

# PB071 – Programování v jazyce C

Union, I/O, Práce se soubory

# Vnitrosemestrální test

- Termín 7.4. v 12:00 a 13:00 v D1 (příští týden)
  - Bude vypsáno hned v 14:10
- Nutno se přihlásit přes IS
- Formou papírového odpovědníku
- Celkově zisk max. 20 bodů

# Union

# union

```
union energy_t{  
    int iEnergy;  
    float fEnergy;  
    unsigned char bEnergy[10];  
};
```

- Deklarace a přístup obdobně jako **struct**
- Položky se ale v paměti překrývají
  - překryv od počáteční adresy
- Velikost proměnné typu union odpovídá největší položce
  - aby bylo možné i největší uložit
  - + případné zarovnání v paměti
- Pozor: dochází k reinterpretaci bitových dat
  - potenciální zdroj chyb a problémů
- Často kombinováno jako podčást struct s další položkou obsahující datový typ

# struct vs. union (rozložení paměti)

```
struct energy_t{  
    int iEnergy;  
    float fEnergy;  
    unsigned char bEnergy[10];  
};
```



```
union energy_t{  
    int iEnergy;  
    float fEnergy;  
    unsigned char bEnergy[10];  
};
```



# union – přístup na úrovni bajtů

- Možnost snadno přistupovat k jednotlivým bajtům většího typu (snáze než bitovými operátory)
  - (pozor na Little vs. Big endian)

```
union intByte {  
    int iValue;  
    unsigned char bValue[sizeof(int)];  
};  
  
int main(void) {  
    union intByte value = { 100};  
    // value contains bits encoding number 100 (as integer)  
    printf("%d", value.iValue);  
    printf("%c%c%c%c", value.bValue[0], value.bValue[1],  
            value.bValue[2], value.bValue[3]);  
    value.bValue[2] = 3;  
    printf("%d", value.iValue);  
    // third byte in integer was set to 3  
    return EXIT_SUCCESS;  
}
```

# union – uložení různých typů v různý čas

```
union energy_t{
    int iEnergy;
    float fEnergy;
    unsigned char bEnergy[10];
};

enum energy_type { integer, real, bigbyte};

struct avatar_energy {
    enum energy_type energyType;
    union energy_t    energy;
};

struct avatar_energy avatEnerg = {integer, .energy.iEnergy = 100};

switch (avatEnerg.energyType) {
    case integer: printf("%d", avatEnerg.energy.iEnergy); break;
    case real: printf("%f", avatEnerg.energy.fEnergy); break;
    case bigbyte: printf("%c%c",
                        avatEnerg.energy.bEnergy[0],
                        avatEnerg.energy.bEnergy[1]);
                break;
}
```

# K čemu jsou unie dobré?

- Paměťová a časová optimalizace

```
enum value_type { integer, byte};  
union value_t{  
    int iValue;  
    unsigned char bValue;  
};  
struct node {struct node* pNext;enum value_type valueType;union value_t value;};
```

- Úspora paměti, pokud je užito k ukládání více položek různých typů, ale použít vždy jen jeden
  - např. seznam s namixovanými datovými typy



# K čemu jsou unie dobré?

- Úspora času, pokud můžeme znovuvyužít existující položku s jiným typem namísto jejího znovuvytvoření
  - např. předalokovaný seznam s různými hodnotami
- Pozn.: Inicializovat lze pouze první položku
  - od C99 i další pomocí pojmenovaného inicializátoru

```
union intByte value2 = {.bValue[0] = 3, .bValue[3] = 7};
```

# Vstup a výstup I/O

# Standardní vstup a výstup

- Koncept standardního vstupu a výstupu
  - program nemusí vědět, kdo mu dává vstupní data
  - program nemusí vědět, kam vypisuje výstupní data
  - defaultně standardní vstup == **klávesnice**
  - defaultně standardní výstup == **obrazovka**
- Zařízení vstupu/výstupu lze snadno zaměnit
  - standardní vstup ze souboru
    - Windows: `program.exe < soubor.txt`
    - Unix: `cat soubor.txt | program`
  - standardní výstup do souboru
    - Windows: `program.exe > output.txt`
    - Unix: `./program > output.txt`

# Vstup a výstup v C

- Základní možnosti vstupu a výstupu už známe
  - výstup na obrazovku (`puts`, `printf`)
  - vstup z klávesnice (`getc`, `scanf`)
- Funkce pro vstup a výstup jsou poskytovány standardní knihovnou (`stdio.h`)
  - nejsou tedy přímo součástí jazyka
  - jsou ale součástí vždy dostupné standardní knihovny
- Binární data
  - jaké bajty zapíšeme, takové přečteme
- Textová data
  - na nejnižší úrovni stále binární data, ale *interpretovaná* jako text

# Textová data

- Použito pro vyjádření běžného textu
- Písmena, číslice, mezery, oddělovače, závorky...
  - tisknutelné znaky (ASCII  $\geq 32$ )
- Textová data na rozdíl od binárních přímo interpretujeme
  - s bajtem o hodnotě 71 pracujeme jako písmenem G
- Jak interpretovat ostatní (netextové) hodnoty?
  - různé zástupné symboly, pípnutí...?

000d	00h	(nul)	016d	10h	► (dle)
001d	01h	☉ (soh)	017d	11h	◄ (dc1)
002d	02h	● (stx)	018d	12h	↑ (dc2)
003d	03h	♥ (etx)	019d	13h	!! (dc3)
004d	04h	♦ (eot)	020d	14h	¶ (dc4)
005d	05h	♣ (enq)	021d	15h	§ (nak)
006d	06h	♠ (ack)	022d	16h	■ (syn)
007d	07h	• (bel)	023d	17h	‡ (etb)
008d	08h	▣ (bs)	024d	18h	↑ (can)
009d	09h	(tab)	025d	19h	↓ (em)
010d	0Ah	(lf)	026d	1Ah	(eof)
011d	0Bh	♂ (vt)	027d	1Bh	← (esc)
012d	0Ch	♀ (np)	028d	1Ch	~ (fs)
013d	0Dh	(cr)	029d	1Dh	↔ (gs)
014d	0Eh	↓ (so)	030d	1Eh	▲ (rs)
015d	0Fh	○ (si)	031d	1Fh	▼ (us)

Michael Goerz, Sept. 04 2000

# Nový řádek

- `printf("Prvni radek \n Druhy radek");`
  - nový řádek v C je speciální znak (`\n`)
- Nový řádek - implementačně závislé na OS
  - Unix: `\n` (ASCII = 10, new line)
    - `\n` posun dolů o jeden řádek
  - Windows: `\r \n` (ASCII = 13 10)
    - `\r` carriage return – návrat válce psacího stroje doleva
    - `\n` posun dolů o jeden řádek
- `printf("Prvni radek \n");`
  - na Unixu: Prvni radek `\n`
  - na Windows: Prvni radek `\r\n`

