

# PB071 – Programování v jazyce C

Všemožné zajímavosti, co by vás mohli  
zajímat 😊

# Návrhové vzory, antivzory, refactoring

# Návrhové vzory

- Návrhový vzor je opakovaně použitelné řešení pro často se vyskytující problém
  - [http://sourcemaking.com/design\\_patterns](http://sourcemaking.com/design_patterns)
- Často zmiňováno v kontextu objektově orientovaného programování, ale jde o obecný princip
- Např. Jak pracovat jednotným stylem s funkcemi používající jiné API?
  - funkce dělají stejné (nebo hodně podobné) věci
  - jsou ale programovány různými vývojáři => různé API
  - (C knihovny nebo např. callback funkce)
  - návrhový vzor Adapter (dodatečný kód vytvářející očekávané rozhraní)

```

if(server.is_file_in_database(path)){
    server.set_licence_data_from_database(path);
    char type;
    char constrain;
    bool right_input = false;
    permissions new_permissions = {{FULLY, 0, {0,0,0}}, {FULLY, 0, {0,0,0}}, {FULLY, 0, {0,0,0}}, {FULLY, 0, {0,0,0}}};
    do{
        cout<<endl<<"Enter type of file (t)text/(m)music/(e)executable: ";
        cin>>type;
        switch (type){
            case 't':
                right_input = true;
                // display
                cout<<"Enter constrain for display (n)no/(p)partially/f(fully): ";
                cin>>constrain;
                switch (constrain){
                    case 'n':
                        new_permissions.display.constricted = NO;
                        new_permissions.display.count = -1;
                        new_permissions.display.interval.year = -1;
                        break;
                    case 'p':
                        int count;
                        new_permissions.display.constricted = PARTIALLY;
                        cout<<"Count of display (-1 for not set): ";
                        cin>>count;
                        if(count > -1){
                            new_permissions.display.count = count;
                        }
                        else{
                            new_permissions.display.count = -1;
                        }
                        int year, month, day;
                        cout<<"Enter year (-1 for not set): ";
                        cin>>year;
                        if(year > -1){
                            new_permissions.display.interval.year = year;
                            cout<<"Enter month: ";
                            cin>>month;
                            new_permissions.display.interval.month = month;
                            cout<<"Enter day: ";
                            cin>>day;
                            new_permissions.display.interval.day = day;
                        }
                        else{
                            new_permissions.display.interval.year = -1;
                        }
                        break;
                    case 'f': //f is default value
                        break;
                }
            default:
                cerr<<"Wrong input type. Please insert n/p/f."<<endl;
                right_input = false;
                break;
        }
    } while (!right_input);
}

```

## ... a anti-vzory

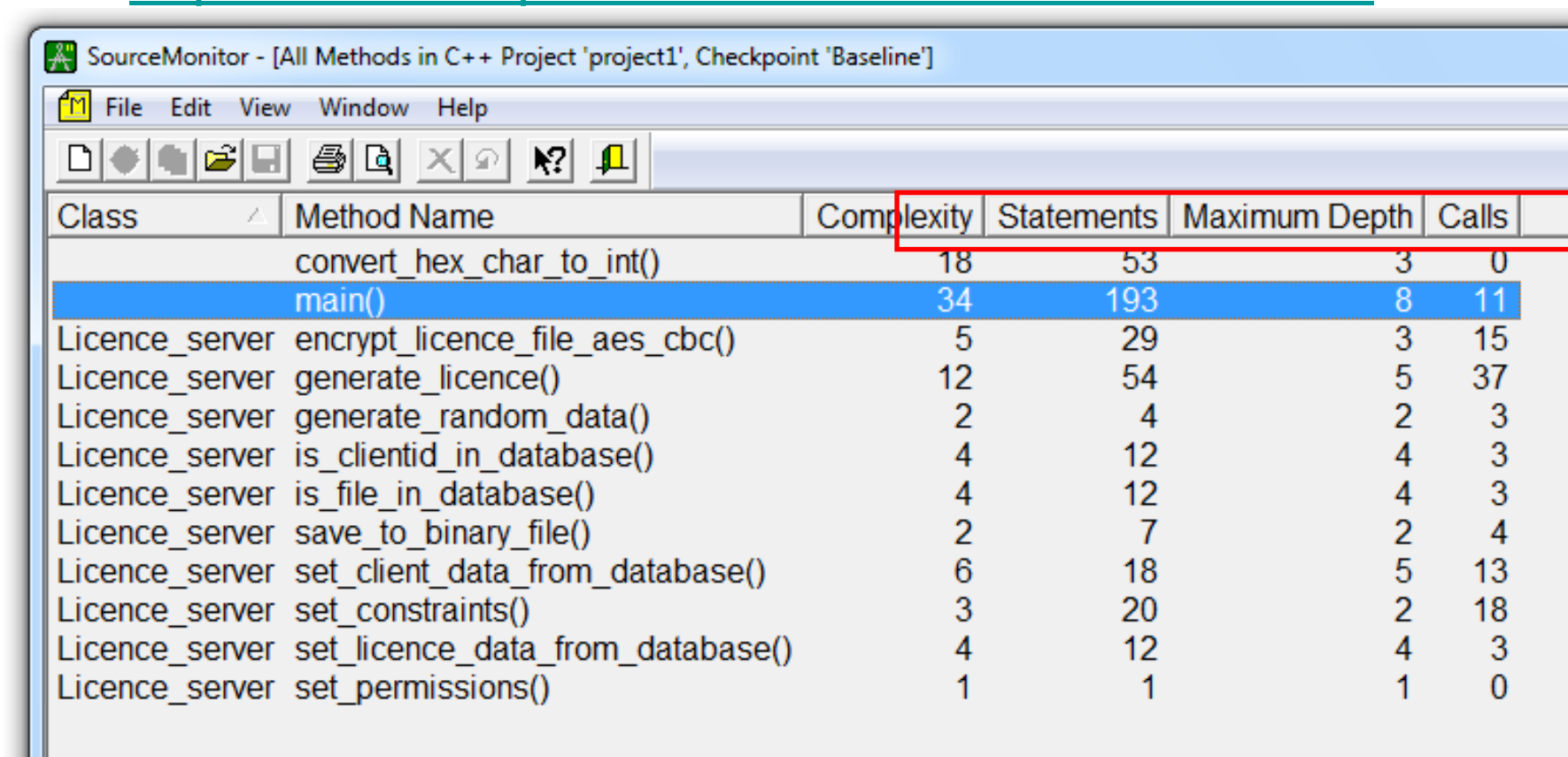
- Často se vyskytující problémy v psaní kódu
- <http://sourcemaking.com/antipatterns>
- Špagetový kód
- Cut&Paste programování
- Velká provázanost
  - každá změna způsobí problém a nutnost dalších změn

# Refactoring

- Úprava kódu z důvodu zlepšení jeho čitelnosti a flexibility
  - např. rozdělení funkcí do více podfunkcí
  - přesun nebo sjednocení souvisejícího kódu
  - čitelnější zápis logických podmínek
  - úprava argumentů funkcí, jejich pojmenování (API)
  - ...
- Nedochází k přidání nové funkčnosti
  - ale může dojít k přidání nových chyb ☺
- <http://www.sourcemaking.com/refactor>
- Většinou manuální práce, ale nástroje mohou mít podporu pro některé pomocné operace
  - např. přejmenování proměnné v celém projektu
  - např. identifikace problematických míst v kódu

# Source monitor – example outputs

- <http://www.campwoodsw.com/sourcemonitor.html>



The screenshot shows the SourceMonitor application window titled "SourceMonitor - [All Methods in C++ Project 'project1', Checkpoint 'Baseline']". The window has a menu bar (File, Edit, View, Window, Help) and a toolbar. Below the toolbar is a table with the following columns: Class, Method Name, Complexity, Statements, Maximum Depth, and Calls. The table lists several methods, with 'main()' highlighted in blue. A red rectangle highlights the 'Complexity', 'Statements', 'Maximum Depth', and 'Calls' columns.

