#### *Structs na linguagem C*

**Introdução**

Um ***registro***, é uma estrutura de dados que permite agrupar variáveis possivelmente de diferentes tipos de dados formando uma ***estrutura heterogênea de dados***. Registro é adequado para descrever ***entidades*** do mundo real pois permite agrupar numa única estrutura diversas variáveis que descrevem as características ou atributos relevantes daquela entidade.

Para ilustrar, imagine uma empresa que deseja registrar dados de seus empregados conforme a figura abaixo:



Observa-se a necessidade de registrar várias informações sobre cada empregado, como seu código, nome, categoria funcional, etc.. A princípio, pensa-se em definir uma variável para cada informação. Entretanto seria bastante confuso e complicado tratar cada informação da “*entidade*” empregado de forma isolada. Daí, a ideia de ***registro***, que agrega em uma estrutura unificada todas as informações relevantes sobre a entidade abstraída, no exemplo, a entidade *empregado*.

Cada informação do registro (ou cada quadro na figura acima) é denominado ***campo*** e pode ser acessado ou manipulado individualmente dentro do registro.

**Registros (*structs*) na Linguagem C**

Em C, ***registros*** são definidos por meio da palavra reservada ***struct***. A sintaxe para declarar em C uma *variável do tipo registro* é:

**struct** {

tipo1 campo1;

tipo2 campo2;

...

tipon campon;

} nome\_da\_variável;

Por exemplo, para declarar uma variável que representa uma data (dia, mês e ano) na linguagem C, pode-se utilizar o seguinte código:

struct {

int dia;

int mes;

int ano;

} dt\_aniversario;

Pela codificação, ***dt\_aniversario*** é uma variável do tipo registro, com 3 campos, *dia*, *mes* e *ano*, todos do tipo ***int***. Porém, por ser uma estrutura complexa e às vezes de longa digitação e considerando que um programa pode ter diversas variáveis do mesmo tipo de registro, a boa prática estabelece que não devem ser declaradas ***variáveis registro*** e sim ***tipos de dados registro***. A *sintaxe* de declaração de um ***tipo de dado registro*** em C é:

***struct*** nome\_do\_tipo\_do\_registro {

tipo1 campo1;

tipo2 campo2;

…

tipon campon;

};

Deste modo, pode ser declarado o ***tipo de dado*** data que generaliza toda data, como:

struct data {

int dia;

int mes;

int ano;

};

E então, quando necessário no programa, declarar variáveis do ***tipo*** data, por exemplo, as variáveis *data1* e *dt\_aniversario* poderiam ser declaradas:

**struct data** data1;

**struct data** dt\_aniversario;

Deste modo, o registro da entidade “empregado”, apresentado na figura acima, pode ser declarado, por exemplo:

**struct** empregado {

long codigo;

char nome[51];

char cat\_func[21];

double salario;

**struct** data dt\_inic\_proj;

int temp\_aloc\_proj;

};

Observa-se que o campo *dt\_inic\_proj* da ***struct empregado*** é, por sua vez, um registro do tipo já definido ***struct data***. Observa-se ainda que cada campo tem seu próprio tipo de dado, que pode ser diferente dos demais. Esta é a razão pela qual registros são denominados ***estruturas heterogêneas***.

**Manipulando registros na linguagem C**

A partir da declaração do registro ***struct empregado*** feita acima, pode-se declarar variáveis deste tipo. Por exemplo:

**struct** empregado emp1, emp2;

Então, ***emp1*** e ***emp2*** são variáveis do tipo ***struct empregado***, isto é, ***emp1 e emp2*** são *registros*! Mas como manipular *variáveis registros*? Na realidade, o que são manipulados são os ***campos*** da *variável registro*. A sintaxe em C para acessar um campo do registro é a seguinte:

nome\_da\_variavel.nome\_do\_campo

Por exemplo, os campos da variável ***emp1*** podem ser iniciados como no exemplo a seguir:

emp1.codigo = 12436;

strcpy(emp1.nome, "José da Silva");

strcpy(emp1.cat\_func, "Engenheiro");

emp1.salario = 12500,00;

emp1.dt\_inic\_proj.dia = 01;

emp1.dt\_inic\_proj.mes = 07;

emp1.dt\_inic\_proj.ano = 2018;

emp1.temp\_aloc\_proj = 180;

Note que o campo ***dt\_inic\_proj***, por ser estruturado, apresenta um aninhamento de pontos para referenciar os campos do *registro* ***struct data***.

