UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

06 – Programmi sicuri in C

INGEGNERIA INFORMATICA

Programmazione Avanzata

Anno di corso: 1

Anno accademico di offerta: 2024/2025

Crediti: 6

Prof. Claudio MENGHI

Dalmine

08 Ottobre 2023

Svantaggi ad usare linguaggi astratti

C'è un prezzo da pagare se si vogliono usare linguaggi safe come Java, ...

- Prestazioni inferiori
  - per il controllo dei limiti nell'accesso agli array, garbage collection per evitare dangling pointers
- Impiego di maggiore memoria
  - per tenere informazione sui tipi, sulla dimensione degli array
- Annotazione dei tipi
  - maggiore verbosità nelle dichiarazioni
- Porting di codice già esistente in C
  - (per quanto Java abbia sintassi simile al C)

Vantaggi del C

Il C è tutt'oggi usato per molte applicazioni come il sistema operativo, i device drivers

- Ha prestazioni elevate
- Permette la gestione esplicita della memoria
- Permette il controllo della rappresentazione dei dati a basso livello
- Riuso del codice esistente già scritto in C

Alcune violazioni di sicurezza del C

Errori “spaziali” di accesso alla memoria

- Out of bound access, buffer overflow, …

Errori “temporali” di accesso alla memoria

- Dangling pointers,

Errori di cast

- Tra diversi tipi di puntatori, tra puntatori e dati interi, tipi unione

Memory leaks

- Programmi che non rilasciano la memoria anche quando non serve più

Come rendere il C safe?

Come rendere il C safe?

- Restringere il linguaggio (non permettere all'utente di scrivere codice unsafe)
    - Vantaggi: codice safe per definizione
    - Svantaggi: limitiamo l'espressività

Come rendere il C safe?

- Restringere il linguaggio (non permettere all'utente di scrivere codice unsafe)
  - Vantaggi: codice safe per definizione
  - Svantaggi: limitiamo l'espressività

- Controlliamo mediante opportune procedure se abbiamo scritto codice unsafe
  - Vantaggi: non limitiamo l'espressività
  - Svantaggi: perdiamo delle violazioni (a meno che risolviamo l'halting problem)
    troviamo delle violazioni che non sono violazioni

Come rendere il C safe?

- Restringere il linguaggio (non permettere all'utente di scrivere codice unsafe)
  - Vantaggi: codice safe per definizione
  - Svantaggi: limitiamo l'espressività

- Controlliamo mediante opportune procedure se abbiamo scritto codice unsafe
  - Vantaggi: non limitiamo l'espressività
  - Svantaggi: perdiamo delle violazioni (a meno che risolviamo l'halting problem)
  - Può essere fatto in:
    - Modo statico: compile time
    - Modo dinamico: durante l'esecuzione

Come rendere il C safe?

1. Tools per l'analisi statica e dinamica per trovare safety violations
   - esempio **Purify** della Rational/IBM è un tool per l'analisi dinamica per scoprire errori di accesso alla memoria, valgrind
   - Analisi statica: cppcheck, lint ...

2. Librerie per rendere il programmi C safe

3. Uso di un sottoinsieme «safe»

4. Tools, e linguaggi per prevenire safety violation con due approcci distinti
   1. rendere sicuri programmi C : SafeC, CCured
   2. varianti safe del C: Cyclone, Vault

Analisi statica con splint

- http://www.splint.org/
- Splint, short for Secure Programming Lint, is a programming tool for statically checking C programs for security vulnerabilities and coding mistakes. Formerly called LCLint, it is a modern version of the Unix lint tool.
- Splint has the ability to interpret special annotations to the source code, which gives it stronger checking than is possible just by looking at the source alone.
- Ultima versione 2007 (Ultimo aggiornamento 2010)

Esempio di splint

```
#include <stdio.h>
int main()
{
    char c;
    while (c != 'x');
    {
        c = getchar();
        if (c = 'x')
            return 0;
        switch (c) {
        case '\n':
        case '\r':
            printf("Newline\n");
        default:
            printf("%c",c);
        }
    }
    return 0;
}
```