Class	Method Name	Complexity	Statements	Maximum Depth	Calls
	convert_hex_char_to_int()	18	53	3	0
	main()	34	193	8	11
Licence_server	encrypt_licence_file_aes_cbc()	5	29	3	15
Licence_server	generate_licence()	12	54	5	37
Licence_server	generate_random_data()	2	4	2	3
Licence_server	is_clientid_in_database()	4	12	4	3
Licence_server	is_file_in_database()	4	12	4	3
Licence_server	save_to_binary_file()	2	7	2	4
Licence_server	set_client_data_from_database()	6	18	5	13
Licence_server	set_constraints()	3	20	2	18
Licence_server	set_licence_data_from_database()	4	12	4	3
Licence_server	set_permissions()	1	1	1	0

- Complexity: 1-10(OK), 11-20(někdy), > 20(NOK)

# Open-source portály, verzovací nástroje, reverzní inženýrství



## Další verzovací nástroje

- [http://en.wikipedia.org/wiki/Revision\\_control](http://en.wikipedia.org/wiki/Revision_control)
- SVN, GIT, Mercurial, Bazaar...
- [http://en.wikipedia.org/wiki/Comparison\\_of\\_revision\\_control\\_software](http://en.wikipedia.org/wiki/Comparison_of_revision_control_software)



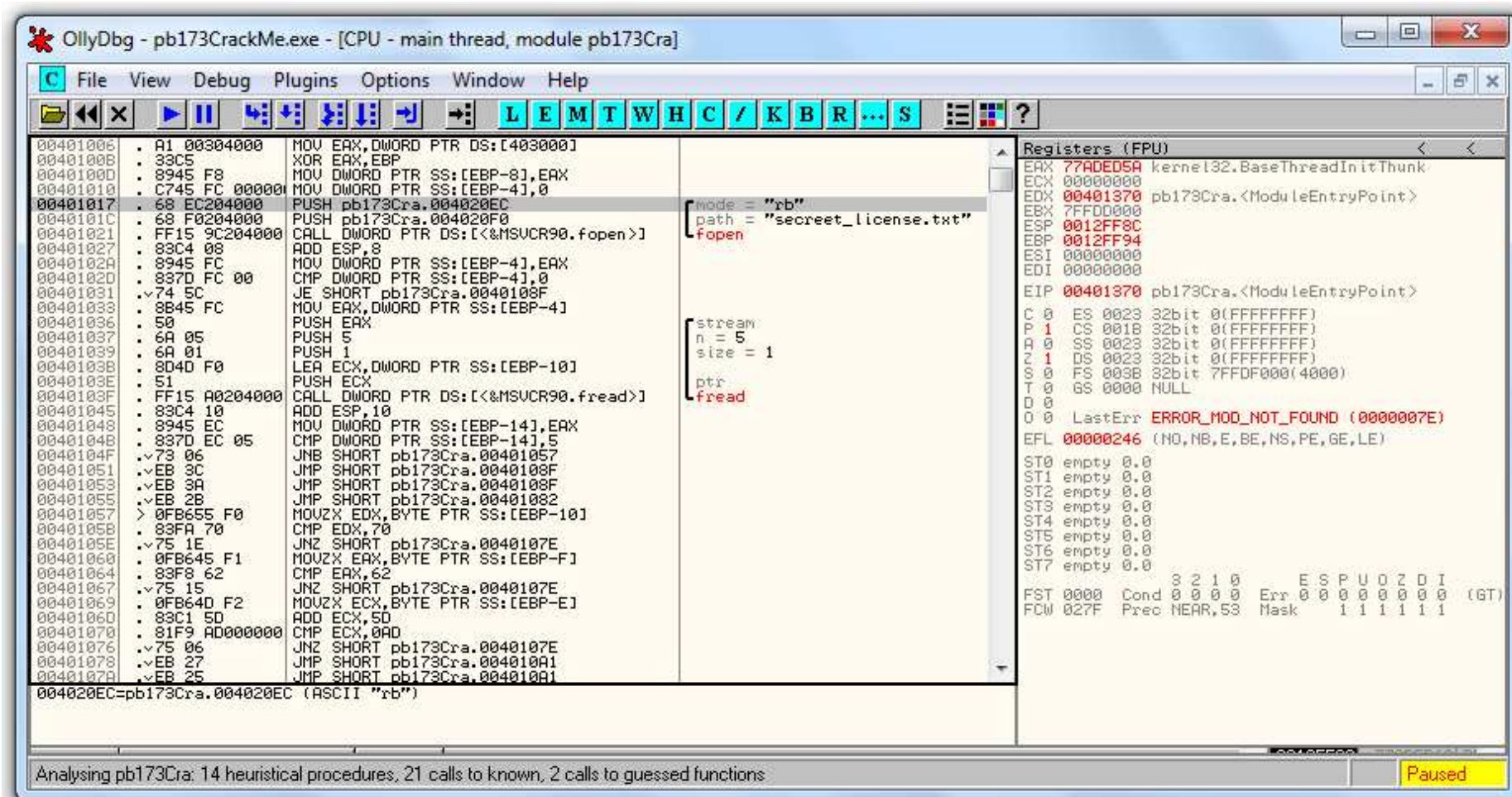
# Open-source portály

- Open-source portály
  - <https://sourceforge.net>
  - <https://code.google.com/hosting/>
  - <https://github.com/>
  - <http://www.codeplex.com/>
  - ...
- Zapojte se do existujícího projektu
  - TODO list, bugs
- Založte vlastní projekt
  - bakalářka, vlastní nápad...
  - (dobrá reference při pohovoru do firmy)

# Assembler, Reverzní inženýrství RE

- Schopnost (částečné) práce na úrovni assembleru zvětšuje pochopení programu a možnost ladění problémů
- Podpora v IDE (Disassembly režim) během debuggingu
  - Visual Studio → Go to Disassembly
  - QT Creator → Debug → Operate by instructions
- Specializované nástroje (OllyDbg, IDA...)
- RE: získání původního kódu z přeložené binárky
  - není ale omezeno jen na software
    - [http://en.wikipedia.org/wiki/Reverse\\_engineering](http://en.wikipedia.org/wiki/Reverse_engineering)
  - The Reverse Code Engineering Community:
    - <http://www.reverse-engineering.net/>
  - Tutoriály: <http://www.tuts4you.com>

# OllyDbg <http://www.ollydbg.de/>



# Knihovna SDL

- Simple Direct Media Layer <http://www.libsdl.org>
- Pokročilá knihovna pro práci s grafikou
- Více možností než Allegro



# NCurses, PDCurses

- NCurses 5.9: <http://www.gnu.org/s/ncurses/>
  - Unix/Linux
- PDCurses: <http://pdcurses.sourceforge.net/>
  - port pro Windows, PDCurses.dll
- Knihovna pro práci s „grafikou“ v textové konzoly
  - <http://www.paulgriffiths.net/program/c/curses.php>

# NCurses

The image shows a terminal window with two main applications running side-by-side. On the left is a ncurses-based naval battle game titled 'Batalla Naval ncurses client v0.60 by Ricardo Quesada (c)1996,97,98'. The game interface includes a 'Your board' and an 'Enemy board', both represented as 10x10 grids. A menu on the right side of the game window offers options: disconnect, Send board, status, send Msg, Config, abOut, and Quit. At the bottom of the game window, it says 'Batalla Naval ncurses client v0.60. by Ricardo Quesada. Use TAB, CURSOR LEFT & CURSOR RIGHT to change. Connected to bserver:homero.pjn.gov.ar. You bserver version:0.59' and shows 'Player 1' connected.

On the right is an htop system monitor window. It displays system statistics: Tasks: 75, 32 thr; 1 running; Load average: 0.01 0.07 0.08; Uptime: 1 day, 13:27:36. It also shows memory usage: 630/3819MB and swap usage: 0/0MB. Below this is a table of running processes:

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	Command
17721	hisham	40	0	3976	2032	1596	S	0.0	0.1	ssh -p 2222 hisham
17720	root	40	0	7452	4664	2348	S	0.0	0.1	python ./main.py p
2675	hisham	40	0	49836	27860	16656	S	0.0	0.7	pidgin
1232	root	40	0	1860	340	228	S	0.0	0.0	/bin/dhpcd wlan0 -h par
1197	root	40	0	3776	900	656	S	0.0	0.0	wpa_supplicant -B -i wla
1193	hisham	40	0	30092	16176	9568	S	0.0	0.4	/usr/bin/python -O /usr/
1187	hisham	40	0	20584	5716	4252	S	0.0	0.1	xfce4-settings-helper
1183	hisham	40	0	21156	8820	6868	S	0.0	0.2	xfdesktop
1181	hisham	40	0	19684	6240	5120	S	0.0	0.2	Thunar --daemon
1179	hisham	40	0	31796	12464	8700	S	0.0	0.3	xfce4-panel
1608	hisham	40	0	31796	12464	8700	S	0.0	0.3	xfce4-panel
1207	hisham	40	0	33892	12024	8868	S	0.0	0.3	/System/Index/lib/xfc
1214	hisham	40	0	33892	12024	8868	S	0.0	0.3	/System/Index/lib/
1206	hisham	40	0	0	0	0	Z	0.0	0.0	xfce4-battery-p
1204	hisham	40	0	23420	12060	8296	S	0.0	0.3	/System/Index/libexec

At the bottom of the htop window, there is a legend for function keys: F1Help, F2Setup, F3Search, F4Filter, F5Tree, F6SortBy, F7Nice, F8Nice +F9Kill, F10Quit.

