigma imperativo
  - Conceito de Paradigma
  - Paradigma Imperativo X Orientação a Objetos
- A Linguagem Imperativa C
  - Características e Críticas
  - Ponteiros
  - Funções de entrada e saída
  - Arrays e Ponteiros