- Variable c used before definition
- Suspected infinite loop. No value used in loop test (c) is modified by test or loop body.
- Assignment of int to char: c = getchar()
- Test expression for if is assignment expression: c = 'x'
- Test expression for if not boolean, type char: c = 'x'
- Fall through case (no preceding break)

Analisi dinamica con Purify

approccio analisi dinamica

input programmi C/C++ (qualsiasi)

output eseguibili linkati con Purify

metodo inserimento di controlli per trovare durante l'esecuzione errori di accesso alla memoria o memoria non rilasciata

pro si applica a codice già esistente

contro rallenta l'esecuzione e non garantisce la scoperta di ogni errore

info http://www-306.ibm.com/software/awdtools/purify/

https://www.ibm.com/support/pages/tools-purify

valutazione: ★★★

ANCORA ATTIVO!

Analisi con valgrind

- Valgrind is an instrumentation framework for building dynamic analysis tools.
- The Valgrind distribution currently includes six production-quality tools:
  - a memory error detector,
  - two thread error detectors,
  - a cache and branch-prediction profiler, a call-graph generating cache and branch-prediction profiler, and a heap profiler. It also includes three experimental tools: a heap/stack/global array overrun detector, a second heap profiler that examines how heap blocks are used, and a SimPoint basic block vector generator.
  - It runs on the following platforms: X86/Linux, AMD64/Linux, ARM/Linux, PPC32/Linux, PPC64/Linux, X86/Darwin and AMD64/Darwin (Mac OS X 10.5 and 10.6).
- Is open source (www.valgrind.org)
- ANCORA ATTIVO!

Esempio con valgrind

```
#include <stdlib.h>
  void f(void)
  {
     int* x =
       malloc(10 * sizeof(int));
     x[10] = 0;
  }

  int main(void)
  {
     f();
     return 0;
  }
```

- Compile your program with -g to include debugging information so that Memcheck's error messages include exact line numbers.
- valgrind --leak-check=yes myprog
- Che errori trova …

- Attenti non trova alcuni errori (overrun di variabili statiche)

Esempio uso con https://github.com/dynamorio/drmemory

- Per windows potete usare drmemory
- Cpp check
- https://cppcheck.sourceforge.io/
- .. Tanti altri

Librerie Safe per le stringhe

Scopo: evitare i buffer overflow e altri problemi tipici delle stringhe e dei buffer di char

1. Safe C String Library

http://www.zork.org/safestr/

Autori: Matt Messier e John Viega

2. ISO/IEC TR 24731

Meyers, Randy. Specification for Safer, More Secure C Library Functions, ISO/IEC TR 24731, June 6, 2004

www.open-std.org/jtc1/sc22/wg14/www/docs/n1172.pdf

Supportato da Microsoft

SafeC

approccio	traduttore da C a C (C fatto sicuro)
input	programmi C (qualsiasi)
output	programmi C sicuri
metodo	garantisce la cattura delle violazioni di memoria e vari errori run time inserendo dei controlli e aggiungendo informazioni (ad esempio ai puntatori)
pro	si applica a C codice già esistente
contro	rallenta l'esecuzione e aumenta la memoria necessaria
info	http://www.eecs.umich.edu/~taustin/

valutazione: ★

http://sourceforge.net/projects/safeclib/

- The Safe C Library provides bound checking memory and string functions per ISO/IEC TR24731. These functions are alternative functions to the existing Standard C Library. The functions verify that output buffers are large enough for the intended result, and return a failure indicator if they are not. Optionally, failing functions call a "runtime-constraint handler" to report the error. Data is never written past the end of an array. All string results are null terminated. In addition, the functions in ISO/IEC TR 24731-1:2007 are re-entrant: they never return pointers to static objects owned by the function.

Uso di sottoinsieme del C

- Posso decidere di usare un sottoinsieme del C
- Senza aritmetica dei puntatori?
- Senza puntatori?

MISRA C

IAR Systems implements the The Motor Industry Software Reliability Association's *Guidelines for the Use of the C Language in Vehicle Based Software.*

MISRA C describes a subset of C, suited for use when developing safety-critical systems.

MISRA C is a set of rules to be checked by the compiler. A message is generated for every deviation from a required or advisory rule, unless you have disabled it. Each message contains a reference to the MISRA C rule deviated from.