# NCurses - demo

```
/* Prelozit: */
/* module add ncurses */
/* gcc testcurses.c -o testcurses -lncurses */
/* Pred spustenim nastavit: export TERM=xterm */
#include <ncurses.h>
#include <unistd.h>

int main() {
    int i,ch;
    char text[]="Stiskni nejakou klavesu ";
    char usr[10],pwd[10];
    WINDOW *ww;
    /* inicializace */
    initscr(); /* Zacatek prace s curses */
    cbreak(); noecho(); keypad(stdscr,1); /* Nastav implicitní režimy */
    nodelay(stdscr,0); nl();
    clear(); /* Vymaz obrazovku */
    /* pis po obrazovce - uhlopricne */
    mvaddstr(0,20,"Zkouska psani po obrazovce:"); /* pis text na zadanou pozici */
    getmaxyx(stdscr,maxr,maxc);
    move(maxr-1,0); /* Presun kurzor */
    printw("Okno ma %d radku, %d sloupcu",maxr,maxc); /* od pozice kurzoru pis text */
    for(i=0;i<23;i++) { mvaddch(i+1,3*i,text[i]); /* od pozice kurzoru pis znak */
        napms(300); /* Cekej zadany pocet milisekund */
        refresh(); /* Teprve ted se zmeny vykresli! */
    }
}
```



```

/* zkouska cteni z klavesnice */
ch=getch(); /* Cti znak z klavesnice */
attrset(A_STANDOUT); /* Dale pis zvyraznene */
mvprintw(2,30,"%s %c","Stiskl jsi: ",ch); /* Jako printf, ale do okna curses na danou pozici */
refresh();
sleep(5u); /* Cekej 5 sekund */
clear();
refresh();
/* zkouska okna */
ww=newwin(5,20,10,30); /* Vytvor podokno */
wborder(ww,'|','|','-','=','!','!',+',','*'); /* Oramuj okno */
echo(); /* Vstup vypisuj */
mvwaddstr(ww,1,2,"Login: "); /* Pis do podokna */
wgetstr(ww,usr); /* Cti retez v okne */
noecho(); /* Vstup nevypisuj */
wmove(ww,3,2);waddstr(ww,"Password: ");
wgetstr(ww,pwd);
wrefresh(ww); /* Prekresli jen podokno */
delwin(ww); /* Zrus podokno */
attrset(A_REVERSE); /* Nadale prohod barvu popredi a pozadi (negativ) */
mvprintw(20,10,"Vsichni sem! "
    "Uzivatek \"%s\" ma heslo \"%s\"",usr,pwd);
refresh();
sleep(5u);
clear();

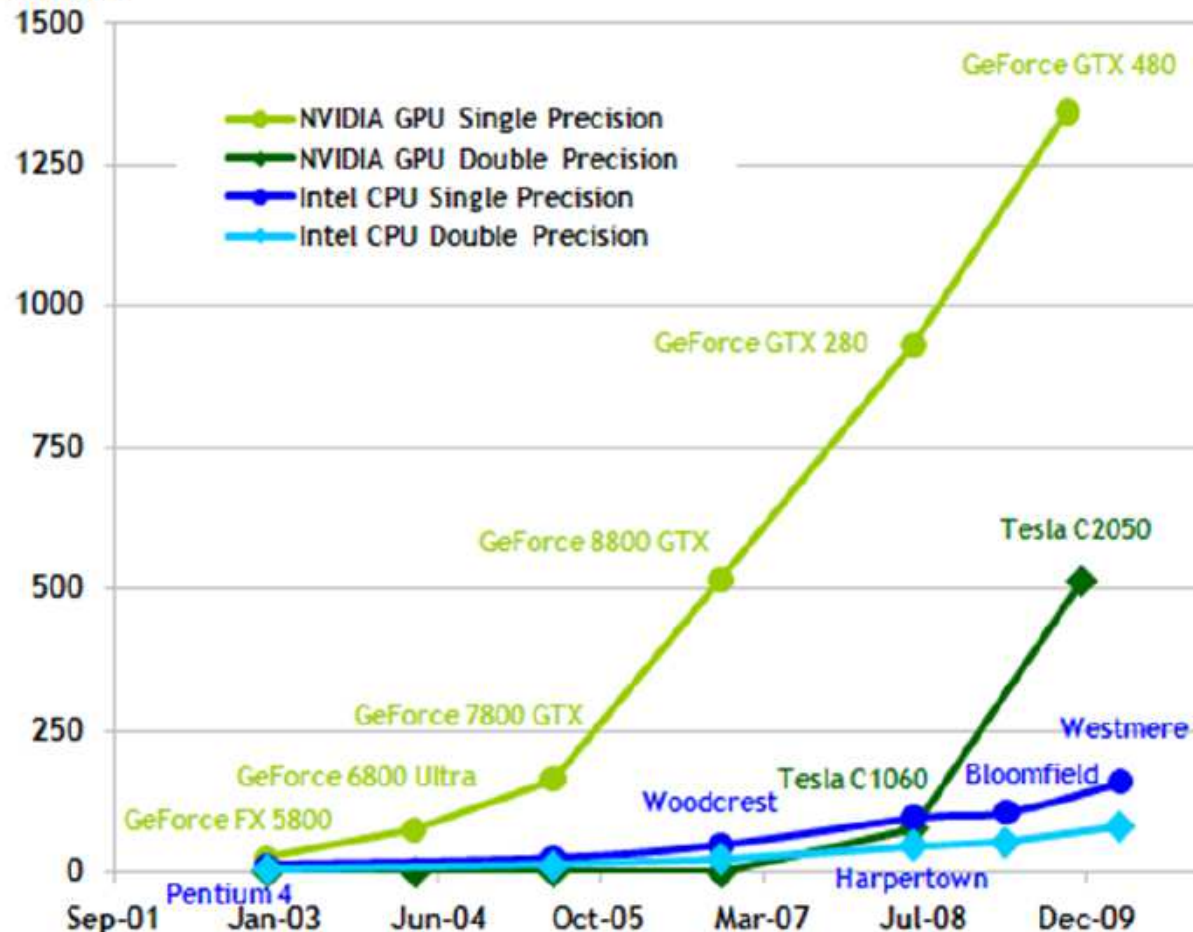
```

```
/* zkouska barvy */
if(has_colors()) { /* Umi terminal zpracovavat barvy? */
    start_color(); /* Pracuj s barvami */
    init_pair(1, COLOR_RED, COLOR_YELLOW); /* Definuj dvojici barev (pozadi, popredi) */
    attron(COLOR_PAIR(1)); /* Pouzij definovanou dvojici barev */
    mvprintw(4,30,"BAREVNY TEXT");
    attroff(COLOR_PAIR(1)); /* Prestan pouzivat dvojici barev */
    refresh();
    i=getch(); /* Cekej na zadani znaku */
}
endwin(); /* Konec prace s ncurses */
return 0;
}
```

# CUDA – výpočty na grafických kartách

- Masivně paralelní programování na kartách nVidia
  - stovky jader, tisíce vláken na jedné GPU
  - máte pravděpodobně doma!
- Rozšíření jazyka C pro paralelní výpočty
  - obohaceno o konstrukce pro paralelní spouštění výpočtů
  - vývojové nástroje dostupné zdarma
- CUDA toolkit
  - <http://developer.nvidia.com/cuda-toolkit-40>
- CUDA programming guide
  - [http://developer.download.nvidia.com/compute/cuda/3\\_0/toolkit/docs/NVIDIA\\_CUDA\\_ProgrammingGuide.pdf](http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf)
- CUDA seminář na Standfordu
  - <http://itunes.apple.com/itunes-u/programming-massively-parallel/id384233322#ls=1>

Theoretical  
GFLOP/s



# CUDA - ukázka

unikátní identifikace vlákna  
(přiřazeno automaticky)

- Paralelní sečtení vektoru po složkách

```
__global__ void VecAdd(float* A, float* B, float* C) {  
    int i = threadIdx.x;  
    C[i] = A[i] + B[i];  
}  
int main() {  
    // Invocation with N threads  
    VecAdd<<<1, N>>>(A, B, C);  
}
```

sečtení dvou prvků vektoru

funkce VecAdd spuštěna na  
N vláknech

- [http://developer.download.nvidia.com/compute/cuda/3\\_0/toolkit/docs/NVIDIA\\_CUDA\\_ProgrammingGuide.pdf](http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf)

# Cppcheck



- A tool for static C/C++ code analysis
  - Open-source freeware, <http://cppcheck.sourceforge.net/>
- Last version 1.61 (2013-08-03)
- Used to find bugs in open-source projects (Linux kernel... )
- Command line & GUI version
- Standalone version, plugin into IDEs, version control...
  - Code::Blocks, Codelite, Eclipse, Jenkins...
  - Tortoise SVN
  - not Visual Studio ☹
- Cross platform (Windows, Linux)
  - `sudo apt-get install cppcheck`

# Cppcheck – what is checked?

- Bound checking for array overruns
- Suspicious patterns for class
- Exceptions safety
- Memory leaks
- Obsolete functions
- sizeof() related problems
- String format problems...
- See full list