# Vyrovnávací paměť pro vstup a výstup

- Data mezi producentem a spotřebovatelem nemusí být přenesena ihned
  - text na obrazovku vypisován po řádcích
  - data na disk zapisována po blocích
  - z optimalizačních důvodů se nevolá spotřebitel pro každý elementární znak
- Produkováná data jsou ukládána do vyrovnávací paměti (tzv. buffering)
  - vyčtení proběhne při jejím zaplnění
    - (nastavení aplikace nebo OS)
  - nebo externím vynucením (fflush(stdout))

# Práce s vyrovnávací pamětí

```
int getPutChar() {  
    printf("Zadavej znaky a pak Enter: ");  
    fflush(stdout); // force output of printf  
  
    char input = 0;  
    while((input = getchar()) != '\n') {  
        putchar(input + 1);  
        //fflush(stdout);  
    }  
    printf("\n"); // Put newline to indent program output  
  
    return 0;  
}
```



# printf - podrobněji

- Často používaná funkce ze standardní knihovny
  - <http://www.cplusplus.com/reference/cstdio/printf/>
- `int printf(const char * format, ...);`
  - `%[flags][width][.precision][length]specifier`
- Tabulka běžných formátovacích znaků

specifier	Output	Example
c	Character	a
d or i	Signed decimal integer	392
e	Scientific notation (mantise/exponent) using e character	3.9265e+2
E	Scientific notation (mantise/exponent) using E character	3.9265E+2
f	Decimal floating point	392.65
g	Use the shorter of %e or %f	392.65
G	Use the shorter of %E or %f	392.65
o	Unsigned octal	610
s	String of characters	sample
u	Unsigned decimal integer	7235
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (capital letters)	7FA
p	Pointer address	B800:0000
n	Nothing printed. The argument must be a pointer to a signed int, where the number of characters written so far is stored.	
%	A % followed by another % character will write % to stdout.	%

# Počet desetinných míst a délka čísla

- Mezi symbol % a symbol typu lze umístit dodatečné formátování
  - %5.2f
- Omezení/rozšíření počtu vypisovaných desetinných míst
  - defaultně 6 desetinných míst
  - %.2f, %.8f, %.0f
- Zarovnání výpisu na zadaný celkový počet cifer
  - %10f

# Výpis čísla - ukázka

```
#include <stdio.h>
int main() {
    float fValue = 3.1415926535;
    printf("%f", fValue);
    printf("\n");
    printf("%.2f", fValue);
    printf("\n");
    printf("%.8f", fValue);
    printf("\n");
    printf("%10f", fValue);
    printf("\n");
}
```

3.141593  
3.14  
3.14159274  
3.141593

# Možnosti výpisu ukazatele a soustavy

- Výpis ukazatele

- `printf("%p", &value);`

- Výpis vhodné číselné soustavy

- `%d` desítková
- `%o` osmičková
- `%x` šestnáctková (často se uvádí jako `0x%x` → `0x30`)

```
#include <stdio.h>
int main() {
    int value = 16;
    printf("%p", &value);
    // e.g., 0022ffc1
    printf("%d %o %x", value, value, value);
    // 16 20 10
    return 0;
}
```

# printf – chybný typ argumentu

- Pozor na chybnou specifikaci parametru
  - formátování se provede, ale chybně přetypované
  - viz. funkce s proměnným počtem parametrů
    - `va_arg(arg, int)`

- Typicky vypíše chybnou hodnotu

```
float fValue = 3.1415926535;  
printf("%d", fValue);
```

→ 1610612736

- Při chybné specifikaci `%s` výpis smetí nebo pád
  - v paměti se hledá koncová nula

# Formátované načítání ze vstupu

# scanf

```
int value;  
scanf("%d", &value);  
char smallString[50];  
scanf("%s", smallString);
```

- **int** scanf(**char\*** format , ... );
- Analogie printf, ale pro načítání ze vstupu
  - ze standardního vstupu se čtou hodnoty a ukládají do poskytnutých argumentů
  - argumenty poskytnuty jako ukazatele
  - formát obdobný jako pro printf (nepoužívají se počty desetinných cifer)
- Pokud je načítáno více hodnot, tak musí být na vstupu odděleny bílým znakem
  - mezera, tabulátor
- Pozor: Při čtení jsou bílé znaky zahazovány
- scanf vrací počet načtených položek
  - EOF (End Of File == -1), pokud se nepodařilo načíst nic

# scanf ukázka Avatar

```
enum weapon_t {sword,axe,bow};  
struct avatar_t {  
    char nick[32];  
    float energy;  
    enum weapon_t weapon;  
};  
  
void scanfDemo() {  
    struct avatar_t myAvat;  
    scanf("%s", &myAvat.nick);  
    scanf("%f", &myAvat.energy);  
    scanf("%d", &myAvat.weapon);  
}
```



# Problematika ošetření délky vstupu

- Ošetření vstupních dat je velmi důležitá věc
  - umožňuje korektně upozornit uživatele
  - zamezuje nechtěnému chování programu
  - zamezuje záměrnému útoku na program
- `scanf` a řetězec: `scanf ("%s", smallString) ;`
  - řetězec má omezenou délku, zadaný vstup může být delší
  - `%50s` omezí načtenou délku na 50 znaků (pak ale na 51 koncová nula)

```
#include <stdio.h>
int main() {
    char smallString[51];
    scanf("%s", smallString);

    scanf("%50s", smallString);
    printf("%s", smallString);
}
```

# Formátovaný zápis a čtení z řetězce

# sprintf, sscanf,

- printf a scanf pracují se standardním vstupem a výstupem
- Namísto vstupu a výstupu lze použít pole znaků
- **int** sprintf ( **char** \* str, **const char** \* format, ... );
  - stejné jako printf, výstup jde ale do řetězce
  - vrací počet zapsaných **znaků**
  - pozor na celkovou délku výstupu
- **int** sscanf (**const char** \* str, **const char** \* format, ...);
  - stejné jako scanf, výstup načítán z řetězce
  - vrací počet načtených **položek** (ne znaků)

# Ukázka sprintf

```
#include <stdio.h>
enum weapon_t {sword,axe,bow};
struct avatar_t {
    char nick[32];
    float energy;
    enum weapon_t weapon;
};

int main() {
    struct avatar_t myAvat = {"Hell", 100, axe};
    char message[1000];
    sprintf(message, "Avatar '%s' with energy %.2f is ready!",
            myAvat.nick, myAvat.energy);
    puts(message);
    return 0;
}
```

