Universidade Presbiteriana Mackenzie  
Ciência da Computação – 05P11  
Paradigmas de Linguagens de Programação  
20/02/2025

Alan Meniuk Gleizer  
RA 10416804

Atividade – Paradigma Imperativas

Obs. Todas as atividades cuja entrega principal é em código tem o código neste documento, em texto, e o código fonte .c no mesmo arquivo .zip.

**01. Para linguagem C, C++ e Java, liste todos os tipos básicos suportados e seus tamanhos em bytes.**

Linguagem C  
É importante notar que os tamanhos dos tipos em C dependem da implementação e arquitetura. Os tamanhos listados abaixo consideram o compilador GCC de 32 bits. Existem tipos de largura fixa, definidos na biblioteca stdint.h, como int8\_t, int16\_t, int32\_t e int64\_t.

char: 1 byte

short int: 2 bites

int: 4 bytes

long int: 4 bytes (igual a int em arq de 32 bits!)

float: 4 bytes

double: 8 bytes

Linguagem C++  
Em geral são os mesmos tipos primitivos de C, com a inclusão de bool de 1 byte.

Java  
Diferente de C e C++, os tipos primitivos tem tamanho fixo.

byte: 1 byte

short: 2 bytes

int: 4 bytes

long: 8 bytes

float: 4 bytes

double: 8 bytes

char: 2 bytes

boolean: em geral 1 byte

**02. Explique o seria uma linguagem fortemente tipada, de exemplos de linguagens fortemente tipadas e não fortemente tipadas.**

Uma linguagem fortemente tipada não permite operações entre tipos diferentes / incompatível sem conversão ou casting explícitos. Isso garante maior segurança e previsibilidade na execução do código, mas requer mais atenção aos tipos durante a codificação. Uma linguagem fracamente tipada, por outro lado, realiza conversão de tipos implicitamente. Isso pode simplificar o código e desonerar o programador, mas também pode levar a erros inesperados e de difícil debug. C, C++, Java e Python são fortemente tipadas. Python tem tipagem dinâmica, mas é considerada fortemente tipada. Já JavaScript, PHP e Lua são fracamente tipados.

**03. Explique porque a expressão \*p.dia tem significado muito diferente de (\*p).dia.**

As expressões têm significados diferentes devido à ordem de precedência das operações em C. O “.”, que representa acesso a um elemento do struct, tem precedência maior do que o operador unário \*. Assim, \*p.dia está na verdade fazendo \*(p.dia), ou seja, primeiro acessando a variável dia dentro de p, e na sequência dereferenciando. Isso só é válido se p for um struct e dia, um ponteiro. (\*p).dia, por outro lado, primeiro dereferencía p, que deve ser um ponteiro para um struct, e depois acessa dia. Em geral, (\*p).dia é a opção mais usada, e é equivalente a p->dia.

**04. Leia a seção 5.4.3 do livro de Sebesta e responda qual é vantagem e desvantagens de se utilizar variáveis com tempo de vida estático.**

De acordo com Sebesta, “Variáveis estáticas são vinculadas a células de memória antes do início da execução de um programa e permanecem vinculadas a essas mesmas células até que a execução do programa termine.”. Assim, a grande vantagem é que elas são alocadas na memória uma única vez e mantém seu valor entre diferentes chamadas de função. Isso é mais eficiente e pode ser útil em alguns contextos. A maior desvantagem é que variáveis estáticas são menos flexíveis (por exemplo, em recursão), e também ocupam memória mesmo depois que seu uso não é mais necessário.

**05. Por que o código abaixo está errado?**

**void troca (int \*x, int \*y){**

**int \*temp;**

**\*temp = \*x; \*x = \*y;**

**\*y = \*temp;**

**}**

O código está errado pois temp foi declarado como um ponteiro, ou seja, para armazenar um endereço, mas na terceira linha, ao dereferenciar x (\*x) e atribuir esse valor a temp, estamos atribuindo um valor inteiro a temp.

