Estrutura de Dados

Profa. Juliana Kaizer Vizzotto

Aula 1

Introdução

- Por que estudar *estrutura de dados*?
- Por que o estudo de estruturas de dados é importante para o curso de computação?

Nossa disciplina:

1. Estudo de estrutura de dados e algoritmos para manipulação de tais estruturas.

Introdução

- Por que estudar *estrutura de dados*?
- Por que o estudo de estruturas de dados é importante para o curso de computação?

Nossa disciplina:

1. Estudo de estrutura de dados e algoritmos para manipulação de tais estruturas.

Roteiro

- Revisão
  - Tipos de dados em C
  - Tópicos Avançados em Ponteiros
  - Matrizes n-dimensão dinâmica com uso de Ponteiros
  - Passagem por referência
- TAD (Tipos abstratos de Dados)
- Litas, Pilhas e Filhas
- Árvores
- Grafos

Revisão:

Tipos primitivos da linguagem C

```
char ch;
int i;
float f;
double d;
```

Tipos complexos da linguagem C

```
char *s; char s[11];
int v[10];
struct
int *p;
```

Revisão:

Struct: estrutura heterogênea. Define um grupo arbitrário de objetos relacionados.

```
struct tag {
            declaracao dos membros
};
```

- A definicão de `struct` declara uma variável?
- Define um tipo novo!!!
- Para declarar uma variável do tipo `struct`:

  `struct tag nome_da_variavel`

Revisão:

Exemplo: desejamos guardar informacão sobre a frequência de palavras em algum texto

```
struct contapalavra {
            char palavra[TAMANHO];
            int freq;
};
```

Definindo um vetor de tais estruturas:

```
struct contapalavra contafreq[TAMANHO2];
```

Combinando:

```
struct contapalavra {
            char palavra[TAMANHO];
            int freq;
} contafreq[TAMANHO2];
```

Revisão:

Operacões em Structures

```
contafreq[0].freq ?
contafreq[0].palavra[0]?
scanf(‘‘%s %d’’,contafreq[0].palavra,&contafreq[0].freq)
```

Copiando struturas:

```
contafreq[1] = contafreq[0]
```

Revisão:

Operacões em Structures

```
contafreq[0].freq ?
contafreq[0].palavra[0]?
scanf(‘‘%s %d’’,contafreq[0].palavra,&contafreq[0].freq)
```

Copiando struturas:

```
contafreq[1] = contafreq[0]
```

Revisão: variáveis simples como argumentos de funcões.

Quando uma funcão é chamada, memória para os parâmetros é alocada e os valores dos argumentos são copiados nesse espaco. Então memória é alocada para as variáveis locais a funcão é executada.

```
main()
{
 int x, y;
 x = 0;
 y = 1;
 printf(‘‘Main antes da swap: x = %d, y = %d\n’’,x,y);
 swap(x,y)
 printf(‘‘Main depois da swap: x = %d, y = %d\n’’,x,y);
}
```

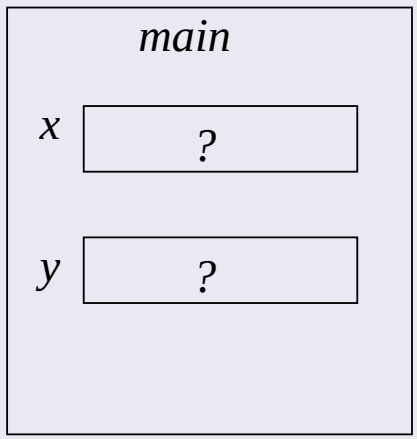
Revisão: variáveis simples como argumentos de funcões.

```
void swap(x,y)
int x, y;
{
 int temp;
 printf(``No inicio da  swap: x = %d, y = %d\n'',x,y);
 temp = x;
 y = temp;
 printf(``No final da swap: x = %d, y = %d\n'',x,y);
}
```

O objetivo deste programa é trocar os valores de x e y no programa principal. ???