# Secure C library

- Bezpečnější varianty často zneužívaných funkcí
  - Kontrola mezí při manipulaci s řetězcí
  - Lepší ošetření chyb
- Dostupné také v novém C standardu ISO/IEC 9899:2011
- Microsoftův překladač obsahuje dodatečně rozšířené bezpečnostní varianty běžných CRT funkcí
  - MSVC překladač vypíše varování C4996, o něco více pokrytých funkcí než v C11
- 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://msdn.microsoft.com/en-us/library/wd3wzwt%28v=vs.80%29.aspx>
  - <http://www.drdobbs.com/cpp/the-new-c-standard-explored/232901670>

# Secure C library – vybrané funkce

## ● Formátovaný vstup a výstup

- Funkce přijímají dodatečný argument s délkou pole
- `gets_s`
- `scanf_s`, `wscanf_s`, `fscanf_s`, `fwscanf_s`, `sscanf_s`, `swscanf_s`, `vscanf_s`, `vfwscanf_s`, `vsscanf_s`, `vswscanf_s`
- `fprintf_s`, `fwprintf_s`, `printf_s`, `snprintf_s`, `swprintf_s`, `sprintf_s`, `swprintf_s`, `vfprintf_s`, `vfwprintf_s`, `vprintf_s`, `vwprintf_s`, `vsprintf_s`, `vsnprintf_s`, `vswprintf_s`

## ● Funkce pro práci se soubory

- Přijímají ukazatel na `FILE*`
- Vrací chybový kód
- `tmpfile_s`, `tmpnam_s`, `fopen_s`, `freopen_s`

```
char *gets(  
    char *buffer  
);  
  
char *gets_s(  
    char *buffer,  
    size_t sizeInCharacters  
);
```

# Secure C library – vybrané funkce

- Okolní prostředí (environment, utilities)
  - getenv\_s, wgetenv\_s
  - bsearch\_s, qsort\_s
- Funkce pro kopírování bloků paměti
  - memcpy\_s, memmove\_s, strcpy\_s, wcscpy\_s, strncpy\_s, wcsncpy\_s
- Funkce pro spojování řetězců
  - strcat\_s, wcscat\_s, strncat\_s, wcsncat\_s
- Vyhledávací funkce
  - strtok\_s, wcstok\_s
- Funkce pro manipulaci času...

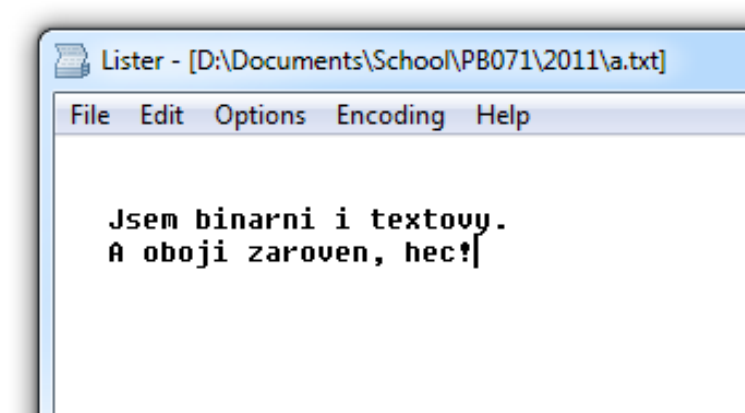
# Práce se soubory



# Typy souborů

- Soubory obsahující binární data
  - při zápisu a čtení jsou ukládána data přesně tak, jak je zadáte
- Soubory obsahující textová data
  - přesněji: binární soubor interpretovaný jako text
  - při čtení a zápisu může docházet k nahrazení některých bajtů

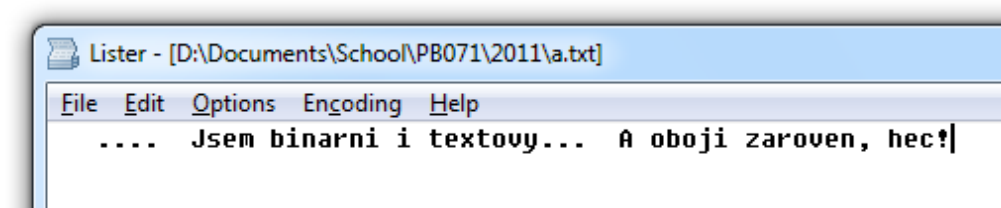
# Binární vs. textový



Lister - [D:\Documents\School\PB071\2011\a.txt]

File Edit Options Encoding Help

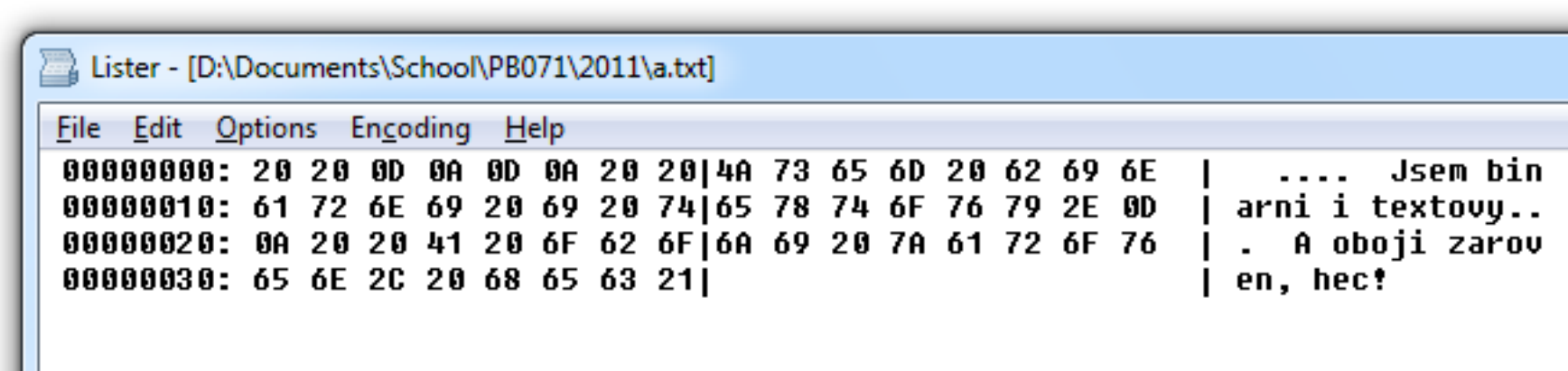
Jsem binarni i textovy.  
A oboji zaroven, hec!



Lister - [D:\Documents\School\PB071\2011\a.txt]

File Edit Options Encoding Help

.... Jsem binarni i textovy... A oboji zaroven, hec!



