



Narzędzia wspomagające programowanie w językach C/C++ dla systemu Linux

Autor: Adam Stolcenburg

- gcc – GNU Compiler Collection (dawniej GNU C Compiler)
 - <https://gcc.gnu.org/>
 - Projekt GNU
 - GNU GPL 3+ with GCC Runtime Library Exception
 - pierwsza wersja udostępniona 22 marca 1987 roku
 - najnowsza stabilna wersja 6.2 z 22 sierpnia 2016
- clang
 - <http://clang.llvm.org/>
 - Apple, Microsoft, Google, ARM, Intel, AMD
 - University of Illinois/NCSA Open Source License
 - pierwsza wersja udostępniona 11 lipca 2007
 - najnowsza stabilna wersja 3.9.0 z 2 września 2016

Jak używać kompilatorów



```
gcc source.cpp -o binary
```

- generowanie finalnego pliku wykonywalnego: -o
- generowanie pośrednich plików binarnych: -c
- definicja poziomu optymalizacji: -O0, -O1, -O2, -Os, -O3
- dodanie informacji debugowych: -ggdb
- definicja poziomu ostrzeżeń
 - clang – włączenie wszystkich ostrzeżeń: -Weverything
 - gcc – przykładowe opcje włączające ostrzeżenia: -pedantic -Wall -Wextra -Wcast-align -Wcast-qual -Wctor-dtor-privacy -Wdisabled-optimization -Wformat=2 -Winit-self -Wlogical-op -Wmissing-declarations -Wmissing-include-dirs -Wnoexcept -Wold-style-cast -Woverloaded-virtual -Wredundant-decls -Wshadow -Wsign-conversion -Wsign-promo -Wstrict-null-sentinel -Wstrict-overflow=5 -Wswitch-default -Wundef
- wyłączanie wybranych ostrzeżeń: -Wno-
- mnóstwo innych opcji opisanych na stronach elektronicznego podręcznika: man gcc, man clang

Błędy kompilacji



```
class MyClass { };  
void test(MyClass* my_class_ptr) {  
    MyClass my_class = my_class_ptr;  
}
```

- gcc

error.cpp: In function 'void test(MyClass*)':

error.cpp:3:22: error: conversion from 'MyClass*' to non-scalar type 'MyClass' requested

```
    MyClass my_class = my_class_ptr;  
                        ^
```

- clang

error.cpp:3:11: **error:** no viable conversion from 'MyClass *' to 'MyClass'

```
    MyClass my_class = my_class_ptr;  
        ^           ~~~~~
```

error.cpp:1:7: note: candidate constructor (the implicit copy constructor) not viable: no known conversion from 'MyClass *' to 'const MyClass &' for 1st argument; dereference the argument with *

```
class MyClass { };  
    ^
```

1 error generated.

```
void test(int end) {  
    for (unsigned int i = 0; i < end; ++i) {  
    }  
}
```

- gcc

warning.cpp: In function 'void test(int)':

warning.cpp:2:32: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]

```
    for (unsigned int i = 0; i < end; ++i) {  
                                ^
```

- clang

warning.cpp:2:30: **warning:** comparison of integers of different signs: 'unsigned int' and 'int' [-Wsign-compare]

```
    for (unsigned int i = 0; i < end; ++i) {  
        ~ ^ ~~~
```

1 warning generated.

Makra ułatwiające debugowanie

- `__FUNCTION__` - nazwa aktualnej funkcji
- `__PRETTY_FUNCTION__` - nazwa aktualnej funkcji wraz zakresem, parametrami i zwracaną wartością
- `__LINE__` - linia w pliku źródłowym
- `__FILE__` - nazwa pliku źródłowego

```
#include <stdio>
```

```
struct Test {
```

```
    static void print() {
```

```
        printf("%s\n%s\n%s %d\n", __FUNCTION__, __PRETTY_FUNCTION__, __FILE__, __LINE__);
```

```
    }
```

```
};
```

```
int main() { Test::print(); return 0; }
```

- wynik działania

```
print
```

```
static void Test::print()
```

```
macros.cpp 4
```

Make (1)

