Erros Frequentes no Teste Global de 27 de Janeiro de 2010

Apresenta-se uma lista de ERROS frequentes, realizados pelos alunos na resolução do teste global. Cada entrada da lista indica algo que NÃO está correcto. Recomenda-se vivamente a resolução e verificação dos problemas do teste, quer em papel, quer em ambiente de desenvolvimento, tendo em especial atenção os erros típicos que são indicados abaixo.

1.

a)

- Colocar o pré-processador a avaliar expressões.
- Omitir o operador * antes de "%s"
- Reduzir o número parêntesis das expressões.

b)

- Indicar warning por haver conversão implícita para int da expressão 31/3.
- Indicar warning por haver conversão implícita da string "%s" em carácter.
- Assinalar erros de compilação em vez de warnings.
- Apresentar no output um valor que não seja inteiro.
- Não apresentar no output o carácter '%'.

2.

a)

- Declarar outras "coisas" que não sejam variáveis globais.
- Colocar os símbolos locais ao módulo como variáveis locais de funções do módulo.
- Colocar os símbolos T ou t com variáveis do tipo char* , char[] ou FILE*.

b)

- Justificar que _f e/ou _a estão indefinidos num dos módulos.
- Justificar que símbolos com o mesmo nome são de tipos diferentes em cada módulo.

3.

a)

- Manter dados em `eax`, `ecx` e/ou `edx` apesar do `call`.
- Tentar repor valores em `eax`, `ecx` e/ou `edx` por `pop` dos argumentos do `call`, esquecendo-se de que a função chamada pode ter modificado os argumentos em *stack*.
- Em vez de `v`, invocar `fx` com os primeiros quatro *bytes* apontados por `v`.
- Não corrigir o *stack* após o `call` ou corrigir para o lado errado.
- Em cada iteração, voltar a ler o valor original de `v` do *stack* destruindo o incremento feito na iteração anterior. [*uma distracção estranhamente comum!*]
- Avançar `v` antes do primeiro `call`.

b)

- Código pouco natural em C (ex.: `while` em vez de `for` ou `a[0]` em vez de `*a`).

c)

- Passagem de ponteiro nulo ou de valor aleatório no parâmetro `str` de `toString`.
- Alocação de espaço insuficiente para resultado de `toString` (o resultado não é um *byte*, nem um ponteiro; são até 22 caracteres, incluindo terminador!). E as alocações em *stack* têm de ser de dimensão múltipla de 4.
- Alocação estática ou dinâmica de espaço para resultado de `toString`, embora possíveis, são soluções inferiores à alocação em *stack*.
- Alguns alunos passam em `str` de `toString` um ponteiro para o *stack*, mas sem ter reservado o espaço (com `sub esp, 24`), pelo que os `push`s e `call`s das instruções seguintes ficam sobrepostos nesse espaço.
- Reaproveitamento de valor colocado em *stack* como argumento (nomeadamente o ponteiro `str`). Tal não é possível, uma vez que as funções chamadas podem alterar os argumentos que lhes foram passados.
- Passagem de argumentos pela ordem errada. [*um erro estranhamente comum, tendo em conta que a consulta de argumentos, de um modo geral, é feita correctamente!*]
- Manter dados em `eax`, `ecx` e/ou `edx` apesar dos `call`s.
- Considerar que existem valores de retorno em `eax` para funções com retorno `void`.
- Invocar `puts` em ciclo, como se fosse `putchar`.
- Não corrigir o *stack* após `call` ou corrigir para o lado errado.

d)

- Alocação automática do *array* de `VC` com dimensão não constante é sempre ilegal em C90. Deveria ter sido usada alocação dinâmica.
- Alocação de *array* de `VC *` em vez de *array* de `VC`.
- Não libertação da memória alocada dinamicamente.
- Leitura deficiente de valores inteiros do *standard input* (`getchar` simples ou outro, em vez de `scanf`, `atoi(gets)` ou função *ad-hoc*).
- Invocação de `addValue` com 2º parâmetro errado. Deveria ser passado o endereço de um inteiro, iniciado a zero fora do ciclo.
- Apresentação dos valores no *standard output* sem recorrer a `callAll`.
- Passar como 3º argumento de `callAll` o resultado da invocação de `print`, em vez do seu endereço.
- O número de valores a apresentar não é necessariamente o máximo (que foi passado como argumento do `main`), mas sim a dimensão real da tabela.
- Estranhas assinaturas de `main`.
- Utilização directa de *string* com dígitos como se fosse um inteiro!

e)

- Não inclusão de *srepl.s* no processo de *build*.
- Não inclusão de *print.s* no processo de *build*.
- Não inclusão de *prog.c* no processo de *build*.
- Utilização de argumentos inapropriados na ligação: `-Wall -pedantic -g -O2`
- Compilação de ficheiros *.h*!

5. *Os erros estão listados por ordem decrescente de ocorrência*

- Colocação do construtor na tabela de métodos virtuais (construtor virtual).
- Colocação do ponteiro para a tabela de métodos virtuais no final das estruturas.
- Não tipificação do ponteiro para a tabela de métodos virtuais (definido como `void *`) .
- Omissão de *casts* no ponteiro `this` (definido como `Scrambler *`) dos métodos ou na construção das tabelas de métodos virtuais dos *scramblers* concretos.
- Omissão do ponteiro para os métodos *`prepare`* na definição do tipo que representa a tabela de métodos virtuais.
- Omissão do tipo abstracto ou interface `Scrambler.`
- Colisão nos nomes dos métodos dos vários tipos.
- Tabelas de métodos com número de entradas distinto (omissão de `prepare` no `Scrambler0`)
- Definição de operações de alocação dinâmica para as classe concretas não era necessária.
    i. Iniciação do ponteiro para tabela de métodos virtuais nos métodos `new`.
- (Tentativa de) iniciação das *vtables* nos construtores [*tentativa porque nem sequer foi alocado espaço*].
- Colocação de campos no tipo abstracto `Scrambler`.
- Não iniciação do ponteiro para as *vtables* nos construtores.
- Afectação com `NULL` na entrada do método `prepare` na *vtable* do `Scrambler0`.
- Não tipificação dos ponteiros para métodos (definidos como `void *`) no tipo das tabelas de métodos virtuais.
- Definição das *vtables* como variáveis locais dos construtores.
- Tipo `Scrambler` corresponde ao tipo da tabela de métodos virtuais.

Bom estudo!