Tipos primitivos em C:

|  |  |
| --- | --- |
| **Tipo** | **Tamanho em bytes** |
| char | 1 |
| short | 2 |
| int | 4 |
| long | 4 |
| long long | 8 |
| float | 4 |
| double | 8 |

\*Tecnicamente até existe o bool no C, mas apenas ao importar bool.h

Tipos primitivos em C++:

|  |  |
| --- | --- |
| **Tipo** | **Tamanho em bytes** |
| bool | 1(bit) |
| char | 1 |
| short | 2 |
| int | 4 |
| long | 4 |
| long long | 8 |
| float | 4 |
| double | 8 |
| wchar\_t | 2 a 4 |
| char16\_t | 2 |
| char32\_t | 4 |
| long double | 8, 12 ou 16 |

Tipos primitivos em Java:

|  |  |
| --- | --- |
| **Tipo** | **Tamanho em bytes** |
| boolean | 1(bit) |
| byte | 1 |
| char | 1 |
| short | 2 |
| int | 4 |
| long | 4 |
| float | 4 |
| double | 8 |

2.

Linguagens fortemente tipadas são aquelas nas quais os tipos de cada variável precisam, obrigatoriamente, ser claramente definidos ao declarar a variável. Em não fortemente tipadas, os tipos também existem, porém nelas, não é obrigatório definir o tipo da variável ao declará-la.

Exemplos fortemente tipadas: C, C++, Java, Typescript, C#

Exemplos fracamente tipadas: Python, PHP, Javascript

3.

O operador “.” Tem um nível de precedência maior do que o operador “\*”, logo, quando eu uso \*p.dia, é o mesmo que usar \*(p.dia), o que neste caso seria desreferenciar o valor obtido, isso é, eu acessaria o valor armazenado no endereço que “dia” aponta. Logo, o atributo “dia” teria que ser do tipo ponteiro para isso funcionar.

Por outro lado com (\*p).dia equivale a p->dia, isso é, primeiro desreferenciamos p, ou seja, acessamos a estrutura que ele aponta e depois estou acessando o valor dia da variável p, a qual é a instancia de um struct que contém dia como atributo.

4.

Uma variável estática é uma variável cujo tempo de vida se estende durante toda a execução do programa. Isso acontece pois, variáveis estáticas mantêm o seu valor na célula onde estão alocadas, mesmo após a função onde foram declaradas terminarem de executar. Outra característica desse tipo de variável é que o espaço delas na memória é alocado em tempo de compilação.

Essas características trazem uma série de vantagens e desvantagens. Dentre as vantagens, elas tendem a ser mais perfomaticas em termos de tempo, pois, por serem alocadas durante a compilação e nunca serem eliminadas, o programa não gasta tempo alocando e desalocando o espaço na memória. Sem falar que o acesso a essas variáveis é direto, sem necessitar de cálculos de endereços.

Por outro lado, essas variáveis podem não ser muito úteis em funções recursivas, pois independente da chamada recursiva, o valor da variável se mantem o mesmo. Sem falar que elas podem ser um desperdício de memória pois mesmo se a função com essa variável nunca for usada na execução, vai ser alocado espaço na memória de qualquer forma. Dois subprogramas também não podem compartilhar da mesma variável/estrutura de dados caso essa seja estática. As variáveis estáticas tendem a ser menos flexíveis do que as variáveis típicas usadas.

5.

A black background with white text

AI-generated content may be incorrect.

O código acima está incorreto pois a variável \*temp, a qual é uma variável do tipo ponteiro para inteiro, logo ele só deve armazenar endereços para variáveis. Porém, na linha 3, o código não está apontando para nenhum endereço válido, resultando em um segmentation fault. Quando você iguala com \*x você está recebendo o valor que o endereço dentro de x aponta, que neste caso é do tipo sim, não o endereço, por isso o \*temp não aceita, pois ele espera um endereço de int, e não um int em si. O correto seria temp ser uma variável int normal, sem ser um ponteiro.