- narzędzie służące do automatyzacji procesu budowania plików wykonywalnych i bibliotek z plików źródłowych
- pierwsza wersja z 1977 roku
- GNU make - <https://www.gnu.org/software/make/>
- opis procesu budowania opisany w plikach “makefile”
- przepisy składające się z celów oraz listy zależności
- polecenia dla celów uruchamiane jeśli czas modyfikacji plików z listy zależności jest późniejszy niż czas modyfikacji celu
- polecenia budujące cel na podstawie listy zależności definiowane poniżej listy zależności w liniach rozpoczynających się od znaku tabulacji

```
target: target.cpp  
g++ target.cpp -o target
```

- możliwość wyboru celu z linii poleceń

```
make target
```

Make (2)



- definicja celów za pomocą wzorców dopasowania

```
%.o: %.cpp  
g++ $< -c
```

- parametryzowanie za pomocą zmiennych

```
%.o: %.cpp  
$(CXX) $< -c
```

- możliwość ustawienia zmiennych z lini poleceń

```
make CXX=clang target.o
```

- obsługa rozgałęzień

```
ifeq $(RELEASE),1  
CXXFLAGS=-O3  
else  
CXXFLAGS=-ggdb -O0  
endif
```

- obsługa wbudowanych funkcji: https://www.gnu.org/software/make/manual/html_node/Functions.html

```
SOURCES=$(wildcard *.cpp)  
OBJECTS=$(patsubst %.cpp,%.o,$(SOURCES))
```


Make – przykładowy plik “makefile”

```
SOURCES=$(wildcard *.cpp)
OBJECTS=$(patsubst %.cpp,%.o,$(SOURCES))
BINARY=example
```

```
ifeq ($(RELEASE),1)
CXXFLAGS=-O3
else
CXXFLAGS=-ggdb -O0 -Wall
endif
```

```
all: $(BINARY)
```

```
$(BINARY): $(OBJECTS)
    $(CXX) $^ -o $@
```

```
%.o: %.cpp
    $(CXX) $(CXXFLAGS) $< -c
```

```
clean:
    rm -f $(BINARY) $(OBJECTS)
```

- <https://www.kdevelop.org/>
- nowoczesne, darmowe zintegrowane środowisko programistyczne napisane w C++
- rozumie C++
 - zaawansowany system auto-uzupełniania
 - podświetlanie zmiennych na podstawie zakresu ich widoczności
 - wygodna nawigacja między funkcjami
- łatwa integracja z zewnętrznymi systemami budowania – make, cmake, qmake
- brak wbudowanego kompilatora
- obsługa debuggera gdb
- automatyczne formatowanie
- wsparcie dla refaktoryzacji
- wizualizacja hierarchii class
- integracja z systemami kontroli wersji
- wbudowane wsparcie dla Doxygena
- sprawdzanie pisowni wewnątrz komentarzy i literałów

- <http://cppcheck.sourceforge.net/>
- narzędzie służące do statycznej analizy kodu
- nacisk na unikanie tzw. false positives
- wykrywa następujące rodzaje błędów:
 - dostęp poza obszarem zaalokowanej pamięci
 - wycieki pamięci
 - dereferencje wskaźnika null
 - dostęp do niezainicjalizowanych zmiennych
 - nieprawidłowe użycie biblioteki STL
 - nieużywany oraz nadmiarowy kod
 - i wiele innych: <http://sourceforge.net/p/cppcheck/wiki/ListOfChecks>
- bezproblemowe użycie

cppcheck --enable=style,performance,portability <nazwa pliku/katalogu>

Cppcheck – przykładowe błędne kody

- bug1.cpp

```
void test() {  
    char tab[10];  
    for (int i = 0; i <= 10; ++i) {  
        tab[i] = 0;  
    }  
}
```

- bug2.cpp

```
class Base {  
};  
class Derived : public Base {  
    const char* m_ptr;  
public:  
    Derived() : m_ptr(new char) { }  
    ~Derived() { delete m_ptr; }  
};  
void test() {  
    Base* base_ptr = new Derived();  
    delete base_ptr;  
}
```

Cppcheck – rezultat działania

- `cppcheck bug1.cpp`

Checking bug1.cpp...

[bug1.cpp:4]: (error) Buffer is accessed out of bounds: tab

- `cppcheck bug2.cpp`

Checking bug2.cpp...

[bug2.cpp:1]: (error) Class 'Base' which is inherited by class 'Derived' does not have a virtual destructor.