Vale notar que, ao verificar se essa resposta estava certa no chat GPT, um novo erro foi detectado:  
 *O problema não está somente na atribuição \*temp = \*x, mas sim no fato de que temp é um ponteiro não inicializado. Ou seja, temp foi declarado, mas não recebeu um endereço válido antes de ser dereferenciado (\*temp = \*x). Isso causa comportamento indefinido, pois temp pode apontar para um local inválido na memória.*

**06. Qual é a saída do programa abaixo tente explicar o acontece com os ponteiros do programa.**

**int main (void) {**

**int i; int \*p;**

**i = 1234; p = &i;**

**printf("\*p++ = %d\n", \*p++);**

**printf("p = %ld\n", (long int) p);**

**printf("\*(p++) = %ld\n", (long int) \*(p++));**

**printf("(\*p)++ = %ld\n", (long int) (\*p)++);**

**return 0;**

**}**

Saída:

\*p++ = 1234

p = 140736946917016

\*(p++) = 1626043648

(\*p)++ = -1417542245

\*\*\* stack smashing detected \*\*\*: terminated

O grande problema está no primeiro printf: \*p++ primeiro acessa o conteúdo do endereço apontado por p, e imprime 1234 corretamente. Na sequência p é incrementado, mas por se tratar de um ponteiro, estamos na verdade incrementando um endereço. Não temos como saber o que existe no novo endereço que está sendo acessado, nem mesmo se é um endereço válido dentro do espaço de endereçamento. Todos os prints subsequentes são, dessa forma, comportamento indefinido:

printf("p = %ld\n", (long int) p); -> imprime o novo endereço (incrementado)

printf("\*(p++) = %ld\n", (long int) \*(p++)); -> primeiro dereferencia p, imprime o valor, e incrementa o ENDEREÇO armazenado em p

printf("(\*p)++ = %ld\n", (long int) (\*p)++); -> primeiro dereferencia p, imprime o valor, e incrementa o VALOR apontado por p, mas p continua no mesmo endereço

**07. Escreva um programa que leia um inteiro representando um tempo medido em minutos e calcula o número equivalente de horas e minutos armazenando o cálculo em um tipo registro (por exemplo, 131 minutos equivalem a 2 horas e 11 minutos). Para esse exercício use o tipo tempo definido abaixo para armazenar a resposta do programa, ao final o seu programa imprime o resultado armazenado na variável do tipo tempo.**

**typedef struct{**

**int horas;**

**int minutos;**

**}tempo;**

#include <stdio.h>

typedef struct{

int horas;

int minutos;

}tempo;

int lerInt()

{

// funcao da biblioteca propria para ler um int sem os problemas de scanf

char buffer[32]; // buffer para input

int num;

if (fgets(buffer, sizeof(buffer), stdin) == NULL)

{

return 0; // se fgets é NULL, houve erro de leitura ou EOF

}

if (sscanf(buffer, "%d", &num) != 1)

{

return 0; // verificar se entrada realmente é int

}

return num;

}

int main() {

int entrada = 0;

do

{

printf("Informe a entrada em minutos: ");

entrada = lerInt();

} while (entrada <= 0);

tempo saida;

saida.horas = entrada / 60;

saida.minutos = entrada % 60;

printf("%d horas e %d minutos", saida.horas, saida.minutos);

return 0;

}

**08. Escreva um programa que leia um valor inteiro correspondente à idade de uma pessoa em dias e informe-a em anos, meses e dias armazenado em variável registro do tipo data definida abaixo:**

**typdef struct{**

**int dia;**

**int mes;**

**int ano;**

**}data;  
Obs.: apenas para facilitar o cálculo, considere todo ano com 365 dias e todo mês com 30 dias. Nos casos de teste nunca haverá uma situação que permite 12 meses e alguns dias, como 360, 363 ou 364. Este é apenas um exercício com objetivo de testar seu entendimento do tipo registro.**

#include <stdio.h>

typedef struct {

int dia;

int mes;

int ano;

} data;

int lerInt() {

// funcao da biblioteca propria para ler um int sem os problemas de scanf

char buffer[32]; // buffer para input

int num;

if (fgets(buffer, sizeof(buffer), stdin) == NULL) {

return 0; // se fgets é NULL, houve erro de leitura ou EOF

}

if (sscanf(buffer, "%d", &num) != 1) {

return 0; // verificar se entrada realmente é int

}

return num;

}

int main() {

int entrada = 0;

do {

printf("Informe a entrada em dias: ");

entrada = lerInt();

} while (entrada <= 0);

data saida;

saida.ano = entrada / 365;

entrada = entrada % 365;

saida.mes = entrada / 30;

saida.dia = entrada % 30;

printf("%d anos, %d meses e %d dias", saida.ano, saida.mes, saida.dia);

return 0;

}

**09. Escreva uma função que recebe dois parâmetros, um número inteiro representando um tempo em minutos e um tempo armazenado na estrutura typedef struct tempo representando um tempo em horas:minutos. A função calcula a soma tempo em minuto (parâmetro inteiro) com o tempo armazenado na estrutura, ao final o tempo atualizado é devolvido pela função. Por exemplo se for informado 71 minutos e 3hs:15min a função deve devolver o tempo na estrutura de 4hs:16min.**