Categories of Rules

Environment rules

Character sets

Comments

Identifiers

Types

Constants

Declaration and definitions

Initialization

Operators

Conversions

Expressions

Control flow

Functions

Preprocessing directives

Pointers and arrays

Structures and unions

Standard Libraries

Example – Comments rule

Rule 9 (required) Comments shall not be nested.

How the rule is checked?

The compiler will generate an error, indicating a violation of this rule, if /* is used inside a comment.

Rule 10 (advisory) Sections of code should not be 'commented out'.

How the rule is checked?

The compiler will generate an error, indicating a violation of this rule, whenever a comment ends with *;*, *{*, *or }*.

Note: This rule is checked in such a manner that code samples inside comments are allowed and do not generate an error.

Example – Identifier rule

Rule 12 (required)

No identifier in one namespace shall have the same spelling as an identifier in another namespace.

How the rule is checked?

The compiler will generate an error, indicating a violation of this rule, if a declaration or definition would hide an identifier if they were in the same namespace.

For example, fields of different structures will not generate an error.

Example of rule violations

```
struct an_ident { int an_ident; } an_ident;
```

Example of correct code

```
struct a_struct { int a_field; } a_variable;
```

Example – Types rule

Rule 13 (advisory) The basic types of char, int, short, long, float, and double should not be used,

but specific-length equivalents should be typedef'd for the specific compiler, and these type names used in the code.

How the rule is checked

The compiler will generate an error, indicating a violation of this rule, if any of the basic types given above is used in a declaration or definition that is not a typedef.

Example of rule violations

int x;

Example of correct code

typedef int SI_16

SI_16 x;

Example – Constant rule

Rule 18 (advisory) Numeric constants should be suffixed to indicate type, where an appropriate suffix is available.

How the rule is checked?

The compiler will generate an error, indicating a violation of this rule, for any integer constant whose type is not the same in any standard-conforming implementation.

Example of rule violations

100000

Examples of correct code

30000

100000L

100000UL

Sui puntatori:

MISRA rule that states the only pointer math allowed is the indexing operation.

CCured

approccio traduttore da C a C (C fatto sicuro)

input programmi C con annotazioni particolari (opzionali)

output programmi C sicuri

metodo abbina analisi statica, controlli dinamici e garbage collector

pro si applica a C codice già esistente con minime modifiche

contro rallenta l'esecuzione

info http://manju.cs.berkeley.edu/ccured/

valutazione: ★★

Cyclone

approccio	safe C-like language
input	programmi C modificati
output	programmi C sicuri
metodo	controlli dinamici solo dove necessario e garbage collector
pro	minimo overhead di tempo e di memoria
contro	richiede di modificare i programmi originali
info	http://cyclone.thelanguage.org/

valutazione: ★★★

Vault

approccio safe C-like language

input programmi C modificati

output COM objects

metodo linguaggio astratto simile a Java/C#

pro minimo overhead di tempo e di memoria

contro richiede di riscriver i programmi originali

info http://research.microsoft.com/vault/

OUTDATED

valutazione: ★★

Confronto sui linguaggi

Controllo sui dettagli a basso livello

- SafeC e CCured: pieno utilizzo del C, operazioni limitate sui puntatori
- Cyclone: più restrittivo del C
- Vault: meno efficiente, più astratto

Opzioni sulla gestione della memoria

- Cyclone: diverse opzioni
- Vault: oggetti “lineari” e regioni
- SafeC: malloc() e free() esplicite
- CCured: garbage collection