Lister - [D:\Documents\School\PB071\2011\a.txt]

File Edit Options Encoding Help

00000000:	20 20 0D 0A 0D 0A 20 20	4A 73 65 6D 20 62 69 6E		.... Jsem bin
00000010:	61 72 6E 69 20 69 20 74	65 78 74 6F 76 79 2E 0D		arni i textovy..
00000020:	0A 20 20 41 20 6F 62 6F	6A 69 20 7A 61 72 6F 76		. A oboji zarov
00000030:	65 6E 2C 20 68 65 63 21			en, hec!

# Práce se soubory

1. Otevřeme soubor (připojíme se k souboru)
  - `fopen()`
  - získáme ukazatel na soubor (`FILE*`)
2. Čteme/zapisujeme z/do souboru
  - `fscanf, fprintf, fread, fwrite...`
  - využíváme ukazatel na soubor
3. Ukončíme práci se souborem (zavřeme soubor)
  - `fclose()`

# Jak otevřít soubor – mód otevření

- Mód otevření volit na základě požadovaného chování
  - Chceme číst z existujícího souboru? "**r**"
  - Chceme vytvořit nový soubor a zapisovat do něj? "**w**"
  - Chceme zapisovat na konec existujícího souboru? "**a**"
  - Chceme číst i zapisovat do nového souboru? "**w+**"
  - Chceme číst i zapisovat do existujícího souboru?
    - čtení i zápis kdekoli "**r+**"
    - čtení kdekoli, zápis vždy na konec "**a+**"
  - Chceme s daty pracovat v binárním namísto textového režimu? Přidáme b: "\_b" (např. "**rb**")
- <http://www.cplusplus.com/reference/clibrary/cstdio/fopen/>

# Otevření souboru

```
FILE* file = fopen("D:\\test.txt", "r");
```

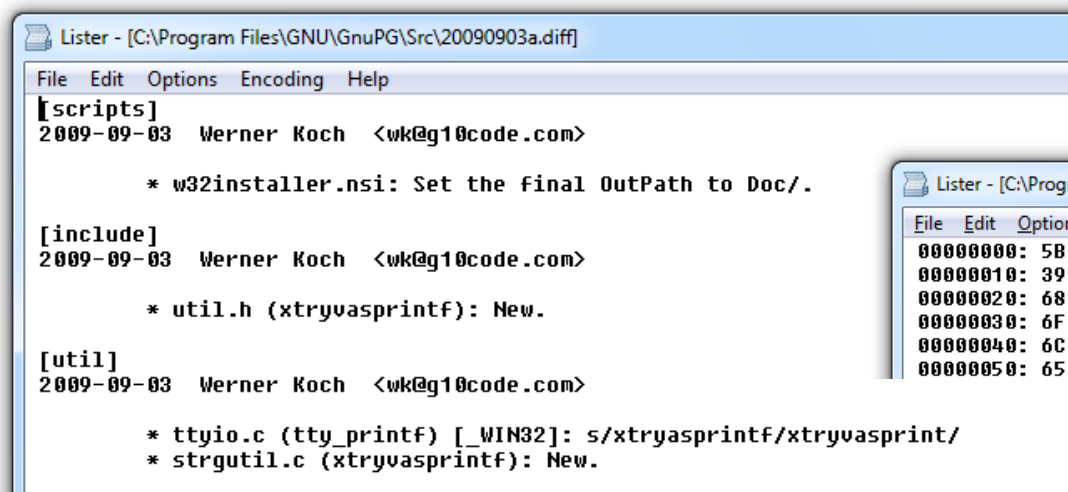
- **FILE\* fopen(const char\* filename, const char\* mode);**
- filename obsahuje cestu k souboru
  - relativní: test.txt, ../test.txt
  - absolutní: c:\\test.txt
- Pozor na znak '\\' v řetězci obsahující cestu
  - C pokládá \\ za speciální znak, nutno použít escape sekvenci \\
  - "c:\\test.txt" → "c:\\\\test.txt"
- mode obsahuje specifikaci způsobu otevření
  - čtení/zápis/přidání na konec, textový/binární režim
- Při korektním otevření získáme ukazatel typu FILE
  - při chybě NULL
  - nemusíme "znát" deklaraci FILE (interní záležitost OS)

# Poznámky k otevření souboru

- Defaultně se soubor otvírá jako textový
  - na Unixu je textový i binární mód identický
  - na Windows se nahrazují konce řádků
- Pozor na smazání existujícího souboru
  - `fopen("existuje.txt", "w")` → `existuje.txt` velikost 0
- Pozor na situaci, kdy soubor neexistuje
  - `fopen("neexistuje.txt", "r") == NULL`
- Pokud otevřeme soubor pro čtení i zápis ("`rw`"), mezi operací čtení a zápisu by mělo být vynuceno vyprázdnění vyrovnávací paměti (`fflush()`)

# Problém s koncem řádku

- Zobrazení textového souboru vytvořeného na Unixu ve Windows
- Windows očekává konec řádku jako \r\n



```
Lister - [C:\Program Files\GNU\GnuPG\Src\20090903a.diff]
File Edit Options Encoding Help
[scripts]
2009-09-03 Werner Koch <wk@g10code.com>

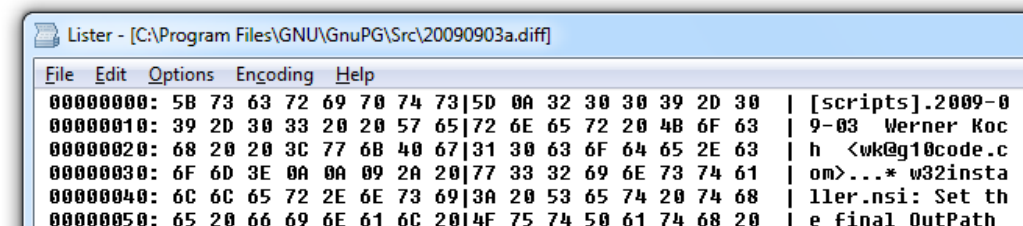
    * w32installer.nsi: Set the final OutPath to Doc/.

[include]
2009-09-03 Werner Koch <wk@g10code.com>

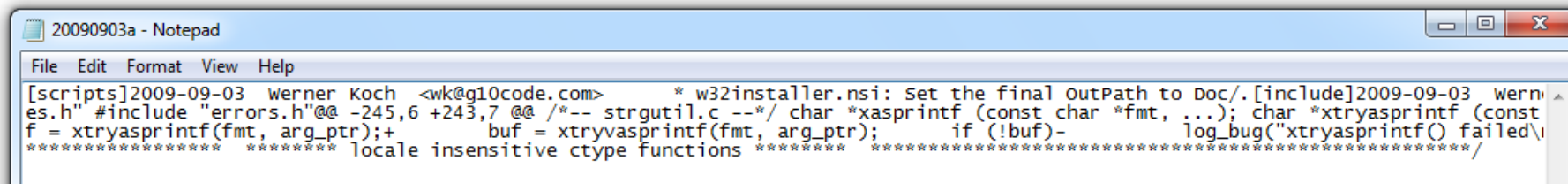
    * util.h (xtryvasprintf): New.

[util]
2009-09-03 Werner Koch <wk@g10code.com>

    * ttyio.c (tty_printf) [_WIN32]: s/xtryvasprintf/xtryvasprint/
    * strgutil.c (xtryvasprintf): New.
```



```
Lister - [C:\Program Files\GNU\GnuPG\Src\20090903a.diff]
File Edit Options Encoding Help
00000000: 5B 73 63 72 69 70 74 73|5D 0A 32 30 30 39 2D 30 | [scripts].2009-0
00000010: 39 2D 30 33 20 20 57 65|72 6E 65 72 20 4B 6F 63 | 9-03 Werner Koc
00000020: 68 20 20 3C 77 6B 40 67|31 30 63 6F 64 65 2E 63 | h <wk@g10code.c
00000030: 6F 6D 3E 0A 0A 09 2A 20|77 33 32 69 6E 73 74 61 | om>...* w32insta
00000040: 6C 6C 65 72 2E 6E 73 69|3A 20 53 65 74 20 74 68 | ller.nsi: Set th
00000050: 65 20 66 69 6E 61 6C 20|4F 75 74 50 61 74 68 20 | e final OutPath
```



```
20090903a - Notepad
File Edit Format View Help
[scripts]2009-09-03 werner koch <wk@g10code.com> * w32installer.nsi: Set the final OutPath to Doc/. [include]2009-09-03 werni
es.h" #include "errors.h"@ -245,6 +243,7 @@ /*-- strgutil.c --*/ char *xasprintf (const char *fmt, ...); char *xtryasprintf (const
f = xtryasprintf(fmt, arg_ptr);+ buf = xtryvasprintf(fmt, arg_ptr); if (!buf)- log_bug("xtryasprintf() failed\l
***** locale insensitive ctype functions *****
```