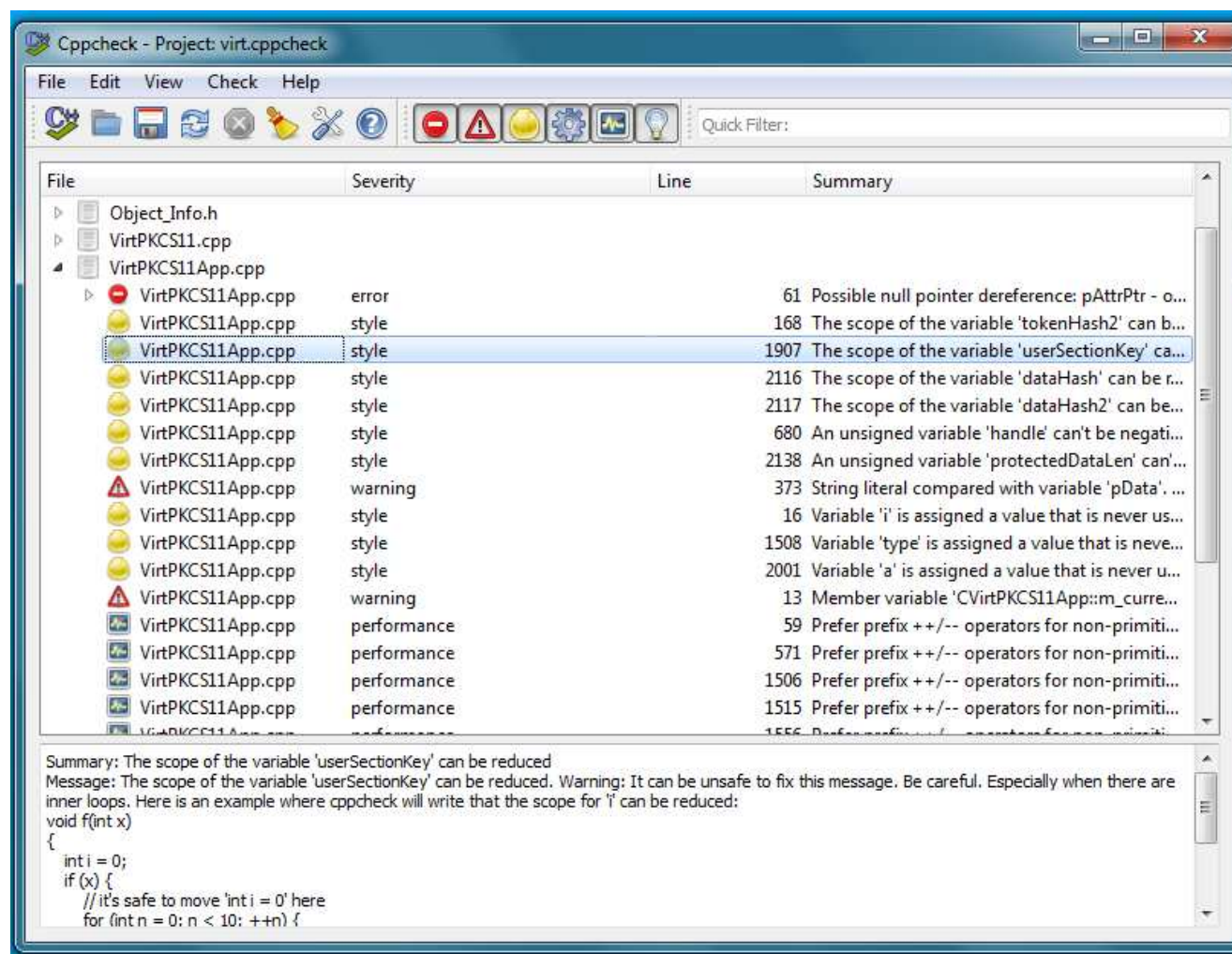
[http://sourceforge.net/apps/mediawiki/cppcheck/index.php?title=Main\\_Page#Checks](http://sourceforge.net/apps/mediawiki/cppcheck/index.php?title=Main_Page#Checks)

# Cppcheck – categories of problems

- **error** – when bugs are found
- **warning** - suggestions about defensive programming to prevent bugs
- **style** - stylistic issues related to code cleanup (unused functions, redundant code, constness...)
- **performance** - suggestions for making the code faster.
- **portability** - portability warnings. 64-bit portability. code might work different on different compilers. etc.
- **information** - Informational messages about checking problems



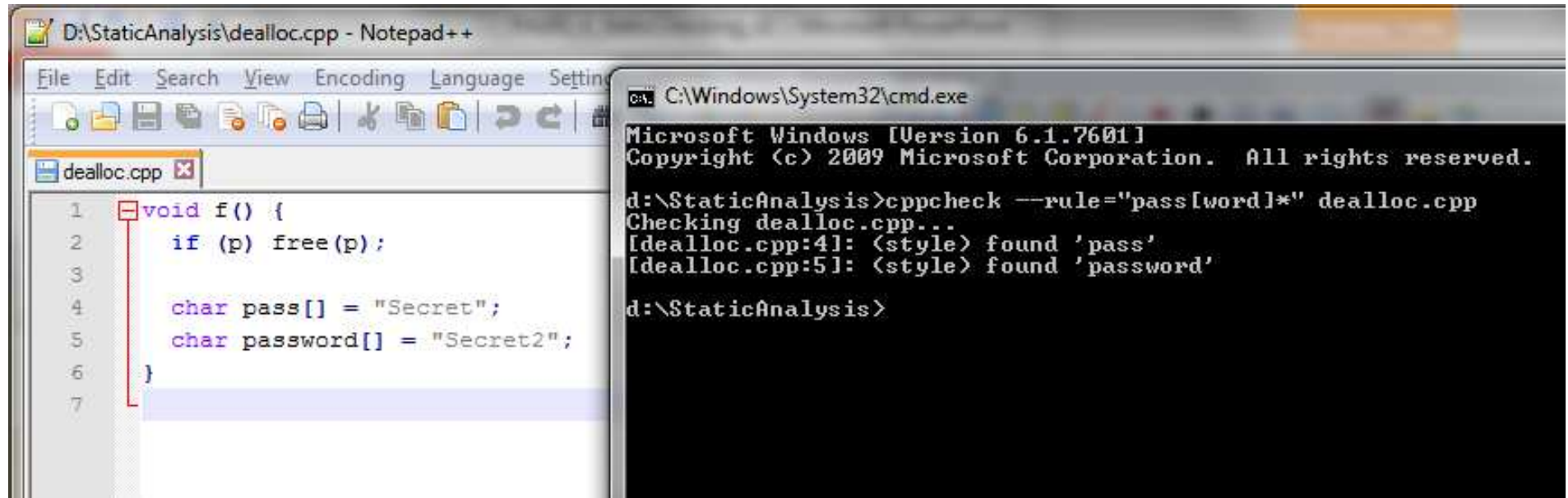
# Cppcheck



# Cppcheck – simple custom rules

- User can write own regular expression-based rules
  - Perl Compatible Regular Expressions [www.pcre.org](http://www.pcre.org)
  - limited only to simpler analysis
  - executed over simplified code (code after preprocessing)
    - <http://sourceforge.net/projects/cppcheck/files/Articles/writing-rules-2.pdf>
- Regular expression can be supplied on command line
  - `cppcheck.exe --rule="." file.cpp`
    - match any code, use to obtain simplified code
  - `cppcheck.exe --rule="pass[word]*" file.cpp`
    - match any occurrence of pass or password or passwordword...
- Or via XML file (for stable repeatedly used rules)

`cppcheck.exe --rule="pass[word]*" file.cpp`



- `cppcheck.exe --rule="if \( p \) { free \( p \) ; }" file.cpp`
  - will match only pointer with name 'p'

# ISO/IEC 9899:2011 (C11)

# ISO/IEC 9899:2011 (C11)

- Nejnovější verze standardu (2011)
  - [http://en.wikipedia.org/wiki/C11\\_\(C\\_standard\\_revision\)](http://en.wikipedia.org/wiki/C11_(C_standard_revision))
  - přidány drobné rozšíření jazyka
  - přidány některé funkce dříve dostupné jen v POSIXu
- Pěkný souhrn motivací a změn
  - <http://www.jauu.net/data/pdf/c1x.pdf>
- Vyzkoušení na Aise:
  - module add gcc-4.7.2
  - gcc -std=c11
  - GCC zatím nepodporuje všechny nové vlastnosti
- (Zatím nejrozšířenější zůstává použití C99)

# Vlákná

- `#include <threads.h>`
- Velmi podobné vláknům v POSIXu (snadný přechod)
  - `pthread_create` -> `thr_create`
  - `pthread_mutex_init` -> `mtx_init`
- Specifikace lokální proměnné ve vlákně `_Thread_local`
  - lokální proměnná ve funkci spuštěné paralelně v několika vláknech
- `_Thread_local` storage-class
- `_Atomic` type qualifier, `<stdatomic.h>`
- Metody pro synchronizaci
- Atomické operace `_Atomic int foo;`

# Atomičnost operací a paměti

- Je `i++` atomické?
  - není, je nutné načíst, zvětšit, uložit
  - u vícevláknového programu může dojít k prolnutí těchto operací
    - načte se hodnota, která ale již nebude po zvětšení aktuální – jiné vlákno uložilo zvětšenou hodnotu `i`
- `atomic_{load,store,exchange}`
- `atomic_fetch_{add,sub,or,xor,and}`
- `atomic_compare_exchange_`

# Exkluzivní otevření souboru - motivace

- Např. MS Word při editaci souboru soubor.doc vytváří ~\$soubor.doc
  - při pokusu o otevření souboru soubor.doc vždy kontroluje, zda se mu podaří vytvořit a otevřít ~\$soubor.doc
  - pokud ne, soubor soubor.doc je již editován
  - ~\$soubor.doc je otevírán pomocí
- Po ukončení programu se zámek ruší
  - korektní ukončení většinou smaže i soubor se zámkem
  - náhlé ukončení ponechá soubor, ale již neblokuje přístup



# Exkluzivní režim otevření souboru

- Jak zjistíme, že soubor (zámku) již existuje?
  - otevři pro čtení
  - když selže, tak vytvoř a otevři pro zápis
  - race condition mezi čtením a vytvořením
  - potřebovali bychom “selži pokud existuje, jinak otevři na zápis”
- Dodatečný režim otevření souboru `fopen("cesta", "wx")`
  - vytvoř a otevři exkluzivně
  - selže pokud již existuje a někdo jej drží otevřený
- Typické využití pro soubory signalizující zámek (lock files)
  - pokud je aplikace spuštěna vícekrát, tak detekuje soubory, které jsou již používány
- Ekvivalentní POSIX příkazu `open(O_CREAT | O_EXCL)`

