Segurança de Ponteiros em C e C++

Akim L. Pizutti, Pedro C. Beck, Rafael M. Rache

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul
Porto Alegre – RS – Brasil

alpizutti@inf.ufrgs.br, pedro.beck@inf.ufrgs.br, rafael.rache@inf.ufrgs.br

Abstract. *This meta-paper describes the style to be used in articles and short papers for SBC conferences. For papers in English, you should add just an abstract while for the papers in Portuguese, we also ask for an abstract in Portuguese ("resumo"). In both cases, abstracts should not have more than 10 lines and must be in the first page of the paper.*

Resumo. *Este meta-artigo descreve o estilo a ser usado na confecção de artigos e resumos de artigos para publicação nos anais das conferências organizadas pela SBC. É solicitada a escrita de resumo e abstract apenas para os artigos escritos em português. Artigos em inglês deverão apresentar apenas abstract. Nos dois casos, o autor deve tomar cuidado para que o resumo (e o abstract) não ultrapassem 10 linhas cada, sendo que ambos devem estar na primeira página do artigo.*

1. Introdução

Na década de 1980, Bjarne Stroustrup, um programador dinamarquês, viu a necessidade de uma nova linguagem de programação mais avançada e poderosa do que as disponíveis na época, como C e Pascal, criando assim o C++.

O nome "C++" foi escolhido por Stroustrup para se referir à evolução da linguagem C, na qual ele se baseou. O símbolo "++" na programação significa incremento de uma unidade e sugere a melhoria da linguagem C original.

A primeira versão pública do C++ foi lançada em 1985, e a linguagem ganhou rapidamente popularidade entre programadores de todo o mundo. A linguagem é caracterizada por sua combinação de recursos de programação orientada a objetos e recursos de programação de baixo nível, permitindo aos programadores criarem sistemas de software complexos que são eficientes e fáceis de manter.

Ao longo dos anos, o C++ evoluiu continuamente, com a introdução de novos recursos e melhorias de desempenho. Em 1998, a linguagem foi formalmente padronizada pela ISO (Organização Internacional de Padronização) e é considerada uma das linguagens de programação mais importantes e influentes da história da computação.

As linguagens de programação C e C++ estão intimamente relacionadas, mas têm algumas diferenças significativas. Uma das suas diferenças será abordada nesse artigo, segurança de ponteiros.

2. Endereços

Antes de abordar a segurança de ponteiros, serão estabelecidos alguns conceitos antes, o primeiro é o de endereços de memória.

Programas de computador utilizam um espaço da memória para armazenar seus dados, esse espaço de memória alocado para um programa é dividido em endereços de memória, que são valores indicando a posição em que um dado está na memória. Esse número pode ser absoluto ou relativo, dependendo do modo de endereçamento utilizado. As diferenças entre os modos de endereçamento são importantes para programação em baixo nível, mas no caso de C e C++ não é necessário se preocupar com isso.

Os dados de um programa ocupam um certo número de bytes consecutivos na memória do computador, dependendo de seu tipo. Por exemplo, em C e C++ uma variável do tipo `char` ocupa um byte de memória, e uma variável do tipo `int` pode ocupar 2 ou 4 bytes, dependendo da arquitetura.

3. Ponteiros

Em C e C++, um ponteiro é uma variável que contém o endereço de memória de outra variável. Em outras palavras, um ponteiro aponta para a localização de uma variável em memória. O conceito de ponteiros é uma das características mais importantes e poderosas das linguagens C e C++.

Os ponteiros em C e C++ são declarados usando o operador de asterisco (`*`). Para declarar um ponteiro, o nome da variável é precedido pelo operador de asterisco. Por exemplo, a declaração `int *p;` declara um ponteiro chamado `p` para um valor inteiro.

Os ponteiros são frequentemente usados em C e C++ para acessar e manipular dados na memória. Ao usar um ponteiro, um programa pode acessar diretamente a memória que armazena uma variável. Por exemplo, um programa pode usar um ponteiro para percorrer um array de valores, usando-o para acessar diretamente cada valor do array na memória.

Os ponteiros também são usados em C e C++ para alocar memória dinamicamente. Em vez de alocar um bloco fixo de memória para uma variável, um programa pode usar um ponteiro e alocar a quantidade necessária de memória para guardar a variável em tempo de execução.

Por mais que C e C++ tenham o conceito de ponteiros, existem algumas diferenças importantes na maneira como os ponteiros são usados em C e C++. Em C, um ponteiro é simplesmente um endereço de memória. Ele pode ser usado para acessar e manipular a memória diretamente, sem a necessidade de qualquer informação adicional. Isso torna os ponteiros em C mais poderosos, mas também mais perigosos, pois é fácil causar vazamentos de memória, estouros de buffer, e outros tipos de erros.

4. Diferenças de Ponteiros em C e C++

Os ponteiros em C possuem tipagem fraca, ou seja, é permitido atribuir um ponteiro do tipo `void*` para qualquer outro tipo de ponteiro sem necessidade de conversão de tipos, essa conversão é realizada implicitamente. Essa característica é bastante utilizada com as funções de alocação dinâmica de memória da biblioteca padrão `malloc()` e `calloc`, a seguir temos alguns exemplos de construções válidas em C que não são válidas em C++:

```
int *p = malloc(5 * sizeof(int));
int *q = calloc(5, sizeof(int));
char *c = "abc";
```

C++ tem regras de tipagem mais restritas e não permite as construções acima, porque é necessário converter explicitamente o tipo dos ponteiros. As construções acima devem ser modificadas para serem construções válidas em C++:

```
int *p = (int*)malloc(5 * sizeof(int));
int *q = (int*)calloc(5, sizeof(int));
char *c = (char*)"abc";
```

No caso das funções `malloc()` e `calloc()`, é necessário realizar a conversão explícita para atribuir a um ponteiro do tipo `int*` porque o tipo de retorno das funções é `void*`; no terceiro caso, é C++ não permite a conversão implicita de strings para `char*`.

4.1. Subsections

The subsection titles must be in boldface, 12pt, flush left.

5. Figures and Captions

Figure and table captions should be centered if less than one line (Figure 1), otherwise justified and indented by 0.8cm on both margins, as shown in Figure 2. The caption font must be Helvetica, 10 point, boldface, with 6 points of space before and after each caption.

6. References

Bibliographic references must be unambiguous and uniform. We recommend giving the author names references in brackets, e.g. [**?**], [**?**], and [**?**].

The references must be listed using 12 point font size, with 6 points of space before each reference. The first line of each reference should not be indented, while the subsequent should be indented by 0.5 cm.