# Aktuální pozice v souboru

- Po otevření souboru je interně uchována aktuální pozice v souboru
  - začátek souboru (módy read “r” a write “w”)
  - konec souboru (mód append “a”)
- Čtení a zápis probíhá na aktuální pozici
- Při čtení/zápisu dochází automaticky k posunu o přečtené/zapsané znaky
- Zjištění aktuální pozice
  - `long int ftell ( FILE * stream );`



# Zavření souboru - fclose

- `int fclose ( FILE * stream );`
- Zavře soubor asociovaný s ukazatelem stream
  - vrací 0 pokud OK
  - i v případě chyby přestane být stream asociovaný se souborem
- Při ukončení programu jsou automaticky uzavřeny všechny otevřené soubory
- Otevřené soubory nesou režii na straně OS
  - může dojít k vyčerpání systémových prostředků

# Čtení ze souboru

- Čte se z aktuální pozice v souboru
  - po přečtení se pozice posune těsně za přečtená data
- Načtení jednoho znaku
  - `int getc ( FILE * stream );`
- Načtení jedné řádky (ukončené `\n`)
  - `char * fgets ( char * str, int num, FILE * stream );`
- Formátované čtení do proměnných
  - `int fscanf ( FILE * stream, const char * format, ... );`
- Blokové čtení na binární úrovni
  - `size_t fread ( void * ptr, size_t size, size_t count, FILE * stream );`
  - načte blok bajtů o zadané délce: `size * count`

# Čtení ze souboru – ukázka po znacích

```
#include <stdio.h>
int main() {
    FILE* file = NULL;
    char fileName[] = "D:\\test.txt";
    if ((file = fopen(fileName, "r")) {
        int value; char chvalue;
        while((value = getc(file)) != EOF) {
            chvalue = value;
            putchar(chvalue);
        }
        fclose(file);
    }
}
```

# Zápis do souboru

- Zapisuje se na aktuální pozici v souboru
  - po zápisu se pozice posune těsně za zapsaná data
- Zápis jednoho znaku
  - `int putc (int character, FILE * stream);`
- Zápis řetězce
  - `int fputs(const char * str, FILE * stream);`
  - pokud chceme zapsat řádku, ukončíme řetězec `"\n"`
- Formátovaný zápis
  - `int fprintf (FILE * stream, const char * format, ...);`
- Blokový zápis na binární úrovni
  - `size_t fwrite (const void* ptr, size_t size, size_t count, FILE* stream);`
  - zapíše blok bajtů `ptr` o zadané délce: `size * count`

# Formátovaný zápis do souboru

```
#include <stdio.h>
enum weapon_t {sword,axe,bow};
struct avatar_t {
    char nick[32];
    float energy;
    enum weapon_t weapon;
};

void writeDemo() {
    struct avatar_t myAvat = {"Hell", 100, axe};
    FILE* file = NULL;
    char fileName[] = "D:\\\\avat1.txt";
    if ((file = fopen(fileName, "w"))) {
        fprintf(file, "Avatar '%s': energy=%.2f, weapon=%d",
            myAvat.nick, myAvat.energy, myAvat.weapon);
        fclose(file);
    }
}
```

# Aktuální pozice v souboru - změna

- Aktuální pozici v souboru lze měnit bez čtení/zápisu
- `int fseek (FILE * stream, long int offset, int origin);`
  - zadaný offset vzhledem k origin
  - `SEEK_SET` – začátek souboru
  - `SEEK_CUR` – aktuální pozice
  - `SEEK_END` – konec souboru
- `void rewind (FILE * stream);`
  - přesune aktuální ukazatel na začátek souboru

# stdin, stdout, stderr

- Standardní soubory
- Automaticky otevřeny a zavřeny
- `printf()` == `fprintf(stdout)`
- `scanf()` == `fscanf(stdin)`
- `getchar()` == `getc(stdin)`

# Odstranění, přejmenování, dočasný soubor

- **int** remove (**const char** \* filename);
  - odstranění souboru dle jména (cesty)
- **int** rename (**const char** \* oldname, **const char** \* newname);
  - přejmenování souboru
- **FILE\*** tmpfile (**void**);
  - otevře dočasný unikátní soubor
  - automaticky zaniká při konci programu
- **char\*** tmpnam (**char** \* str);
  - vrátí unikátní neobsazené jméno souboru
  - POZOR: může dojít k jeho obsazení před otevřením



# Soubor – testování konce

- Používejte konstantu EOF (End Of File)
- V dokumentaci ke konkrétní funkci je uveden případ výskytu a použití