6.

A black background with text and symbols

AI-generated content may be incorrect.

No primeiro printf, estou imprimindo apenas o 1234, pois é o valor apontado pelo endereço armazenado em p. Porém, após isso, o operador ++ incrementa o endereço de p, dessa forma, mudando o local da memória para o qual o endereço de p aponta. Por isso que se eu tentar imprimir \*p++ de novo ou \*p eu terei um valor diferente.

Já o segundo imprime o endereço atualizado armazenado na variável ponteiro de int p. Para mim, o resultado foi: 6422300

No terceiro, primeiro o endereço de p é incrementado, e como ele é um inteiro, ele è incrementado em 4. Perceba que o endereço original já foi incrementado 2 vezes, logo, moveu 8 bytes. E depois é impresso o garbage data ao qual esse novo endereço aponta.

O último vai primeio desreferenciar o valor de p, isso é vai pegar o dado apontado pelo endereço em p e depois vai incrementar o valor apontado pelo endereço, e não o endereço em si.

7-

#include <stdio.h>

#include <stdlib.h>

#define HORASemMIN 60

typedef struct{

    int horas;

    int minutos;

}tempo;

int main (void) {

    int n; int original;

    printf("Quanto tempo (em minutos): ");

    scanf("%d", &n);

    original = n;

    //Inicializando

    tempo resultado;

    resultado.horas = 0;

    resultado.minutos = 0;

    //Metodo guloso

    while(n>=HORASemMIN){

        resultado.horas += 1;

        n -= HORASemMIN;

    }

    resultado.minutos = n;

    printf("%d minutos equivalem a %d horas e %d minutos", original, resultado.horas, resultado.minutos);

    return 0;

    }

8-

#include <stdio.h>

#include <stdlib.h>

#define DIAS\_ANO 365

#define DIAS\_MES 30

typedef struct{

    int dia;

    int mes;

    int ano;

}data;

int main (void) {

    int idade, original;

    printf("Sua idade (em dias): ");

    scanf("%d", &idade);

    data idadeCompleta;

    idadeCompleta.dia = 0;

    idadeCompleta.mes = 0;

    idadeCompleta.ano = 0;

    while(idade>=DIAS\_ANO){

        idadeCompleta.ano += 1;

        idade -= DIAS\_ANO;

    }

    while(idade>=DIAS\_MES){

        idadeCompleta.mes += 1;

        idade -= DIAS\_MES;

    }

    idadeCompleta.dia = idade;

    printf("Voce tem %d dias, %d meses e %d anos de idade.", idadeCompleta.dia, idadeCompleta.mes, idadeCompleta.ano);

    return 0;

    }

9-

// Caio Vinicius Corsini Filho

#include <stdio.h>

#include <stdlib.h>

#define HORASemMIN 60

typedef struct{

    int horas;

    int minutos;

}tempo;

tempo somaTempos(tempo tempoCompleto, int minutos){

    tempoCompleto.minutos += minutos;

    while(tempoCompleto.minutos >= HORASemMIN){

        tempoCompleto.horas += 1;

        tempoCompleto.minutos -= HORASemMIN;

    }

    return tempoCompleto;

}

int main (void) {

    tempo tempoCompleto;

    int minutos, n;

    printf("Quantos minutos: ");

    scanf("%d", &minutos);

    printf("Quantos minutos (tempo completo): ");

    scanf("%d", &n); tempoCompleto.minutos = n;

    printf("Quantos horas (tempo completo): ");

    scanf("%d", &n); tempoCompleto.horas = n;

    tempo resultado = somaTempos(tempoCompleto, minutos);

    printf("%dmin + %dh%dmin = %dh%dmin", minutos, tempoCompleto.horas, tempoCompleto.minutos, resultado.horas, resultado.minutos);

    return 0;

    }