

Revisão Pontos Importantes:

Structs: É um novo tipo que permite a criação de tipos compostos já definidos em uma só estrutura lógica.
 Pode ser usado como reto e é muito útil no uso de dados
 maiores:

Struct nome {

// tipos já existentes

}

typedef struct {

// tipos já existentes

} nome;

Struct ~~do~~ nome nome var; } nome nome var;

Ou seja a um tipo de nome struct var - re & (=)

Além de poder criar uma struct, pode - se copiar para outro
 com repons (=).

→ União:

Semelhante à struct, mas os membros não podem existir, apenas

um por vez. (Toda os membros compartilham o mesmo espaço de memória)

Síntaxe Similar:

```
typedef union {
```

```
// tipos existentes
```

```
{ nome;
```

```
}
```

```
Nome nomevar;
```

A união divide a memória necessária para o maior membro da união. [MEIO INVILIG]

→ Enum:

defini números contínuos para um único, associado à um nome.

Síntaxe: typedef enum {
 ⁰ ¹
 ⁺ ⁺
 valor1, valor2, ... } nome;

[MAIS INVILIG AINDA B]

→ Funções (Passagem por cópia e referência): [IMPORTANTE]

⇒ Cópia: uma função que reúne valores e faz uma cópia para, uma variável de origem.

⇒ Referência: altera a variável original (Poupa memória e recursos)

Para usar os valores que o operador (&) ;^o retira o endereço de memória de uma variável.

Pra ter variáveis que trabalham com endereços, vira - se pointers:

tipo
variável * nome
variável;
+
operadores de ponteiro.
↳ declaração
* nome
variável.
↳ uso

↳ nome = & valor; →
variável
↳ declaração
[Por padrão vetores e matrizes já são
Pointers que apontam para o 1º elemento]

↳ Structs como parâmetros:

Structs no padrão são normais, mas se precisa usar, a referência é feita do mesmo símbolo, apenas o nome de seus membros muito de operações:

Struct

int x,y;

{ Ponto

Váriavel anônima (Ponto *P) {

P->x++;

P->y++;

Main { Ponto p1 = {1,2}, anônimo (*P) } .

P-D X ++ equivalente à: (*P).X ++ ; quando o operador

→ Revisão:

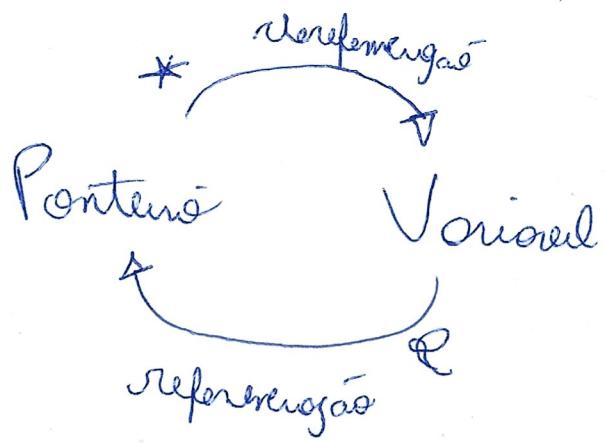
→ revisão
A revisão (ou for melhorar) é uma função que se chama diversas vezes para resolver um problema de forma controlada.

Ela irá realizar uma tarefa e se chamar até que chegue ao caso base, processado chamado Stak, quando for desimpedido ele irá desobedecer as regras e será o de Stak e retorna a 1º chamada com a tarefa realizada.

→ Ponteiros:

" " " " "

Este tipo de ponteiro segue seu tipo de revisão;
Um ponteiro que aponta para nada é anulizado com NULL;



Um ponteiro é:
Para receber dados
(entradas) de uma
Variável do mesmo tipo del

2º Aritmética de ponteiros:

Sabe ponteiro pode somar e subtrair, mas deve-se re-
observar os tipos de dados trabalhados; opinião é que importa no
ponto da segurança. (Sabe-se possível comparar 2 ponteiros).

3º Ponteiro genérico:

É aquele que aponta para qualquer tipo, mas deve-se ter em conta o
costo de converter para o tipo trabalhado.

Void * ptr genérico; (void / byte a seu critério!)

4º Vetor:

Já se sabe que ponteiro faz o 1º endereço da memória do
vetor. (Forma referencial)

5º Ponteiro → Ponteiro:

Em C pode-se fazer uma ou mais variações de ponteiros que compõem
ponto de ligação e de tipos diferentes;

PTR & *PTR & **PTR ...

GREAT TIPS:

- 1º) Costing é necessário de float para int;
- 2º) vetores do tipo: char $\times [n][m]$ não permitem
opens: ~~char~~ $\times [] [m]$ nos funçãois.
- 3º) vetores do tipo: char $\times [n]$ não permitem opens:
 $\star []$ nos funçãois. (não só restrito ao char!)
- 4º) Sempre que possível use-se os recursos de return, break e etc... Els eliminam trashes gratis, além disso une coisas
doses que se formam através for do loop, se não mantenha como
está o esce do programa;
- 5º) funções retornam variáveis que se formam por referência no
seu contexto via ponteiros;
- 6º) funções podem retornar structs e etc...
- 7º) For é melhor amigo do programador!!!
- 8º) Setbuff é necessário em reuniões com, use ele sempre.

ALUNO: Gabin F.F. de Souza - R.A: 2669480 - Prof: Muriel Góes