Revisão: variáveis simples como argumentos de funcões.

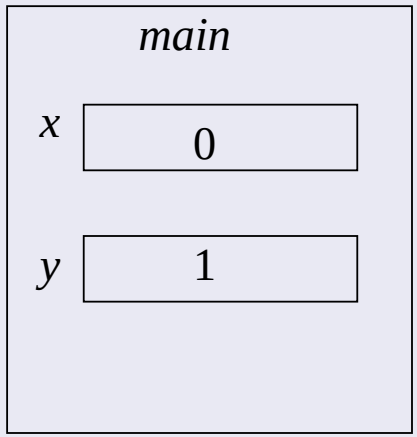
- Main contém duas variáveis locais:
- Quando executamos o programa:



- Usos posteriores de x e y se referem aos conteúdos nos espacos de memória que foram alocados.
- x e y são nomes simbólicos para esses espacos.

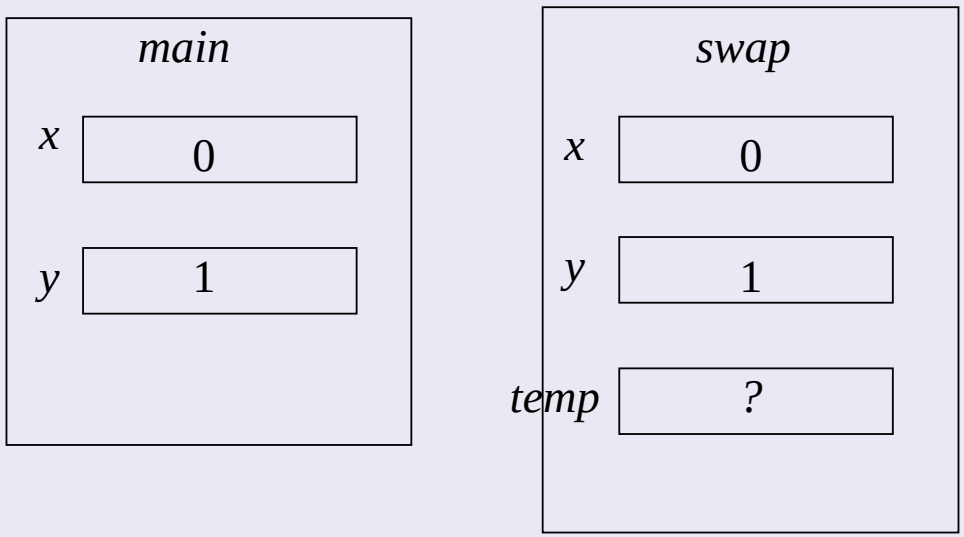
Revisão: variáveis simples como argumentos de funcões.

- x = 0 e y = 1



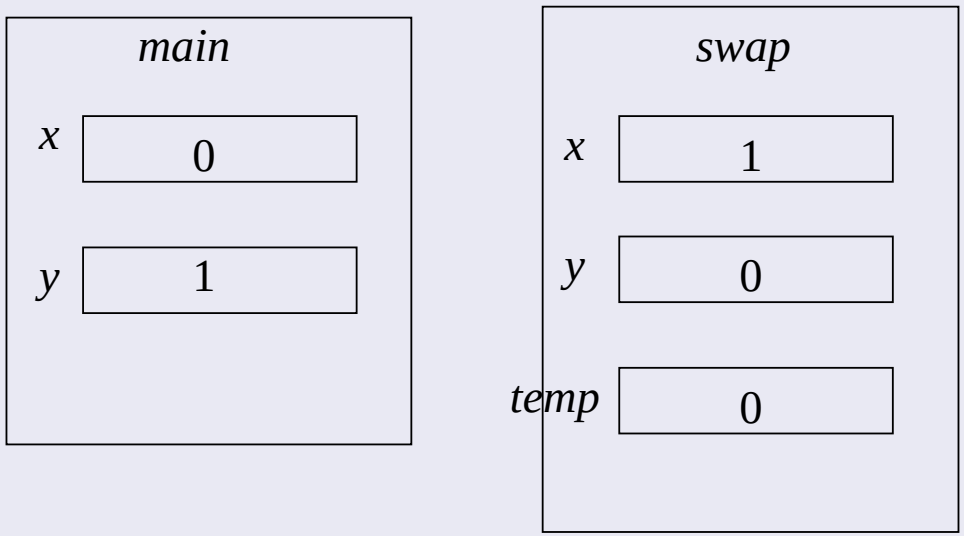
Revisão: variáveis simples como argumentos de funcões.