Clang static analyzer

- <http://clang-analyzer.llvm.org/>
- narzędzie służące do statycznej analizy kodu
- część projektu clang
- zaawansowane algorytmy wnioskowania
- możliwość sprawdzenia pojedynczego pliku
scan-build g++ code.cpp -c
- możliwość sprawdzenia całego projektu bazującego na przykład na make, w czasie analizy ustawiane są między innymi zmienne środowiskowe CC i CXX
scan-build make all
- wizualizacja raportu ze szczegółowym opisem przepływu sterowania

Clang static analyzer – błędny kod

```
int* get_greater_than_zero_or_null(int value, bool force_zero) {  
    int* result_ptr = 0;  
  
    if (value > 0)  
        result_ptr = new int(value);  
    if (force_zero) *result_ptr = 0;  
  
    return result_ptr;  
}
```

Clang static analyzer – przykładowy raport

```
1 int* get_greater_than_zero_or_null(int value, bool force_zero) {  
2   int* result_ptr = 0;  
3  
4   if (value > 0)  
5     result_ptr = new int(value);  
6   if (force_zero) *result_ptr = 0;  
7  
8   return result_ptr;  
9 }
```

1 'result_ptr' initialized to a null pointer value →

2 ← Assuming 'value' is <= 0 →

3 ← Taking false branch →

4 ← Assuming 'force_zero' is not equal to 0 →

5 ← Taking true branch →

6 ← Dereference of null pointer (loaded from variable 'result_ptr')

- <http://ltp.sourceforge.net/coverage/lcov.php>
- narzędzie wizualizujące pokrycie kodu wygenerowane przez gcov
- gcov – standardowe narzędzie wchodzące w skład GCC
- współpracuje zarówno z GCC jak i clang
- możliwość łączenia raportów z wykonania wielu plików wykonywalnych
- wymaga kompilacji i linkowania kodu z opcją **--coverage** (generuje .gcno)
g++ --coverage -c code.cpp -o code
- oraz uruchomienia (generuje .gcda)
./code
- raport z plików .gcno oraz .gcda jest generowany za pomocą
lcov --capture --rc lcov_branch_coverage=1 --directory=. -o cov.info
genhtml --legend --rc lcov_branch_coverage=1 cov.info -o output

lcov – przykładowy raport



LCOV - code coverage report

Current view: [top level](#) - [report](#) - [code.cpp](#) (source / [functions](#))

Test: [cov.info](#)

Date: 2016-12-14 00:04:10

Legend: Lines:

hit

not hit

 | Branches:

+

 taken

-

 not taken

#

 not executed

Lines: 8988.9 %

Functions: 22100.0 %

Branches: 2450.0 %

	Branch data	Line data	Source code
1		1	int* get_greater_than_zero_or_null(int value, bool force_zero) {
2		1	int* result_ptr = 0;
3			
4	[- +]	1	if (value > 0)
5		0	result_ptr = new int(value);
6	[- +]	1	if (force_zero) *result_ptr = 0;
7			
8		1	return result_ptr;
9			}
10			
11		1	int main() {
12		1	get_greater_than_zero_or_null(0, false);
13		1	return 0;
14			}

Generated by: [LCOV version 1.12](#)

- <http://valgrind.org/>
- zbiór narzędzi służących do dynamicznej analizy kodu
 - memcheck – wykrywanie problemów związanych z niewłaściwym użyciem pamięci
 - cachegrind – profilowanie użycia cache
 - callgrind – cachegrind + grafy wywołań, profilowanie kodu
 - massif – monitorowanie użycia pamięci przez poszczególne części kodu
 - helgrind, DRD – wykrywanie problemów związanych z wątkami
- uruchamiany na pliku wykonywalnym
- zalecane ustawienia opcji kompilacji – -ggdb -O0
- wymaga uruchomienia analizowanego fragmentu kodu

Valgrind (memcheck)

- wykrywanie błędów związanych z niewłaściwym użyciem pamięci
 - wycieki pamięci
 - dostęp do złych obszarów pamięci (niezaalkowanych, zwolnionych itd.)
 - użycie niezainicjalizowanych zmiennych
 - niewłaściwe zwolnienie pamięci (wielokrotne, za pomocą złych funkcji)
- spowolnienie działania kodu od 10 do 30 razy
- domyślne narzędzie valgrinda, sposób uruchamiania
valgrind ./a.out
- opcje ułatwiające wykrycie wycieków pamięci
valgrind --leak-check=full ./a.out
- opcje ułatwiające znalezienie źródeł niezainicjalizowanych obszarów pamięci
valgrind --track-origins=yes ./a.out

