PB071 – Principy nízkoúrovňového programování

Union, I/O, Práce se soubory

Slidy pro komentáře (děkuji!):
https://drive.google.com/file/d/1-6gvAzivetENUE54Xc_0yn75r9TbFwp4/view?usp=sharing

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 typ dat uložených v union

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

`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?

- Úspora času, pokud můžeme znovu-využít existující položku s jiným typem namísto jejího znovuvytvoření
  - např. předalokovaný seznam s různými hodnotami

```
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;};
```

- Přímá inicializace pro první položku, další pomocí pojmenovaného inicializátoru (od C99, preferujte)

```
union intByte value2 = { 1 };
union intByte value2 = {.bValue[0] = 3, .bValue[3] = 7};
```

Vstup a výstup I/O

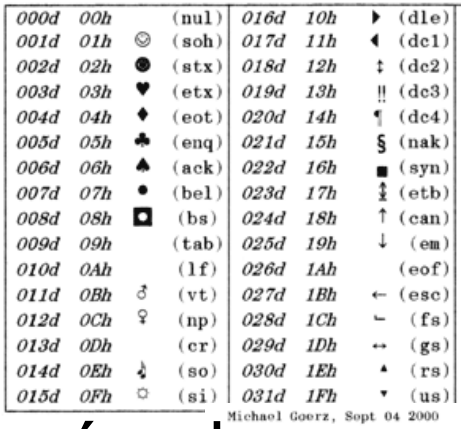
Standardní vstup a výstup

- 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: `./program < soubor.txt`
  - 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 (téměř) vždy dostupné standardní knihovny
    - Výjimkou jsou některé omezené (embedded) platformy nebo jádro OS
    - (BTW: Lze kompilovat i bez standardních knihoven `gcc -nostdlib`)
- Binární data
  - jaké bajty zapíšeme, takové přečteme
- Textová data
  - na nejnižší úrovni stále binární data, ale intepretovaná jako text

Textová data



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

- Použito pro vyjádření běžného textu
- Písmena, číslice, mezery, oddělovače, závorky...
  - tisknutelné znaky (`ASCII >= 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...?

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 doprava
    - `\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

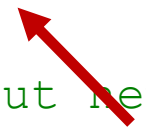
- Data mezi producentem a spotřebitelem nemusí být přenesena ihned
  - text na obrazovku vypisován po řádcích
  - data na disk zapisována po blocích
  - z optimalizačních důvodů se nevolá spotřebitel pro každý elementární znak
- Produkovaná data jsou ukládána do vyrovnávací paměti (tzv. buffering)
  - 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

    int input = 0;
    while((input = getchar()) != '\n') {
       putchar(input + 1);
       //fflush(stdout);
    }
    printf("\n"); // Put newline to indent program output

    return 0;
}
```



Použití fflush() na tomto místě je vysoce neoptimální, odstraňuje výhody použití vyrovnávací paměti (výrazné zpomalení)

printf - podrobněji

- Často používaná funkce ze standardní knihovny
  - http://www.cplusplus.com/reference/clibrary/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
  - Alespoň 10 znaků (včetně tečky), pokud méně, tak doplnění, pokud je více, tak se neořezává

Výpis čísla - ukázka

```c
#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)

```c
#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
- Překladač může upozornit warningem (neignorujte)

Formátované načítání ze vstupu

scanf

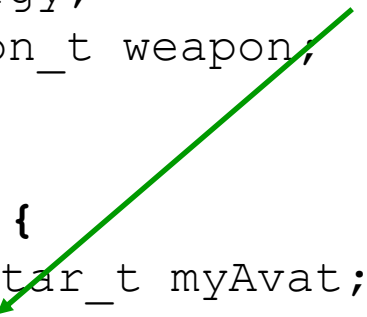
```
int value = 0;
scanf("%d", &value);
char smallString[50];
scanf("%s", smallString);
```

- `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, tabulator (kromě %c)
- 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("%31s", &myAvat.nick);
        scanf("%f", &myAvat.energy);
        scanf("%d", &myAvat.weapon);
}
```