#include <stdio.h>

typedef struct {

int horas;

int minutos;

} tempo;

int main() {

int minutos;

tempo tempo;

// ler minutos

printf("Informe um tempo em minutos: ");

scanf("%d", &minutos);

// ler tempo

printf("Informe um tempo no formato XXhs:YYmin: ");

scanf("%dhs:%dmin", &tempo.horas, &tempo.minutos);

// converter minutos

int horas = minutos / 60;

minutos = minutos % 60;

// fazer soma

tempo.horas += horas;

tempo.minutos += minutos;

// retornar valores

printf("Soma = %dhs:%dmin", tempo.horas, tempo.minutos);

return 0;

}

**10. Calculadora de Fração (parte 1)**

**Em matemática, uma fração é um modo de expressar uma quantidade a partir de uma razão de dois números inteiros. De modo simples, pode-se dizer que uma fração, representada genericamente como a/b, que representa o inteiro a dividido em b partes iguais. Neste caso, a corresponde ao numerador, enquanto b corresponde ao denominador, que não pode ser igual a zero.**

**Operações entre frações:**

**Soma de fração: (a/b)+(c/d)=( (a.d+c.b) / b.d ) => (1/2)+(3/4)=5/4**

**(1\*4+3\*2) / (2\*4) => 10/8**

**Multiplicação de fração: (a/b) \* (c/d)= ( (a\*c) / (b\*d) )**

**Divisão de fração: (a/b) / (c/d) = (a/b)\*(d/c) = (a\*d) / (b\*c)**

**Igualdade: (a/b)== (c/d) se a\*d == b\*c**

**Implemente uma calculadora de fração utilizando a linguagem C no paradigma imperativo, use os conceitos vistos nas aulas, ou seja, para representar uma fração você pode definir um novo tipo fração com typedef struct.**

**As operações definidas para fração devem ser representadas como funções que recebem como parâmetro duas estruturas do tipo fração e retorna uma nova fração, isso para as operações de soma, multiplicação e divisão. A operação de igualdade pode retornar verdadeiro (=1) ou falso (=0) na linguagem C.**

**Para testar as implementações das funções escreva uma função principal (main()) que faz as chamadas das funções que implementam as operações para o tipo fração definido.**

#include <stdbool.h>

#include <stdio.h>

typedef struct {

int numerador;

int denominador;

} fracao;

void printFrac(fracao x) {

printf("%02d\n", x.numerador);

printf("--\n");

printf("%02d\n", x.denominador);

}

fracao somaFrac(fracao a, fracao b) {

fracao c;

c.numerador = (a.numerador \* b.denominador + b.numerador \* a.denominador);

c.denominador = a.denominador \* b.denominador;

return c;

}

fracao multFrac(fracao a, fracao b) {

fracao c;

c.numerador = (a.numerador \* b.numerador);

c.denominador = a.denominador \* b.denominador;

return c;

}

fracao divFrac(fracao a, fracao b) {

fracao c;

c.numerador = (a.numerador \* b.denominador);

c.denominador = b.numerador \* a.denominador;

return c;

}

bool igualdadeFrac(fracao a, fracao b) {

if (a.numerador \* b.denominador == a.denominador \* b.numerador)

return true;

else

return false;

}

int main() {

fracao a;

fracao b;

printf("Informe a fração A\n");

printf("numerador: ");

scanf("%d", &a.numerador);

printf("denominador: ");

scanf("%d", &a.denominador);

printf("\nFração informada:\n");

printFrac(a);

printf("Informe a fração B\n");

printf("numerador: ");

scanf("%d", &b.numerador);

printf("denominador: ");

scanf("%d", &b.denominador);

if (b.denominador == 0 || a.denominador == 0) {

printf("\n\nERRO FATAL: DIVISÃO POR ZERO!!!\n\n");

printf("Encerrando o programa.\n");

printf("Por favor não tente executar o programa novamente.\n");

return 1;

}

printf("\nFração informada:\n");

printFrac(b);

fracao c = somaFrac(a, b);

printf("\nSoma:\n");

printFrac(c);

c = multFrac(a, b);

printf("\nMultiplicação:\n");

printFrac(c);

c = divFrac(a, b);

printf("\nDivisão:\n");

printFrac(c);

printf("\nIgualdade:\n");

if (igualdadeFrac(a, b))

printf("Verdadeiro\n");

else

printf("Falso.\n");

return 0;

}