Valgrind (memcheck) – raport

```
==28815== Memcheck, a memory error detector
==28815== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==28815== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==28815== Command: ./a.out
==28815==
==28815==
==28815== HEAP SUMMARY:
==28815==      in use at exit: 10 bytes in 1 blocks
==28815==    total heap usage: 2 allocs, 1 frees, 18 bytes allocated
==28815==
==28815== LEAK SUMMARY:
==28815==      definitely lost: 10 bytes in 1 blocks
==28815==      indirectly lost: 0 bytes in 0 blocks
==28815==      possibly lost: 0 bytes in 0 blocks
==28815==      still reachable: 0 bytes in 0 blocks
==28815==      suppressed: 0 bytes in 0 blocks
==28815== Rerun with --leak-check=full to see details of leaked memory
==28815==
==28815== For counts of detected and suppressed errors, rerun with: -v
==28815== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Valgrind (memcheck) – wyciek pamięci

```
#include <stdlib.h>

typedef struct {
    char* data_ptr;
} buffer_t;

buffer_t* get_buffer(int size) {
    buffer_t* result_ptr = (buffer_t*)malloc(sizeof(buffer_t));
    result_ptr->data_ptr = (char*)malloc(size * sizeof(char));
    return result_ptr;
}

void test_leak() {
    buffer_t* buffer_ptr = get_buffer(10);
    free(buffer_ptr);
}

int main() {
    test_leak();
    return 0;
}
```

Valgrind (memcheck) – raport wycieku pamięci



```
==28976== HEAP SUMMARY:
==28976==      in use at exit: 10 bytes in 1 blocks
==28976==    total heap usage: 2 allocs, 1 frees, 18 bytes allocated
==28976==
==28976== 10 bytes in 1 blocks are definitely lost in loss record 1 of 1
==28976==    at 0x4C2AB80: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==28976==    by 0x4005A2: get_buffer (memory_leak.c:9)
==28976==    by 0x4005C4: test_leak (memory_leak.c:14)
==28976==    by 0x4005E4: main (memory_leak.c:19)
```

Valgrind (memcheck) – niezainicjalizowana zmienna

```
#include <stdio>

static int counter = 0, sum = 0;

void increment(int i) { sum += i; }
void call30Increment() {
    for (int i = 0; i < 30; ++i) { increment(++counter); }
}
void call100Increment() {
    int counter;
    for (int i = 0; i < 100; ++i) { increment(++counter); }
}

int main() {
    call30Increment();
    call100Increment();
    printf("%d\n", sum);
    return 0;
}
```


Valgrind (memcheck) – raport niezainicjalizowanej zmiennej

```
==29191== Conditional jump or move depends on uninitialised value(s)
==29191==    at 0x4E8158E: vfprintf (vfprintf.c:1660)
==29191==    by 0x4E8B498: printf (printf.c:33)
==29191==    by 0x4005CF: main (uninitialized.cpp:16)
==29191== Uninitialised value was created by a stack allocation
==29191==    at 0x400580: call100Increment() (uninitialized.cpp:8)
```

Valgrind (callgrind)

- profilowanie czasu wykonania kodu
- spowolnienie działania kodu od 20 do 100 razy
- sposób uruchamiania