- Quando swap é chamada, memória para os parâmetros é alocada.



Revisão: variáveis simples como argumentos de funcões.

- Swap troca os valores de suas próprias cópias de x e y.



- Quando swap termina, todas as suas variáveis são detruídas e seu espaco de memória é retornado para o spool.

Revisão: structures como argumentos de funcões.

- Os conteúdos de toda a estrutura são copiados para os parâmetros correspondentes.
- Se os valores dos membros são trocados dentro da funcão, a troca **não** afeta o parâmetro atual.

Revisão: structures como argumentos de funcões.

```
# define TAMNOME 52
struct estudante {
      char nome[TAMNOME]
      int idno;
};
void swap(s, t)
struct estudante s, t;
{
      struct student temp;
      temp = s
      s = t;
      t = temp;
}
```

Funciona???

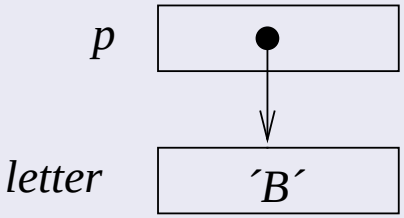
Revisão: vetores como argumentos de funcões.

- Quando vetores são passados como argumentos de funcões, a semântica é um pouco diferente.
- Quando uma funcão troca o valor de um elemento no vetor, o elemento correspondente será trocado no parâmetro atual.

Revisão: ponteiros

- Vimos que funcões não podem trocar o conteúdo de parâmetros atuais declarados como variáveis simples.
- Temos três solucões para este problema:
  1. utilizar todas as variáveis como vetores :-(
  2. usar variáveis globais :-(
  3. usar ponteiros!! :-)

Variável Ponteiro: contém o *endereco* de outra variável!

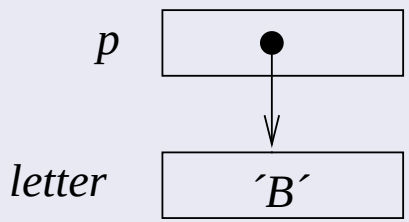



```
/* 1 */ char letter, *p;
/* 2 */ p = &letter;
/* 3 */ letter = 'A';
```

Revisão: ponteiros

- Acessando a localizacão de memória que uma variável ponteiro "aponta": $*nome\_da\_variavel$.

```
/* 4 */ *p = 'B';
```

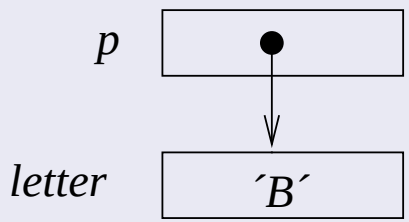


- O operador ($'*'$) pode ser pensando como "siga a seta"!
- Assim, ($'*p'$) pode ser pensando como "siga a seta armazenada em p"!
- ($'*p'$) e *letter* são equivalentes? Por quê?

Revisão: ponteiros

- Acessando a localizacão de memória que uma variável ponteiro "aponta": $*nome\_da\_variavel$.

```
/* 4 */ *p = 'B';
```



- O operador ($'*'$) pode ser pensando como "siga a seta"!
- Assim, ($'*p'$) pode ser pensando como "siga a seta armazenada em p"!
- ($'*p'$) e *letter* são equivalentes? Por quê?