Confronto dei costi

Prestazioni

Cyclone ≈ Vault < CCured << SafeC

Aumento memoria

Cyclone ≈ Vault < CCured << SafeC

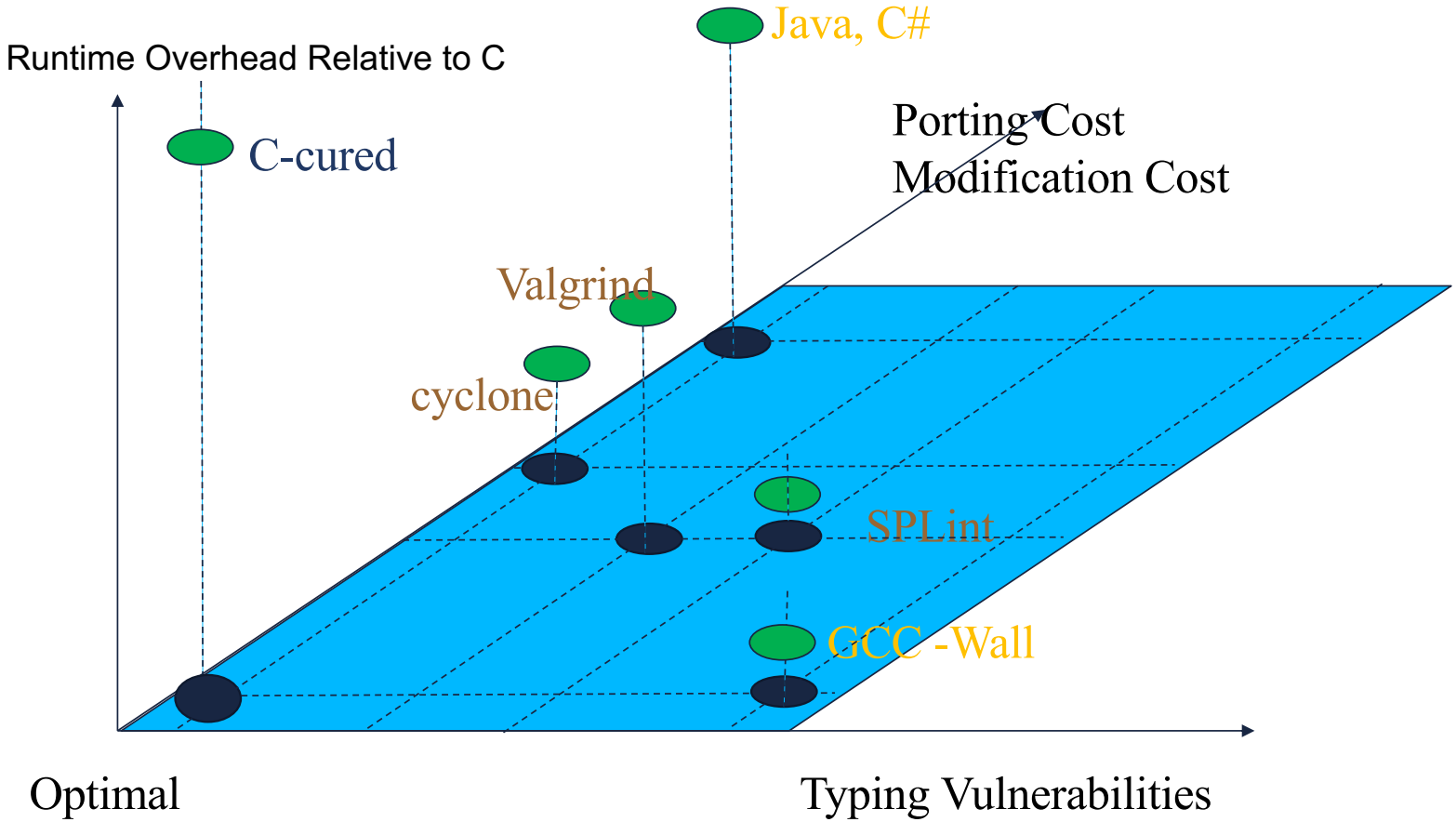
Sforzo per annotare i tipi

SafeC < CCured < Cyclone ≈ Vault

Sforzo per portate codice esistente

SafeC < CCured < Cyclone << Vault

Conceptual Space



In sintesi

- Abbiamo visto:
  - quali sono i pro e contro ad usare i linguaggi safe ad alto livello invece che il C
  - possibili modi di rendere il C safe
- Ricordate che:
  - è possibile effettuare l'analisi dinamica con tool come purify
- Per essere certi di avere programmi in C safe abbiamo visto e confrontato questi tool:
  - SafeC, CCured, Cyclone, Vault

UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

Domande?