```
valgrind --tool=callgrind ./a.out
```

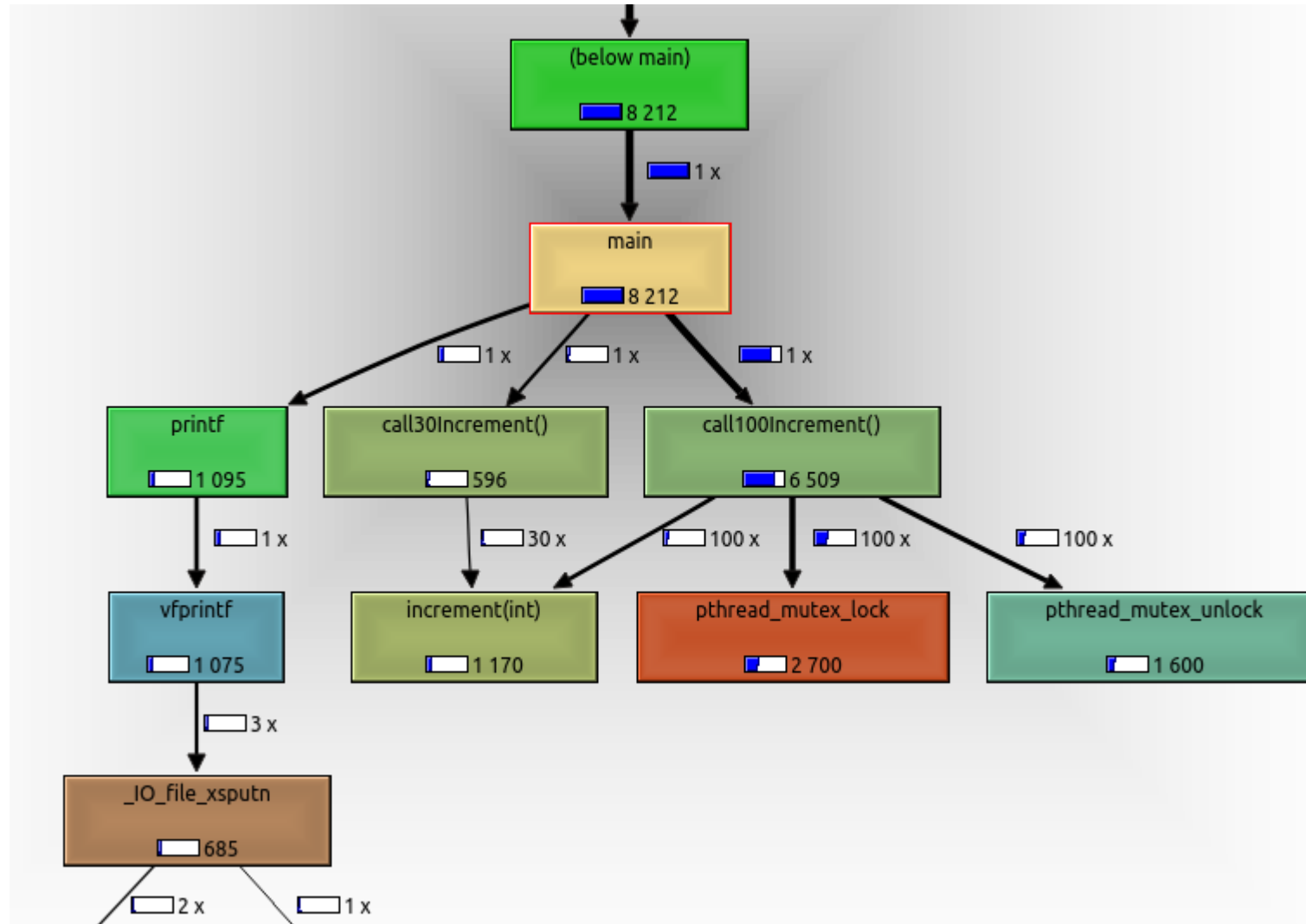
- możliwość kontroli w czasie działania za pomocą narzędzia callgrind_control
 - -i on – włączenie profilowania;
 - -i off – wyłączenie profilowania;
 - -z – zerowanie statystyk
- raport domyślnie generowany do pliku: callgrind.out.<pid>
- możliwość wizualizacji raportu za pomocą kcachegrind

Valgrind (callgrind) – profilowany kod

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

static int counter = 0, sum = 0;
static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
void increment(int i) { sum += i; }
void call30Increment() {
    pthread_mutex_lock(&mutex);
    for (int i = 0; i < 30; ++i) { increment(++counter); }
    pthread_mutex_unlock(&mutex);
}
void call100Increment() {
    for (int i = 0; i < 100; ++i) {
        pthread_mutex_lock(&mutex); increment(++counter); pthread_mutex_unlock(&mutex);
    }
}
int main() {
    call30Increment(); call100Increment(); printf("%d\n", sum);
    return 0;
}
```

kcachegrind – wizualizacja grafu wywołań





Dziękuję

adbglobal.com