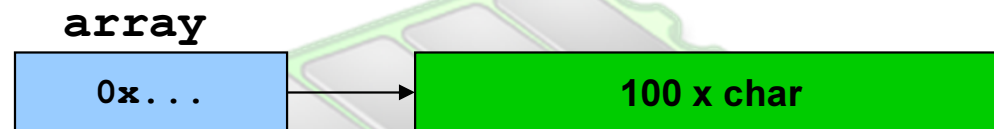
```
#include <stdio.h>
int main() {
    FILE* file = NULL;
    char fileName[] = "D:\\test.txt";
    if ((file = fopen(fileName, "r")) {
        int value;
        while((value = getc(file)) != EOF) {
            putchar(value);
        }
        fclose(file);
    }
}
```

# Funkce pro široké (UNICODE) znaky

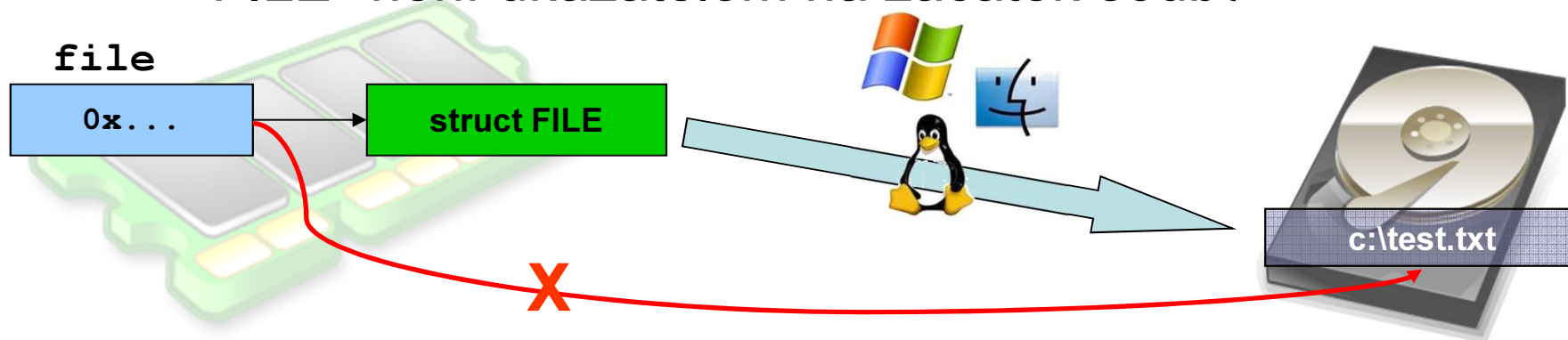
- Hlavičkový soubor `wchar.h`
- Stejně jako `char`, ale funkce s vloženým 'w' do názvu
  - `fwprintf`, `putwchar` ...

# FILE\* vs. char\*

# char\* vs. FILE\*



- `char array[100];`
  - array obsahuje adresu začátku pole o 100 znacích
  - můžeme provádět ukazatelovou aritmetiku
- `FILE* file = fopen("c:\\test.txt", "r");`
  - file obsahuje ukazatel na strukturu typu FILE
  - operační systém využívá FILE pro manipulaci souboru
  - FILE\* není ukazatelem na začátek souboru!



# char\* vs. FILE\*

- Pro soubor nelze ukazatelová aritmetika
  - `file += 2; ...` skočí na paměť za strukturou FILE
  - aktuální pozice v souboru je uložena v položce FILE
- Pro soubor nelze používat funkce typu `strcpy`
  - `strcpy(file, "BAD"); ...` zapisujeme do paměti se strukturou FILE, nikoli do samotného souboru
- FILE je platformově závislá struktura
  - nedoporučuje se spoléhat/využívat přímo její položky
  - operační systém si obsah struktury FILE spravuje sám
    - při každém otevření/čtení/zápisu....

## LINUX

```
typedef struct {
    int         level;      /* fill/empty level of buffer */
    unsigned    flags;      /* File status flags           */
    char        fd;         /* File descriptor             */
    unsigned char hold;     /* Ungetc char if no buffer   */
    int         bsize;      /* Buffer size                  */
    unsigned char *buffer;  /* Data transfer buffer        */
    unsigned char *curp;    /* Current active pointer      */
    unsigned    istemp;     /* Temporary file indicator    */
    short       token;      /* Used for validity checking */
} FILE;
```

## WINDOWS

```
typedef struct _iobuf {
    char*    _ptr;
    int      _cnt;
    char*    _base;
    int      _flag;
    int      _file;
    int      _charbuf;
    int      _bufsiz;
    char*    _tmpfname;
} FILE;
```

# Další práce se souborovým systémem

- Jak zjistit jména souborů v adresáři?
  - Jak změnit aktuální adresář?
  - Jak zjistit atributy souboru (čas, práva)?
  - Jak...?
- 
- Funkce nabízené standardní knihovnou C nestačí
  - řešením je POSIX - později

