Fundamentos da Programação - Projeto 3

Autor: Dahan Schuster

Sobre a contribuição dos integrantes

Este trabalho foi feito de forma individual. Não por escolha, apesar de esta ser minha zono de coforto, confesso, mas por necessidade. Não consegui, ou não tentei o suficiente, encontrar um horário que fosse compatível com o trabalho em dupla. Na verdade, fiz o projeto no sábado dia 10/04, de última hora, pois resolvi focar o restante da semana nas outras disciplinas.

Fazendo então uma autoanálise, sei que poderia ter me esforçado mais para conseguir uma dupla e sair da minha zona de conforto, e me comprometo a consertar isto na próxima oportunidade. O projeto foi feito em algumas horas e poderia ter sido melhor desenvolvido se tivesse a contribuição de outra pessoa, mas pelo menos aprendi bastante e fiquei um pouco mais íntimo dos operadores bitwise, coisa que me assustava bastante antes de começar a fazer o curso. Estou animado em fazer mais questões que envolvam operações bit a bit, pois este tipo de programação é o que me encanta mais.

Detalhes extras sobre as funções

Primeira função: `calculaValorDevido`

A forma de cálculo tradicional pode ser recuperada ao igualar a largura da faixa ao peso do prato. Neste caso, um exemplo de chamada da função seria:

```
float peso = 80,
      custo_fixo = 6,
      preco_kg = 18;

// os últimos dois parâmetros (referentes ao desconto)
// não são utilizados na forma tradicional
calculaValorDevido(peso, custo_fixo, preco_kg, peso, 0, 0);
```

Neste caso, temos que:

```
// ...

float faixas_peso = peso / largura_faixa; /* faixas_peso  == 1 */
int  ultima_faixa = (int) faixas_peso;    /* ultima_faixa == 1 */

// ...

for (i = 0; i < faixas_peso; i++) {   /* apenas uma iteração será feita */
    prct_desconto = (desconto_faixa * i); /* o desconto será 0 */
```

```
    // ...

    if (i == ultima_faixa)            /* será sempre false */
}

// ...
```

Segunda função: `calculaParidade`

A verificação dos bits é feita índex por índex. A lógica é a seguinte: tendo uma constante com um único bit ativo e sabendo a posição deste bit, verificamos se o mesmo bit está ativo em `b`. Isto é feito da seguinte forma:

A diretiva `BIT_INICIAL` guarda o valor utilizado como referência para verificar o bit ativo. Este valor equivale a `00000001`. Em um loop, podemos ir deslocando o bit ativo para a esquerda em cada iteração com o operador `<<`. No caso, movemos o bit `i` vezes à esquerda com a operação `BIT_INICIAL << i`, e salvamos este valor na variável `bitAtivoEmI`.

Com isto, basta fazer outra verificação bit a bit com o operador `&` para verificar se `b` possui o bit na posição `i` ativo. A operação `b & bitAtivoEmI` irá retornar um byte contento todos os bits que estão ativos em `b` e `i` simultaneamente. Como sabemos que `bitAtivoEmI` contém apenas **um** bit ativo, podemos deduzir que um valor diferente de 0 indica que o bit na posição `i` está ativo em `b`, e com isso incremetar a quantidade de bits setados em `b`.

Por fim, basta retornar o resultado booleano da comparação `bitsAtivos % 2 != 0`, que siginifica "`b` possui um número ímpar de bits ativos?" e resulta em 0 ou 1.

Sobre os defafios

O maior desafio foi fazer o *click* na cabeça a respeito dos operadores bitwise. Como estou com programação de alto nível há 4 anos, este tipo de operação parece coisa de outro mundo. O comentário particular que a professora Leyza fez em uma das listas que enviei, sobre uma questão-desafio que envolvia LEDs, pode confirmar isto rsrs

A primeira função foi bem parecida com o Projeto 1, então não chegou a ser muito complicada, apesar de eu ter certeza que poderia ter sido melhor.

A segunda função me fez quebrar a cabeça. Tentei diversas outras opções que não usavam bitwise. O que me fez ir por esse caminho foi o comentário de Leyza que citei acima.

A terceira função não foi muito complicada. Tentei inicialmente multiplicar os chars por 1, 10, 100 e 1000, ainda pensando em números decimais. O resultado, obviamente, deu errado, e a próxima ideia foi a de multiplicar então pelo valor hexadecimal equivalente a 1 (0x01), 10 (0x0A), 100 (0x64) e 1000 (0x3E8), o que também não funcionou. Não imaginei de primeira que era só literalmente adicionar 0x na frente dos múltiplos de 10. Quando fiz isso (`c2 * 0x10`), cheguei a um resultado quase correto. O último passo foi entender que, ao invés de adicionar apenas um zero, era necessário adicionar dois para cada char.

Sobre a superação dos desafios

Como disse na seção anterior, tive que buscar mais informação sobre operadores bitwise para conseguir chegar a uma solução para a segunda função. Pesquisei sobre a utilidade de cada operador e como eles são usados, e revisei o resumo que fiz sobre estes operadores na segunda lista (arquivo bitwise.c), que acabou vindo bem a calhar.

Finalização

Por fim, agradeço pela oportunidade de aprender tanto e pelos desafios muito bem planejados. Vocês são ótimos professores!