# Typově proměnná makra

- Vyhodnocení makra v závislosti na typu proměnné (Type-generic expressions)
- klíčové slovo **\_Generic**

```
#define FOO(X) myfoo(X)

#define FOO(X)
    _Generic((X), long: fool, char: fooc, default foo) (X)
```

# Vylepšená podpora Unicode (UTF-16/32)

- `char16_t` and `char32_t`
- `<uchar.h>`

# Bezpečné varianty některých funkcí

- Funkce z (Secure C Library)
  - [http://docwiki.embarcadero.com/RADStudio/XE3/en/Secure\\_C\\_Library](http://docwiki.embarcadero.com/RADStudio/XE3/en/Secure_C_Library)
  - <http://msdn.microsoft.com/en-us/library/8ef0s5kh%28v=vs.80%29.aspx>
  - <http://www.drdobbs.com/cpp/the-new-c-standard-explored/232901670>
- fopen\_s, fprintf\_s, strcpy\_s, strcat\_s, gets\_s...
  - typicky kontrola délky paměti na ochranu před zápisem za konec alokované paměti (buffer overflow)
- gets() deprecated v C99, nyní úplně odstraněna

# Makra pro zjištění možností prostředí

- `__STDC_VERSION__`
  - makro pro zjištění verze, 201112L je C11

# Anonymní struct a union

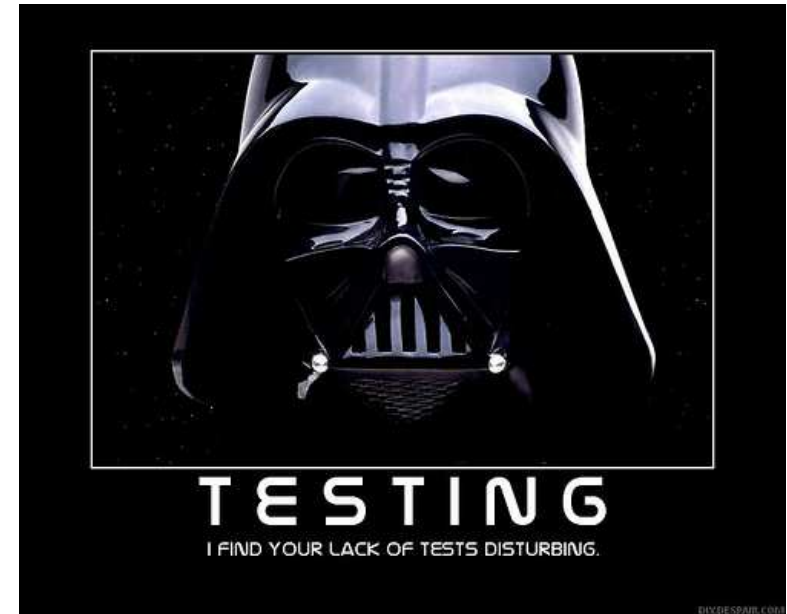
- Struktury a unie bez pojmenování
- Využití pro vnořené deklarace struct nebo union
  - struktura má atribut, který je typu struct / union
  - nikde jinde není použit / potřeba
  - není nutné pojmenovávat

# Funkce s rychlým ukončením

- `_Noreturn`

- specifikace pro překladač, z funkce se nevrátíme (abort, exit...)
- umožňuje lepší optimalizaci překladačem

# Testování, unit testing





# Typy testování

- Manuální vs. Automatické
- Dle rozsahu testovaného kódu
- Unit testing
  - testování elementárních komponent
  - (jednotlivé funkce, třídy)
- Integrační testy
  - test spolupráce několika komponent mezi sebou
  - typicky dodržení definovaného rozhraní
- Systémové testy
  - test celého programu v reálném prostředí
  - ověření chování vůči specifikaci

# Psaní unit testů

- Automatizovaně spouštěné kusy kódu
- Zaměření na testování elementárních komponent
  - obsah proměnných (např. je konstanta `DAYSINWEEK==7?`)
  - chování funkce (např. sčítá funkce korektně?)
  - konzistence struktur (např. obsahuje seznam první prvek?)
- Základní testování opakuje následující kroky:
  1. V testu provedeme elementární komponentu
    - např. spuštění funkce `add`
  2. Obdobně jako pro `assert()` otestujeme výsledek
    - `vhodny_assert (add (-1, 2) == 1) ;`
  3. Pokud není podmínka splněna, vypíšeme hlášení

# MinUnit

- <http://www.jera.com/techinfo/jtns/jtn002.html>
- Extrémně jednoduchý testovací „framework“ pro C/C++
  - lze pustit v libovolném prostředí

vypiš message pokud !(test)

```
/* file: minunit.h */  
#define mu_assert(message, test) do { if (!(test)) return message; } while (0)  
#define mu_run_test(testFnc) do { char *message = testFnc(); tests_run++; \  
    if (message) return message; } while (0)  
extern int tests_run;
```

spust' testFnc() a vrať výsledek

- Pozn. *do{...} while(0)* s testem nesouvisí
  - jde o způsob, jak psát bezpečně makro obsahující více příkazů
  - <http://stackoverflow.com/questions/1067226/c-multi-line-macro-do-while0-vs-scope-block>

# MinUnit – definice jednotlivých testů

```
/* file minunit_example.c */
#include <stdio.h>
#include "minunit.h"

int tests_run = 0;

int foo = 7;
int bar = 4;

static char * test_foo() {
    mu_assert("error, foo != 7", foo == 7);
    return 0;
}

static char * test_bar() {
    mu_assert("error, bar != 5", bar == 5);
    return 0;
}
```

naše proměnné,  
jejichž hodnoty  
budeme testovat

test zda proměnná foo  
je rovna 7 (ok)

test zda proměnná bar  
je rovna 5 (selže)

# MinUnit – spuštění a vyhodnocení testů

```
static char * all_tests() {
    mu_run_test(test_foo);
    mu_run_test(test_bar);
    return 0;
}

int main(int argc, char **argv) {
    char *result = all_tests();
    if (result != 0) {
        printf("%s\n", result);
    }
    else {
        printf("ALL TESTS PASSED\n");
    }
    printf("Tests run: %d\n", tests_run);

    return result != 0;
}
```

**spuštění jednotlivých testů**  
(pozn. zastaví se na prvním chybném)

**výpis v případě nefunkčního testu**

**lze získat celkový počet testů, které proběhly korektně**

## Unit testy – další informace

- Unit testy poskytují robustní specifikaci očekávaného chování komponent
- Unit testy nejsou primárně zaměřené na hledání nových chyb v existujícím kódu
  - většina chyb se projeví až při kombinaci komponent
  - typicky pokryto integračním testováním
- Regresní testy jsou typicky integrační testy
  - testy pro detekci výskytu dříve odhalené chyby
- Klíčové pro provádění refactoringu
  - porušení unit testu je rychle odhaleno

# Unit testy – další informace

- Dělejte testy navzájem nezávislé
- Testujte jedním testem jen jednu komponentu
  - změna komponenty způsobí změnu jediného testu
- Pojmenujte testy vypovídajícím způsobem
  - co (komponenta), kdy (scénář použití), výsledek (očekávaný)
- I další typy testů lze dělat se stejným „frameworkem“
  - rozlišujte ale jasně unit testy od integračních
- Integrační testy vykonají související část kódu
  - např. vložení několika prvků do seznamu a test obsahu

# CxxTest – pokročilejší framework

- <http://cxxtest.tigris.org/>
- Pro C i C++
  - vyžaduje překladač pro C++ a Python
  - testy jsou funkce v potomkovi CxxTest::TestSuite
- Lze integrovat do IDE
  - např. VisualStudio: <http://morison.biz/technotes/articles/23>
- Existuje velké množství dalších možností
  - [http://en.wikipedia.org/wiki/List\\_of\\_unit\\_testing\\_frameworks](http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks)



# CxxTest – dostupné testovací makra

<http://cxxtest.sourceforge.net/guide.html#TOC7>

## Macro

[TS\\_FAIL\(\*message\*\)](#)

[TS\\_ASSERT\(\*expr\*\)](#)

[TS\\_ASSERT\\_EQUALS\(\*x\*, \*y\*\)](#)

[TS\\_ASSERT\\_SAME\\_DATA\(\*x\*, \*y\*, \*size\*\)](#)

[TS\\_ASSERT\\_DELTA\(\*x\*, \*y\*, \*d\*\)](#)

[TS\\_ASSERT\\_DIFFERS\(\*x\*, \*y\*\)](#)

[TS\\_ASSERT\\_LESS\\_THAN\(\*x\*, \*y\*\)](#)

[TS\\_ASSERT\\_LESS\\_THAN\\_EQUALS\(\*x\*, \*y\*\)](#)

[TS\\_ASSERT\\_PREDICATE\(\*R\*, \*x\*\)](#)

[TS\\_ASSERT\\_RELATION\(\*R\*, \*x\*, \*y\*\)](#)

[TS\\_ASSERT\\_THROWS\(\*expr\*, \*type\*\)](#)

[TS\\_ASSERT\\_THROWS\\_EQUALS\(\*expr\*, \*arg\*, \*x\*, \*y\*\)](#)

[TS\\_ASSERT\\_THROWS\\_ASSERT\(\*expr\*, \*arg\*, \*assertion\*\)](#)

[TS\\_ASSERT\\_THROWS\\_ANYTHING\(\*expr\*\)](#)

[TS\\_ASSERT\\_THROWS\\_NOTHING\(\*expr\*\)](#)