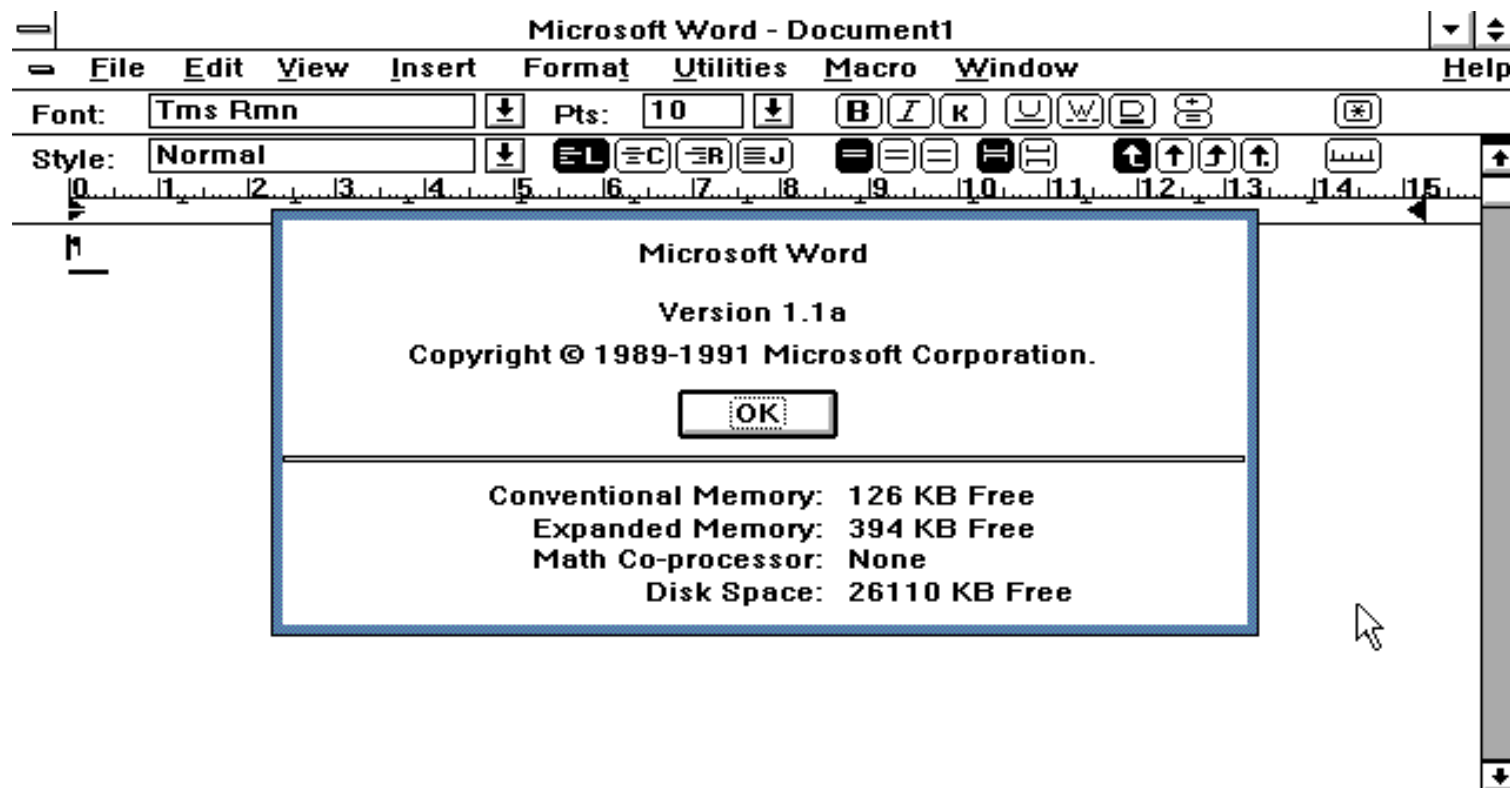
# Shrnutí

- Vstup a výstup
  - abstrakce od konkrétního vstupu/výstupu
  - standardní vstup může být klávesnice, soubor...
- Práce se soubory
  - nutnost interakce s okolním OS
  - pozor na uzavírání souborů po skončení práce



# Bonus – překvapení 😊

# Word for Windows 1.1a (1983)

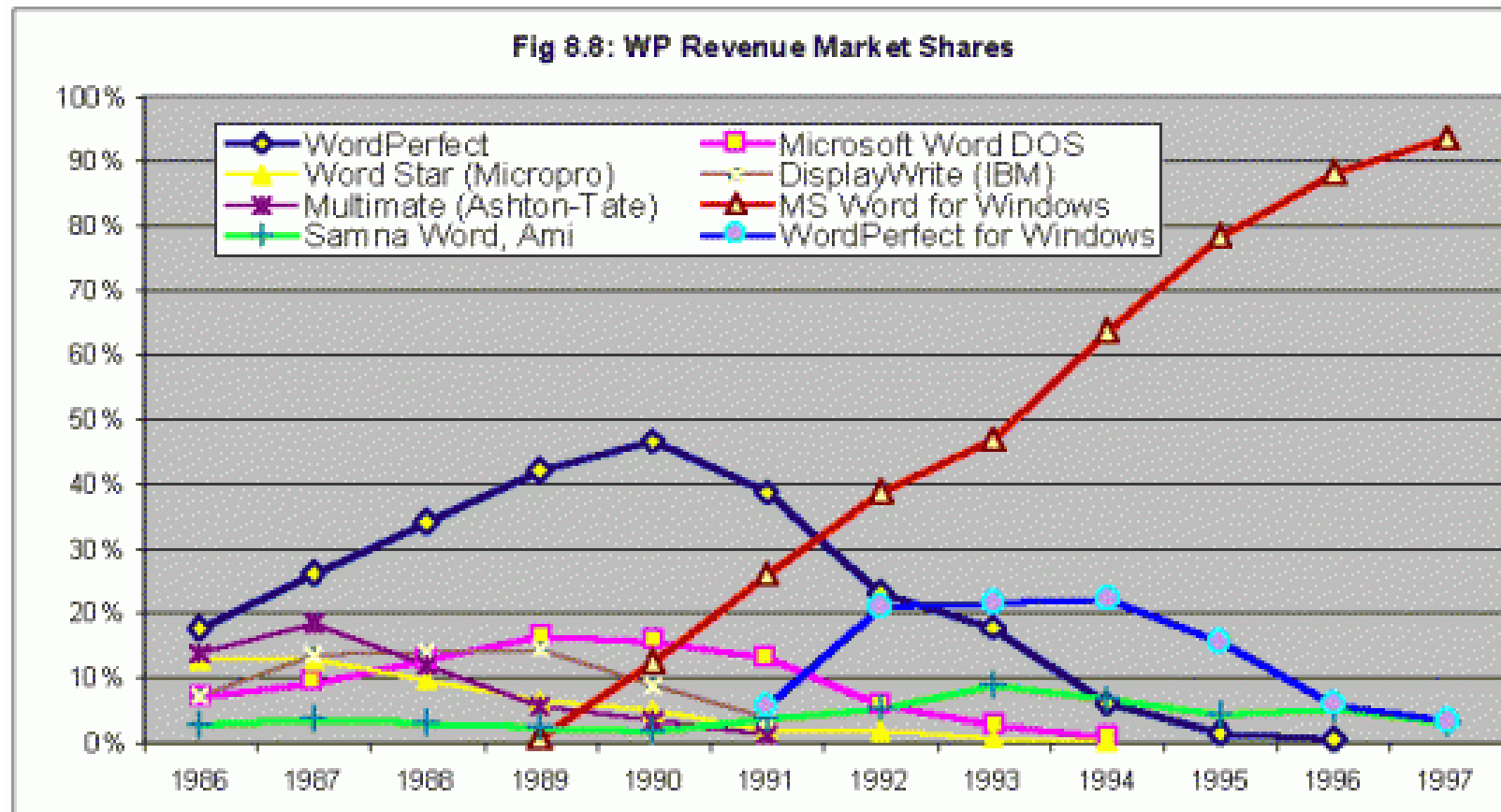


For Help, press F1

[http://www.computerhistory.org/\\_static/atcm/microsoft-word-for-windows-1-1a-source-code/](http://www.computerhistory.org/_static/atcm/microsoft-word-for-windows-1-1a-source-code/)