**A declaração *typedef***

A declaração ***typedef*** da linguagem C cria sinônimos ou *alias* para tipos de dados, por exemplo:

typedef int Inteiro;

No exemplo acima foi criado o sinônimo ***Inteiro*** para o *tipo primitivo* ***int***. Neste caso, pode-se criar variáveis inteiras usando o sinônimo:

Inteiro i, num;

Esta declaração é muito utilizada para renomear tipos ***struct***, simplificando a declaração de variáveis e dando mais semântica para os nomes. Por exemplo:

**typedef struct** empregado {

long codigo;

char nome[51];

char cat\_func[21];

double salario;

**struct** data dt\_inic\_proj;

int temp\_aloc\_proj;

} Empregado;

No exemplo acima, foi criado o sinônimo ***Empregado*** para o tipo ***struct empregado***. Então fica mais simples e com melhor sematica declarar:

**Empregado** emp1, emp2, emprNovato;

do que: **struct** empregado emp1, emp2, emprNovato;

**Nota**: embora não seja uma regra de *sintaxe*, portanto não obrigatório, é padrão de *boas práticas* declarar o *tipo* com letras minúsculas (no exemplo acima, ***struct empregado***) e o *alias* com a primeira letra maiúscula (no exemplo, ***Empregado***). Nesta linha, variáveis são declaradas com letras minúsculas (no exemplo, ***emp1***, ***emp2***), e quando com nomes compostos, o segundo nome com a primeira letra maiúscula (como no exemplo, ***emprNovato***). Sugere-se fortemente adotar sempre esta *boa prática*.

**Manipulação avançada de *structs***

**Passagem de *struct* como parâmetro por *Valor***

Basta declarar a *struct* como parâmetro da *função* ou procedimento e a passagem é como a de uma variável primitiva: é criada uma *variável struct* para o *parâmetro* e o conteúdo do *argumento* (de todos os campos) é cópiado para o parâmetro. Exemplo:

void imprimeEmpregado (Empregado e){

printf(Nome: %s, Salário: R$ %.2f.", e.nome, e.salario);

}

Deste modo, considerando que a variável ***emp1*** já tenha sido iniciada anteriormente, pode-se proceder a chamada abaixo, com os dados do argumento ***emp1*** sendo copiados para o parâmetro ***e***:

...

impreEmpregado(emp1);

...

Também uma ***struct*** pode ser retornada normalmente de uma função:

**Empregado** iniciaEmpregado( ) {

Empregado e;

scanf("%d", &e.codigo);

gets(e.nome");

gets(e.cat\_func);

scanf("%f", &e.salario);

scanf("%d", &e.inic\_proj.dia);

scanf("%d", &e.inic\_proj.mes);

scanf("%d", &e.inic\_proj.ano);

scanf("%d", &e.temp\_aloc\_proj);

**return** e;

}

Assim, a variável ***emp2*** pode ser iniciada com a instrução abaixo:

emp2 = iniciaEmpregado();

**Passagem de *struct* como parâmetro por *Referência***

Neste caso, passa-se um ponteiro para a ***struct***, como no exemplo a seguir:

**void** setSalario(Empregado \*e, double sal) {

e->salario = sal;

}

Então, o salário de ***emp2*** pode ser atualizado com a seguinte chamada:

setSalario(&emp2, 7000.00);

**Nota**: há duas ***formas sintáticas*** para acessar o campo de uma ***struct*** a partir de um *ponteiro*, como ilustrado no exemplo a seguir:

Empregado \*pEmp; // *pEmp* é um ponteiro para tipo *Empregado*

pEmp = &emp2; // *pEmp* “aponta” para variável *emp2*

(\*pEmp).salario = 8000.00; // **forma 1** de acesso a um campo

pEmp->salario = 9000.00; // **forma 2** de acesso a um campo

Observa-se que na **forma 1**, usando o formato de referência ao conteúdo ***(\*pEmp***), então o nome do campo deve ser precedido de ponto: (\*pEmp).salario, (\*pEmp).nome, etc.. Na **forma 2**, usando o formato de referência ao endereço do ponteiro (eEmp), então o nome do campo é precedido do símbolo “aponta” ( -> ): pEmp->salario, pEmp->nome, etc..