[TS\\_WARN\(\*message\*\)](#)

[TS\\_TRACE\(\*message\*\)](#)

## Description

Fail unconditionally

Verify (*expr*) is true

Verify (*x*==*y*)

Verify two buffers are equal

Verify (*x*==*y*) up to *d*

Verify !(*x*==*y*)

Verify (*x*<*y*)

Verify (*x*<=*y*)

Verify *P*(*x*)

Verify *x R y*

Verify that (*expr*) throws a specific type of exception

Verify type and value of what (*expr*) throws

Verify type and value of what (*expr*) throws

Verify that (*expr*) throws an exception

Verify that (*expr*) doesn't throw anything

Print *message* as a warning

Print *message* as an informational message

# Bezpečnostní dopady práce s pamětí a nedostatečného ošetření vstupu

# Demo – buffer overflow u fixního pole

# Předpoklady

- MS Visual Studio 2010 (MSVC kompilátor)
  - dostupné z fakultní MSAA
- V kódu alternativní nastavení pro gcc
  - v ukázkách nepoužito
  - jiné rozložení proměnných v paměti než u MSVC
- Debug režim

```

void demoBufferOverflowData() {
    int          unused_variable = 30;
    #define NORMAL_USER      'n'
    #define ADMIN_USER      'a'
    int          userRights = NORMAL_USER;
    #define USER_INPUT_MAX_LENGTH 8
    char         userName[USER_INPUT_MAX_LENGTH];
    char         passwd[USER_INPUT_MAX_LENGTH];

    // print some info about variables
    printf("%-20s: %p\n", "userName", userName);
    printf("%-20s: %p\n", "passwd", passwd);
    printf("%-20s: %p\n", "unused_variable", &unused_variable);
    printf("%-20s: %p\n", "userRights", &userRights);
    printf("\n");

    // Get user name
    printf("login as: ");
    gets(userName);

    // Get password
    printf("%s@vulnerable.machine.com: ", userName);
    gets(passwd);

    // Check user rights (set to NORMAL_USER and not changed in code)
    if (userRights == NORMAL_USER) {
        printf("\nWelcome, normal user '%s', your rights are limited.\n\n", userName);
    }
    if (userRights == ADMIN_USER) {
        printf("\nWelcome, all mighty admin user '%s'!\n", userName);
    }
}

```

**proměnná udávající  
práva aktuálně  
přihlášeného uživatele**

**pole s fixní délkou (bude  
docházet k zápisu za  
konec)**

**pomocný výpis adres  
lokálních proměnných  
na zásobníku**

**načtení uživatelského  
jména a hesla (bez  
kontroly délky)**

**výpis info uživatele dle  
proměnné s právem**

# Rozložení dat v paměti

```
void demoBufferOverflowData() {
    int unused_variable = 30;
#define NORMAL_USER 'n'
#define ADMIN_USER 'a'
    int userRights = NORMAL_USER;
#define USER_INPUT_MAX_LENGTH 8
    char userName[USER_INPUT_MAX_LENGTH];
    char passwd[USER_INPUT_MAX_LENGTH];

    // print some info about variables
    printf("%-20s: %p\n", "userName", userName);
    printf("%-20s: %p\n", "passwd", passwd);
    printf("%-20s: %p\n", "unused_variable", &unused_variable);
    printf("%-20s: %p\n", "userRights", &userRights);
    printf("\n");

    // Get user name
    memset(userName, 1, USER_INPUT_MAX_LENGTH);
    memset(passwd, 2, USER_INPUT_MAX_LENGTH);
    printf("login as: ");
    fflush(stdout);
}
```

Memory1

Address: 0x0024FB18 {#} Columns: Auto

0x0024FB1B	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x0024FB2C	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x0024FB3D	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x0024FB4E	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x0024FB5F	cc	02	02	02	02	02	02	02	02	02	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x0024FB70	01	01	01	01	01	01	01	01	01	cc	cc	cc	cc	cc	cc	cc	cc	cc	6e
0x0024FB81	00	00	00	cc	cc	cc	cc	cc	cc	cc	cc	1e	00	00	00	cc	cc	cc	cc
0x0024FB92	cc	cc	85	20	45	b8	6c	fc	24	00	9a	20	dd	00	00	00	00	00	00
0x0024FBA3	00	00	00	00	00	00	e0	fd	7f	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x0024FBB4	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x0024FBC5	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x0024FBD6	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x0024FBE7	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x0024FBF8	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x0024FC09	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc

Autos Locals Memory1 Threads Modules Watch 1

unused\_variable

# Spuštění bez problémů

The screenshot shows a debugger window with the following components:

- Global Scope:** Displays the C source code. The code prompts for a username and password, and checks user rights. The user enters 'petr' and 'test', and the program outputs a welcome message for the user 'petr'.
- Memory 1:** A window showing memory addresses and their contents. The address 0x0013FA03 is selected. The contents show the memory layout of the program, including the user input buffer.
- Output Window:** Displays the output of the program. The output shows the program's output, including the welcome message for the user 'petr'.

The program's output is as follows:

```
login as: petr
petr@vulnerable.machine.com: test
Welcome, normal user 'petr', your rights are limited.
```

userName

passwd



# Spuštění útočníkem – userName

zadáno 'evil' do userName

```
// Get user name
memset(userName, 1, USER_INPUT_MAX_LENGTH);
memset(passwd, 2, USER_INPUT_MAX_LENGTH);
printf("login as: ");
fflush(stdout);
gets(userName);

// Get password
printf("%s@vulnerable.machine.com: ", userNam
fflush(stdout);
gets(passwd);

// Check user rights (set to NORMAL_USER and
if (userRights == NORMAL_USER) {
    printf("\nWelcome, normal user '%s', your
    fflush(stdout);
}
if (userRights == ADMIN_USER) {
    printf("\nWelcome, all mighty admin user
```

Memory1		
Address:	0x0024FB1B	Columns: Auto
0x0024FB1B	cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc	iiiiiiiiiiiiiiii
0x0024FB2C	cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc	iiiiiiiiiiiiiiii
0x0024FB3D	cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc	iiiiiiiiiiiiiiii
0x0024FB4E	cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc	iiiiiiiiiiiiiiii
0x0024FB5F	c 02 02 02 02 02 02 02 02 cc cc cc cc cc cc cc cc	i.....iiiiiiii
0x0024FB70	65 76 69 6c 00 01 01 01 cc cc cc cc cc cc cc cc cc	evil...iiiiiiii
0x0024FB81	00 00 00 cc cc cc cc cc cc cc cc cc cc cc cc cc cc	...iiiiiiii...ii
0x0024FB92	cc cc 85 20 45 b8 6c fc 24 00 9a 20 dd 00 00 00 00	ii. E,lu\$.š Ý....
0x0024FBA3	00 00 00 00 00 00 e0 fd 7f cc cc cc cc cc cc cc cc	.....áy.iiiiiiii
0x0024FBB4	cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc	iiiiiiiiiiiiiiii
0x0024FBC5	cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc	iiiiiiiiiiiiiiii
0x0024FBD6	cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc	iiiiiiiiiiiiiiii
0x0024FBE7	cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc	iiiiiiiiiiiiiiii
0x0024FBF8	cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc	iiiiiiiiiiiiiiii
0x0024FC09	cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc	iiiiiiiiiiiiiiii

