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 paper discusses the usage of pointers in C and C++. The concept of pointers is explained and how they are delcared and used in both languages. The differences in pointer typing between C and C++ are presented, as well as the safety features introduced in C++ to help developers work with pointers in a safe manner.*

Resumo. *Este artigo aborda a utilização de ponteiros em C e C++. O conceito de ponteiros é explicado e como eles são declarados e utilizados nas linguagens. São apresentadas as diferenças entre a tipagem de ponteiros em C e C++ e os mecanismos de segurança introduzidos em C++ para utilizar ponteiros de maneira mais segura.*

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, C++ não permite a conversão implicita de strings para `char*`.

Além disso, C++ possui novos operadores para alocar e desalocar memória, `new` é uma palavra-chave utilizada em C++ para alocar memória e inicializar um objeto dinamicamente na execução, e para desalocar a memória de um objeto inicializado com `new`, usa-se a função `delete`. Pode-se argumentar que alocar memória com `new` é mais seguro que usar as funções `malloc()` ou `calloc()` porque não é necessário informar o espaço que deverá ser alocado para uma variável, isso já é tratado internamente pelo programa.

5. Segurança de Ponteiros em C++

A segurança de ponteiros é uma preocupação tanto em C quanto C++, especialmente quando se trata de alocação e desalocação dinâmica de memória.

Em C++, as keywords `new` e `delete` ajudam a tornar o uso de ponteiros mais seguro, pois garantem que a memória seja corretamente alocada e desalocada. Com `new`, é possível criar objetos dinamicamente e inicializá-los, enquanto `delete` libera a memória alocada para esses objetos. O uso de `new` e `delete` também ajuda a prevenir vazamentos de memória e estouros de buffer, o que pode ocorrer com o uso inadequado de `malloc()` e `free()`. Além disso, os ponteiros em C++ não permitem conversão implicita de tipos, ou seja, é necessário indicar o tipo do objeto que está sendo alocado com o uso de funções da biblioteca padrão de C, o que ajuda a evitar erros de tipo.

C++ também possui a keyword `nullptr`, que é um valor especial que pode ser atribuído a um ponteiro para indicar que ele não aponta para nenhum endereço válido de memória. Ao invés de utilizar o valor 0 ou `NULL` para indicar que um ponteiro é nulo, utilizar `nullptr` é mais claro para alguém que está lendo o código, e pode evitar erros.

5.1. Exceções

As exceções em C++ são frequentemente utilizadas para tratar erro durante a alocação de memória com ponteiros. Por exemplo, se um programa tentar alocar uma grande quantidade de memória com `new` e não houver memória suficiente disponível, uma exceção do tipo `std::bad_alloc` será lançada.

6. Conclusão

Ponteiros são uma das características masi importantes e poderosas de C e C++. Eles permitem o acesso direto à memória e alocação dinâmica de memória. No entando, o uso incorreto de ponteiros pode levar a erros de segurança e bugs no programa, como vazamentos de memória e estouros de buffer. C++ possui alguns mecanismos extras para auxiliar desenvolvedores na utilização de ponteiros e prevenir erros, que é uma de suas diferenças de incompatibilidade com C.

7. Bibliografia

ISO/IEC. (2017). Working Draft, Standard for Programming Language C++. 2017. Disponível em: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf.

Stroustrup, B. (1999). An Overview of the C++ Programming Language. Disponível em https://web.archive.org/web/20120816122304/http://www.stroustrup.com/crc.pdf