Revisão: ponteiros

```
main()
{
  char letra, *p;
  char car;
  p = &letra;
  letra = 'A';
  printf(''letra = %c, *p = %c\n'', letra,*p);
  *p = 'B';
  printf(''letra = %c, *p = %c\n'', letra,*p);
  p = &car;
  *p = 'Z';
  printf(''letra = %c, *p = %c, car = %c\n'',
                letra,*p, car);
}
```

Qual a saída do programa acima?

Revisão: Typedef

Criando novos nomes para tipos. Por exemplo

```
struct student {
      char name[80];
      int id;
  };

typedef struct student student_data;

typedef struct student {
      char name[80];
      int id;
  } student;
```

Revisão: Alocacão dinâmica de memória

Ponteiros para alocacão dinâmica de memória: programa requer memória em tempo de execucão!

- Chamar uma funcão que retorna o endereco de algum *espaco novo*.
- Este endereco pode então ser armazenado em uma variável ponteiro acessado utilizando operacões normais.
- Em C, isto é feito através da utilizacão da funcão *malloc*().
- *Malloc*() retorna um ponteiro para um caracter, o qual deve ser transformado no tipo apropriado de ponteiro. Ele tem um argumento que a quantidade de espaco a ser alocada.
- Cada tipo de dado requer uma quantidade diferente de espaco.
- Para isto utiliza-se o operando *sizeof* ().

Revisão: Alocacão dinâmica de memória

```
int *p_Array;
int i, count;
printf(‘‘Quantos inteiros?’’);
scanf(‘‘%d’’, &count);
p_Array = (int *) malloc(count * sizeof(int));
if (p_Array == NULL) {
        printf(‘‘Nao foi possivel alocar espaco!’’);
        exit(1);
 }
printf(‘‘Digite %d inteiros.\n’’,count);
for (i =0;i < count; i++){
     scanf(‘‘%d’’, &p_Array[i]);
 }
free(p_Array);
```

Revisão: Vetores e Alocacão dinâmica

Vetores

- Na declaração `int v[10]`, o símbolo `v`, o qual representa o vetor, é uma *constante* que representa seu endereço inicial, isto é, `v` sem indexação, *aponta* para o primeiro elemento do vetor.
- A linguagem C também suporta aritmética de ponteiros: podemos somar e subtrair ponteiros.
- Se `p` representa um ponteiro para um inteiro, então `p+1` representa um ponteiro para o próximo inteiro armazenado na memória.
- Portanto, escrever `&v[i]` é equivalente a escrever `v+i` e escrever `v[i]` é equivalente a escrever `*(v+i)`.

Revisão: Vetores e Alocacão dinâmica

Passagem de Vetores para Funções

- Passar um vetor para uma função consiste em passar o endereço da primeira posição do vetor.
- Se passarmos um vetor de `int`, devemos ter um parâmetro do tipo `int*`, capaz de armazenar o endereço de inteiros.
- **Passar o endereço inicial do vetor**!.
- Os elementos do vetor não são copiados para a função, o argumento copiado é apenas o endereço do primeiro elemento.

Revisão: Vetores e Alocacão dinâmica

Passagem de Vetores para Funções: exemplo

```
float media(int n, float* v){
  int i;
  float s = 0.0;
  for(i = 0; i < n; i++){
   s +=v[i];
  return s/n} }
int main(void){
 float v[10];
 float med;
 int i;
 for (i=0;i<10;i++)
  scanf("%f",&v[i]);
 med=media(10,v); //passa o end do elemento 1
 printf("Media = %f", med);
 }
}
```

Revisão: Vetores e Alocacão dinâmica

Passagem de Vetores para Funções

- Observe que podemos alterar os elementos do vetor dentro da função!
- **Exercício**: Faça uma função que incremente os elementos do vetor. Implemente a main tb.