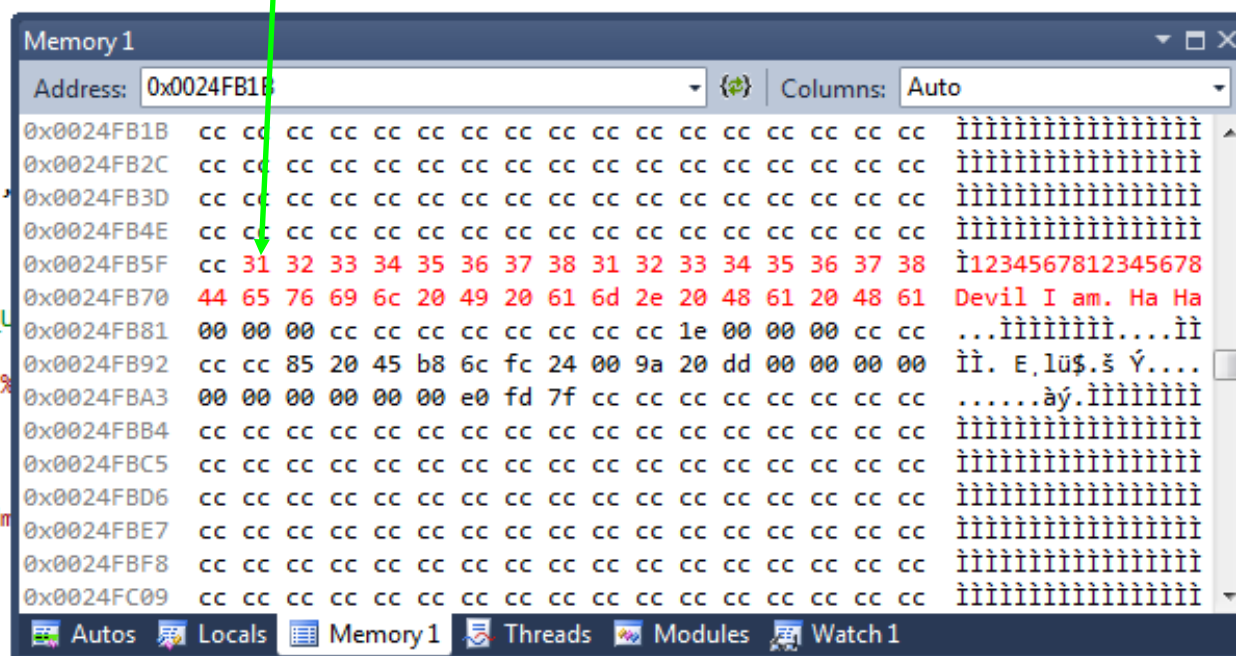


# Spuštění útočníkem - passwd

zadáno

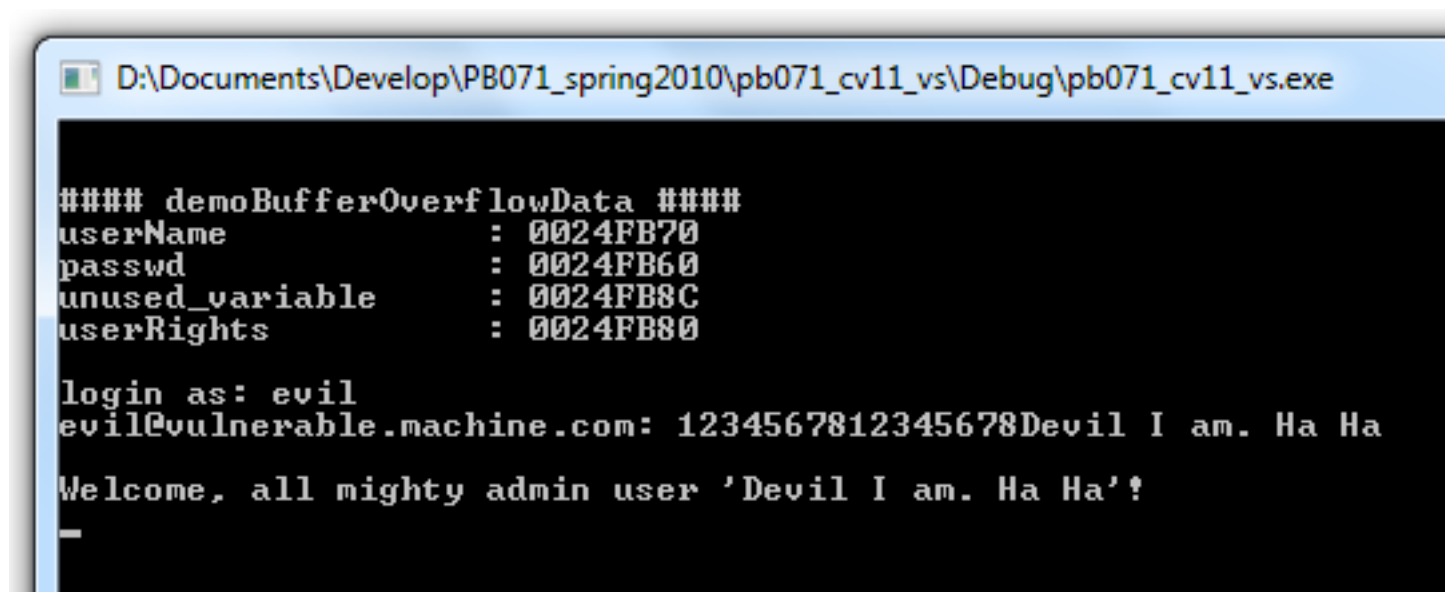
'1234567812345678Devil I am. Ha Ha'  
do passwd

```
printf("login as: ");  
fflush(stdout);  
gets(userName);  
  
// Get password  
printf("%s@vulnerable.machine.com: ",  
fflush(stdout);  
gets(passwd);  
  
// Check user rights (set to NORMAL_USER)  
if (userRights == NORMAL_USER) {  
    printf("\nWelcome, normal user '%s'",  
    fflush(stdout);  
}  
if (userRights == ADMIN_USER) {  
    printf("\nWelcome, all mighty admin '%s'",  
    fflush(stdout);  
}  
  
// How to FIX:
```



- Příliš dlouhé heslo přepsalo v paměti `userName` i `userRights`

# Spuštění útočníkem - výsledek



```
D:\Documents\Develop\PB071_spring2010\pb071_cv11_vs\Debug\pb071_cv11_vs.exe

#### demoBufferOverflowData ####
userName      : 0024FB70
passwd        : 0024FB60
unused_variable : 0024FB8C
userRights    : 0024FB80

login as: evil
evil@vulnerable.machine.com: 1234567812345678Devil I am. Ha Ha

Welcome, all mighty admin user 'Devil I am. Ha Ha'!
```

# Jak může chránit programátor?

- Důsledná kontrola délky načítaných dat
- Preventivní mazání načítaného pole
  - nebo preventivní nastavení posledního bajtu na 0
- Jazyk C nemá příliš pohodlné nástroje pro načtení vstupu s variabilní délkou
  - musíme zjistit dopředu délku vstupu a alokovat (malloc) dostatečné pole
  - nebo řešit situaci, kdy se načítaný vstup nevleze do fixního pole (např. fgets())
- Nelze spoléhat na „bezpečné“ uspořádání dat v paměti
  - různé kompilátory umístí proměnné různě

# Jak může chránit překladač?

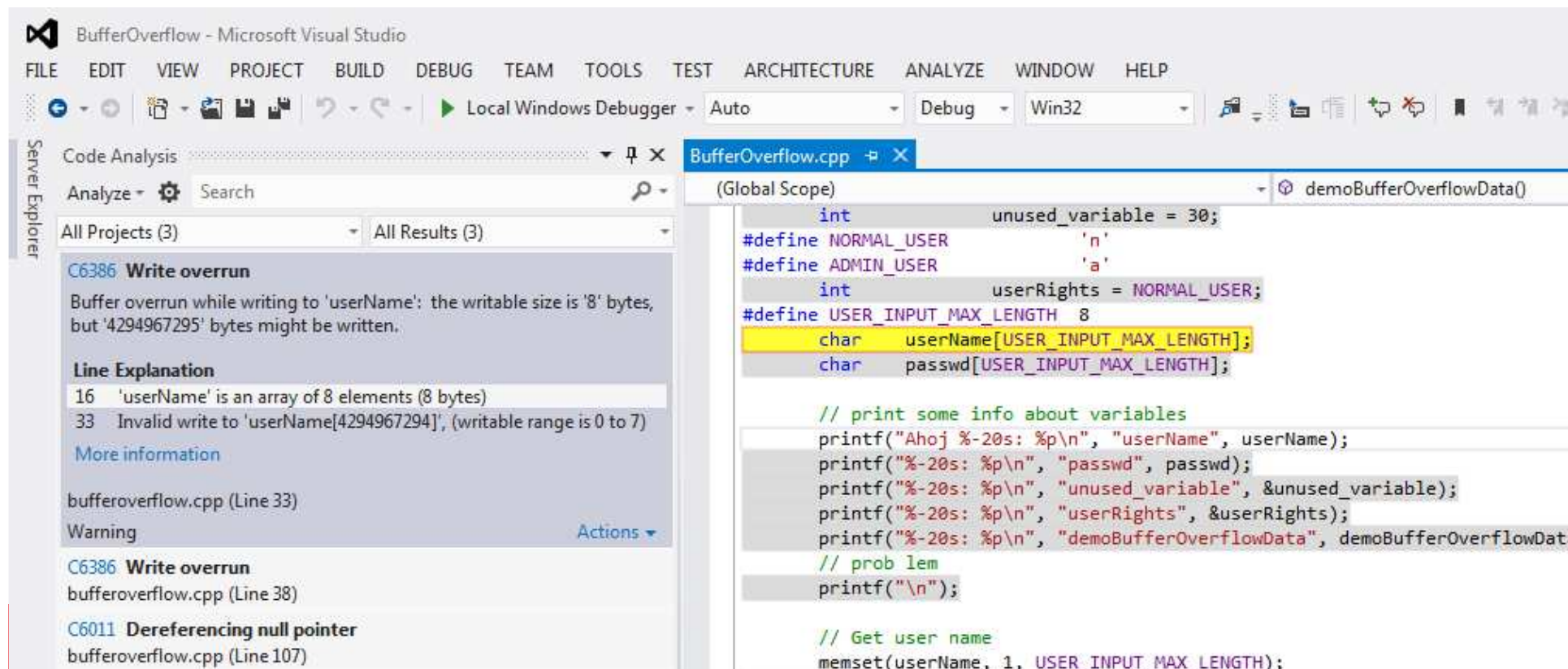
- Překladač může “obalit” citlivé objekty v paměti dodatečnou ochrannou
  - dodatečný paměťový prostor kolem polí se speciální hodnotou (např. 0xcc) – možnost následné detekce přepisu
  - náhodná hodnota (canary word) před návratovou adresou z funkce kontrolované před následování adresy
  - randomizace paměti (ASLR)
  - ochrana datové sekce programu před vykonáním (DEP)
- Dostupné přepínače překladače
  - MSVC: `/RTC1` , `/DYNAMICBASE` , `/GS` , `/NXCOMPAT`
  - GCC: `-fstack-protector-all`

# Jak může chránit dodatečná analýza?

- Statická analýza
  - probíhá nad zdrojovým kódem bez jeho spuštění
  - Např. Cppcheck, Microsoft PREfast...
- Dynamická analýza
  - probíhá nad spuštěnou binarkou programu
  - např. Valgrind (nejen memory leaks)
- Výrazná, automatizovaná pomoc
  - pozor, nedetekuje všechny chyby!

