1. **A correção para o problema apontado pode ser feita da seguinte forma:**

```
void InsereListaSE(tListaSE *lista, tConteudo conteudo)
{
   tNoListaSE *ptrNovoNo;
   tNoListaSE *ultimoNo;

   ASSEGURA(ptrNovoNo = malloc(sizeof(tNoListaSE)), "Nao foi
possivel alocar no");

   ptrNovoNo->conteudo = conteudo;
   ptrNovoNo->proximo = NULL; // O novo nó será o último,
portanto, seu próximo é NULL

   if (*lista == NULL) {
      // Se a lista estiver vazia, o novo nó será o primeiro nó
      *lista = ptrNovoNo;
   } else {
      // Caso contrário, encontre o último nó e insira o novo nó
após ele
      ultimoNo = *lista;
      while (ultimoNo->proximo != NULL) {
         ultimoNo = ultimoNo->proximo;
      }
      ultimoNo->proximo = ptrNovoNo;
   }
}
```

2. **Para definir o tipo tCADEIA podemos adotar a seguinte abordagem:**

```
typedef struct tCADEIA {
   unsigned char byte;
   struct tCADEIA *prox;
} tCADEIA;
```

Tipo tWAV:

```
typedef struct tWAV {
   unsigned int numCanal;
   unsigned int taxaAmostragem;
   unsigned int bitDepth;
   unsigned int tamanhoData;
   tCADEIA *dados;
} tWAV;
```

3. **Podemos efetuar testes para verificar se a taxa de amostragem é válida:**

```
int setMono(tWAV *wav, float *canalMono) {
    if (!wav) {
        printf("Erro: Ponteiro para tWAV é nulo!\n");
        return 0;
    }
    if (wav->taxaAmostragem <= 0) {
        printf("Erro: Taxa de amostragem inválida!\n");
        return 0;
    }
    // Restante da função...
    return 1; // Sucesso
}
```

Aqui temos uma versão em que os parâmetros canalMono, canalR e canal são listas encadeadas:

```
#include <stdio.h>
#include <stdlib.h>

// Estrutura para representar um nó da lista encadeada
typedef struct tNoLista {
    float amostra;
    struct tNoLista *prox;
} tNoLista;

// Função para inserir um novo nó no final da lista
void InsereNoLista(tNoLista **lista, float amostra) {
    tNoLista *novoNo = malloc(sizeof(tNoLista));
    if (!novoNo) {
        printf("Erro: Falha na alocação de memória!\n");
        exit(1);
    }
    novoNo->amostra = amostra;
    novoNo->prox = NULL;

    if (*lista == NULL) {
        // Se a lista estiver vazia, o novo nó será o primeiro nó
```

```
        *lista = novoNo;
    } else {
        // Caso contrário, encontre o último nó e insira o novo nó após ele
        tNoLista *ultimoNo = *lista;
        while (ultimoNo->prox != NULL) {
            ultimoNo = ultimoNo->prox;
        }
        ultimoNo->prox = novoNo;
    }
}

// Função para liberar a memória alocada para a lista
void LiberaLista(tNoLista *lista) {
    while (lista) {
        tNoLista *prox = lista->prox;
        free(lista);
        lista = prox;
    }
}
```