- Napsáno v C# pomocí dot.net frameworku
  - ne tak docela 😊

# Word for Windows - velký úspěch



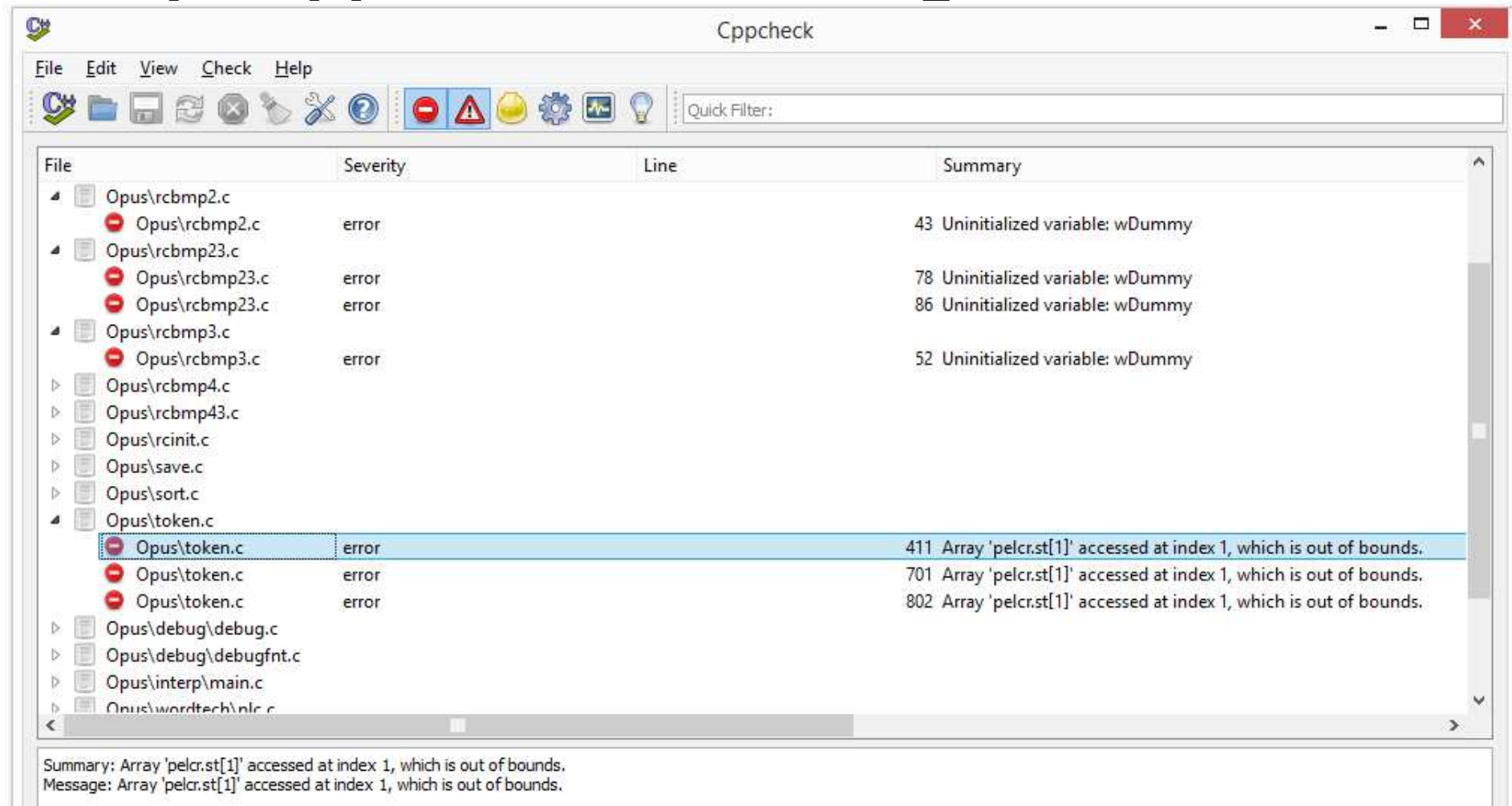
[http://www.computerhistory.org/\\_static/atcm/microsoft-word-for-windows-1-1a-source-code/](http://www.computerhistory.org/_static/atcm/microsoft-word-for-windows-1-1a-source-code/)

# Word for Windows 1.1a (1983)

- “*We set out to write an editor and we finished it about three months.*” Charles Simonyi
  - Pak přechod z Xeroxu do Microsoftu
  - Za další rok hotový nový program
- Napsáno v jazyku C
- Zdrojové kódy dostupné  
<http://www.computerhistory.org/static/atcm/microsoft-word-for-windows-1-1a-source-code/>

# Word 1.1a – Cppcheck

<http://cppcheck.sourceforge.net/>



# Word 1.1a - Source monitor

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

The screenshot shows the SourceMonitor application window. The main window displays a table titled "Files in C Project 'word', Checkpoint 'Baseline' [Base Directory: 'D:\Documents\Develop\Word 1.1a CHM Distribution\Opus\']". The table lists various source files and their associated statistics. Below this, a smaller window titled "C Checkpoints In Project 'word'" shows a table of checkpoints, with the "Baseline" checkpoint selected.

File Name	Lin...	Statements	% Branches	% Comments	Functions	Avg Stmts/Function	Max Complexity	Max Depth	Avg Depth	A...
idle.c	1444	841	25,6	16,7	9	57,9	166	5	1,51	
wordtech\savefast.c	4259	2423	19,5	14,7	29	30,6	108	9+	2,31	
wordtech\disp3.c	3015	1557	22,0	18,1	41	13,4	88	9+	1,92	
renum.c	1622	968	29,3	15,3	14	27,5	79	7	2,20	
wordtech\prcsubs.c	2445	995	25,6	23,6	13	20,8	75	9+	2,20	
rtftrans.c	1163	678	22,0	11,1	9	26,2	65	9+	3,06	
interp\exp.c	2634	1336	32,6	17,3	21	21,3	55	9+	3,41	
eldde.c	1524	900	21,7	13,8	29	8,3	46	7	2,11	
debug\debug2.c	2258	1433	21,5	5,3	37	20,2	44	7	1,86	
ddeclnt.c	1859	964	23,0	16,6	29	10,9	43	9+	2,30	
help.c	2059	1058	25,9	24,1	21	13,4	39	5	1,53	
cmdwnd.c	1512	799	18,3	15,1	17	8,7	38	7	1,51	
wordtech\block.c	1613	914	22,8	16,1	15	19,3	35	6	1,89	
wordtech\format.c	3087	1610	23,1	27,2	21	8,5	33	9+	3,62	
dlgrec.c	1378	807	28,6	7,0	19	10,8	32	6	1,60	
wordtech\select.c	4116	2151	31,8	18,9	45	8,8	32	7	2,12	
interp\core.c	3602	1846	35,3	14,8	42	13,2	31	9+	3,34	
wordtech\outline.c	1024	540	24,6	16,0	11	0,6	20	7	2,16	

Checkpoint Name	Created ...	Fi...	Lines	Statements	% Branches	% Comments	Functions	Avg Stmts/Function	Max Complexity	Ma
Baseline	31 Mar 2014	349	346 566	179 289	21,4	17,4	3 490	8,6	166	