# Microsoft PREfast

- Microsoft Visual Studio 2012/3 Ultimate
  - pro studenty dostupné v rámci MSAA
- Visual Studio → Analyze → Run code analysis...



# Demo – kontrola vstupu pro system()

# Předpoklady

- Nezávislé na překladači
- Funkce `demoInsecureSystemCall()`
  - vypíše ze souboru informace o použití
  - nedovolí použít příkaz 'type' a 'dir'
- Jak může útočník vypsát obsah adresáře?



# Nedostatečné ošetření zakázaného vstupu

```
void demoInsecureSystemCall(const char* command) {  
    FILE* file = NULL;  
    printf("\n\n[USAGE INFO]: ");  
    if ((file = fopen("usage_help.txt", "r")) != NULL) {  
        char c;  
        while ((c = getc(file)) != EOF) putc(c, stdout);  
        fclose(file);  
    }  
    printf("\n\n");  
    // Printing file content is not allowed  
    if (strncmp(command, "type", strlen("type")) == 0) {  
        printf("[INFO] Type command is not allowed!\n");  
        return;  
    }  
    // Listing of directory is not allowed  
    if (strncmp(command, "dir", strlen("dir")) == 0) {  
        printf("[INFO] Dir command is not allowed!\n");  
        return;  
    }  
    // other_comands may not be allowed as well.....  
  
    // We tested for all unwanted commands, input should be safe now, execute it  
    printf("[INFO] Running command '%s'\n", command);  
    system(command);  
}
```

příkaz na spuštění

výpis nápovědy

zákaz 'type'

zákaz 'dir'

spuštění příkazu

# Jak může útočník vypsát adresář?

- Vložení bílých znaků
  - při vyhodnocování `system()` jsou později ignorovány
- Různá velikost znaků (`system()` ignoruje)
- Speciální znaky (tab...)
- Řetězení několik příkazů
- ...

```
demoInsecureSystemCall("dir");           // Directory listing is not allowed

demoInsecureSystemCall(" dir");           // Maybe, we can get around with spaces
demoInsecureSystemCall("DiR");           // ... or different character case
demoInsecureSystemCall("\011dir");        // ... or special character(s) (\011 is tab)
// ... or sequence of commands
demoInsecureSystemCall("echo You can't stop me & dir");
// ... or ...
```

# Jak může útočník vypsat adresář a soubor?

- Předpoklad: výstup volání `system()` není vypisován útočníkovi
  - i spuštění `system("dir")` nepomůže
- Lze využít několik následných volání

```
demoInsecureSystemCall("echo You can't stop me & dir > usage_help.txt");  
demoInsecureSystemCall("echo Hacked");
```

- Lze nepředpokládaně využít stávající funkčnosti
  - např. zápis výstupních dat do souboru s nápovědou

```
demoInsecureSystemCall("type top_secret.txt");  
demoInsecureSystemCall("\x20\x20\x20\x20\x20type top_secret.txt");  
demoInsecureSystemCall("notepad.exe top_secret.txt");
```

# Demo – chybná práce s řetězcí

# Textové řetězce

- Řetězec v C musí být ukončen nulou \0
- Pokud není, velké množství funkcí nefunguje
  - pokračují dokud není v paměti nula (za koncem pole)
- Funkce pro práci s řetězci
  - sprintf, fprintf, snprintf, strcpy, strcat, strlen, strstr, strchr, read...
- Funkce pro práci s pamětí
  - memcpy, memmove
  - (pokud je délka na kopírování zjištěna strlen(string))
- <http://www.awarenetwork.org/etc/alpha/?x=5>

# Kontrola hesla

```
void demoAdjacentMemoryOverflow(char* userName, char* password) {  
    char message[100];  
    char realPassword[] = "very secret password nbu123";  
    char buf[8];  
  
    memset(buf, 0, sizeof(buf));  
    memset(message, 0, sizeof(message));  
    // We will copy only characters which fits into buf  
    strncpy(buf, userName, sizeof(buf));  
  
    // Print username to standard output-nothing sensitive, right?  
    sprintf(message, "Checking '%s' password\n", buf);  
    printf("%s", message);  
    if (strcmp(password, realPassword) == 0) {  
        printf("Correct password.\n");  
    }  
    else {  
        printf("Wrong password.\n");  
    }  
}
```

očekávané heslo

kopie do lokálního pole  
Problém?

výpis veřejné informace –  
díky chybějící koncové nule i  
další paměti s heslem

# Zjištění očekávaného hesla

- Útok je často kombinace několika operací
  - nedostatečná délka paměti
  - chybějící ošetření koncové nuly
  - funkce předpokládající přítomnost koncové nuly

```
demoAdjacentMemoryOverflow("admin", "I don't know the password");  
demoAdjacentMemoryOverflow("adminxxxx", "I still don't know the password");  
demoAdjacentMemoryOverflow("admin", "very secret password nbu123");
```

```

printf("\n");

memset(buf, 0, sizeof(buf));
memset(message, 1, sizeof(message));
strncpy(buf, userName, sizeof(buf));

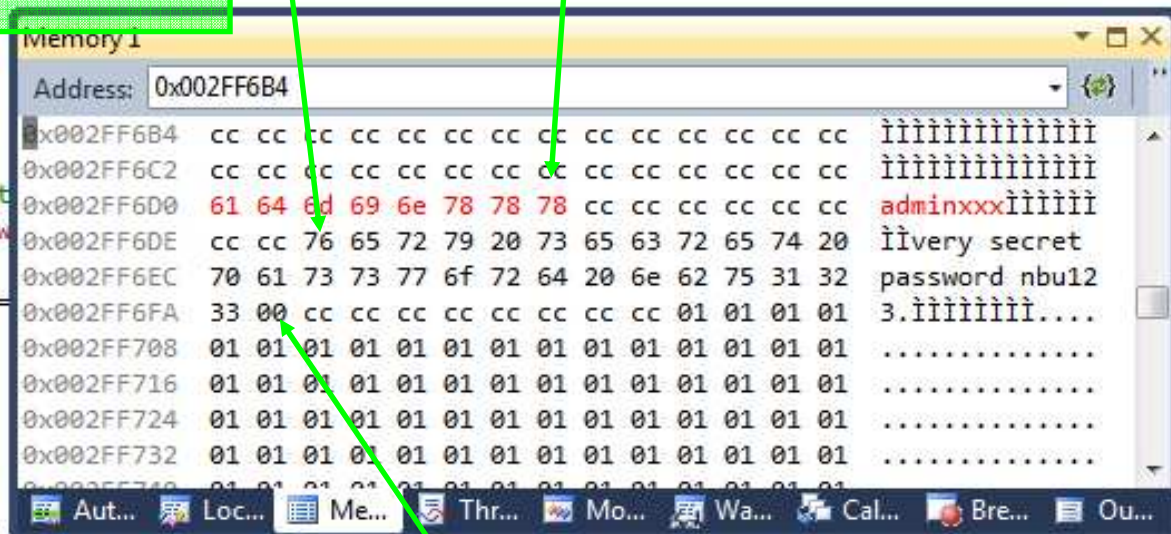
// Now print username to standard out
sprintf(message, "Checking '%s' password", buf);
printf("%s", message);
if (strcmp(password, realPassword) == 0)
    printf("Correct password.\n");
else {
    printf("Wrong password.\n");
}

// FIX: Do not allow to have non-terminated string

```

začátek  
realPassword

vložené userName bez  
koncové nuly



první koncová nula  
pro řetězec buf

```

#### demoAdjacentMemoryOverflow ####
Checking 'admin' password
Wrong password.
Checking 'adminxxx|very secret password nbu123' password
Wrong password.
Checking 'admin' password
Correct password.

```



# Shrnutí

1. Bud'te si vědomi možných problémů a útoků
  - S velkou pravdšpodobností budete vytvářet aplikace v síťovém prostředí
  - Piště pěkně, nevytvářejte snadno napadnutelný kód
  - Nástroje pro automatickou kontrolu za vás všechny problémy nevyřeší
2. Používejte bezpečné verze zranitelných funkcí
  - Secure C library (xxx\_s funkce s příponou \_s, součást standardu C11)
  - datové kontejnery, pole a řetězce s automatickou změnou velikosti (C++)
3. Kompilujte se všemi dostupnými ochrannými přepínači překladače
  - MSVC: `/RTC1` , `/DYNAMICBASE` , `/GS` , `/NXCOMPAT`
  - GCC: `-fstack-protector-all`
4. Používejte automatické nástroje pro kontrolu kódu
  - statická a dynamická analýza, fuzzing, skenery zranitelností
5. Využívejte ochrany nabízené moderními operačními systémy
  - DEP, ASRL...

# Tutoriály

- Buffer Overflow Exploitation Megaprimer (Linux)
  - <http://www.securitytube.net/groups?operation=view&groupId=4>
- Tenouk Buffer Overflow tutorial (Linux)
  - <http://www.tenouk.com/Bufferoverflowc/bufferoverflowvulexploitdemo.html>
- Format string vulnerabilities primer (Linux)
  - <http://www.securitytube.net/groups?operation=view&groupId=3>
- Buffer overflow in Easy RM to MP3 utility (Windows)
  - <https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/>