"%31s" omezuje počet načítaných znaků na max. 31 (nick[32] a koncová nula)

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)
- „Fuzzing“ testování
  - Zašlete programu náhodný vstup
  - Různá délka (i 100kB), různý obsah
  - Program padá => problém
  - Radamsa, MiniFuzz, Peach, AFL…

```c
#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)
- Bezpečnější varianta funkce sprintf() je snprintf()
  - Poskytujeme i délku pole, do kterého zapisujeme
  - Funkce provede kontrolu meze pole

Ukázka sprint a snprintf

```c
#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};
        const unsigned int MESSAGE_LEN = 1000;
        char message[MESSAGE_LEN];
        sprintf(message, "Avatar '%s' with energy %.2f is ready!",
                myAvat.nick, myAvat.energy);
        // snprintf prevents writing after end of message buffer
        snprintf(message, MESSAGE_LEN - 1, "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 řetězci, lepší ošetření chyb
- Dostupné také v novém C standardu ISO/IEC 9899:2011
- Microsoftí 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://msdn.microsoft.com/en-us/library/8ef0s5kh%28v=vs.80%29.aspx
  - http://msdn.microsoft.com/en-us/library/wd3wzwts%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, vfscanf_s, vfwscanf_s, vscanf_s, vwscanf_s, vsscanf_s, vswscanf_s
  - fprintf_s, fwprintf_s, printf_s, printf_s, snprintf_s, snwprintf_s, sprintf_s, swprintf_s, vfprintf_s, vfwprintf_s, vprintf_s, vwprintf_s, vsnprintf_s, vsnwprintf_s, vsprintf_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…

Problém: podpora Secure C library

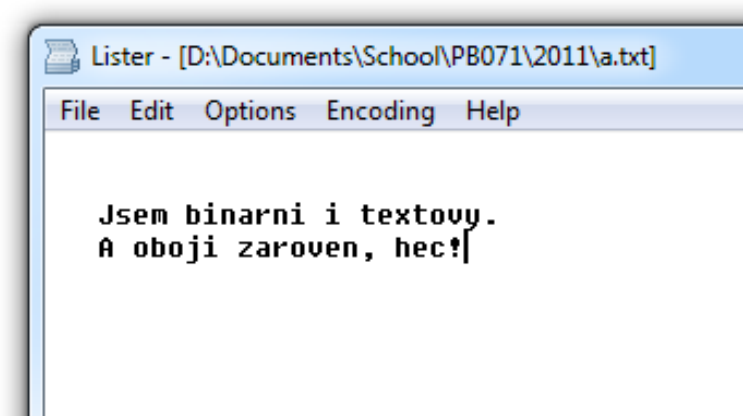
- Standard specifikuje jejich implementaci jen jako volitelnou (optional)
- Některé překladače (např. GCC) neimplementují
  - Kód ve výsledku není přenositelný
- Není jednotný názor na jejich použití
  - Použití těchto funkcí může zamezit dané chybě v konkrétním kontextu (např. zápis za konec pole)
  - Neodstraní ale příčinu (chybějící/chybnou kontrolu argumentů)

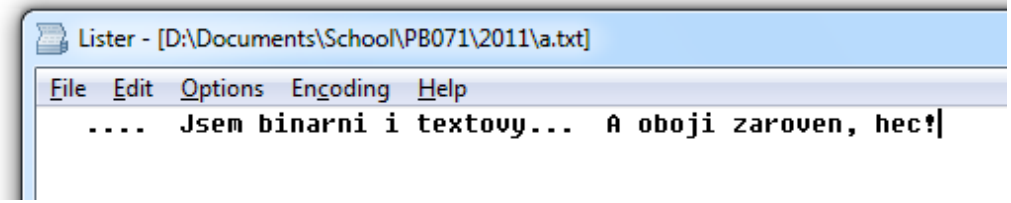
Práce se soubory

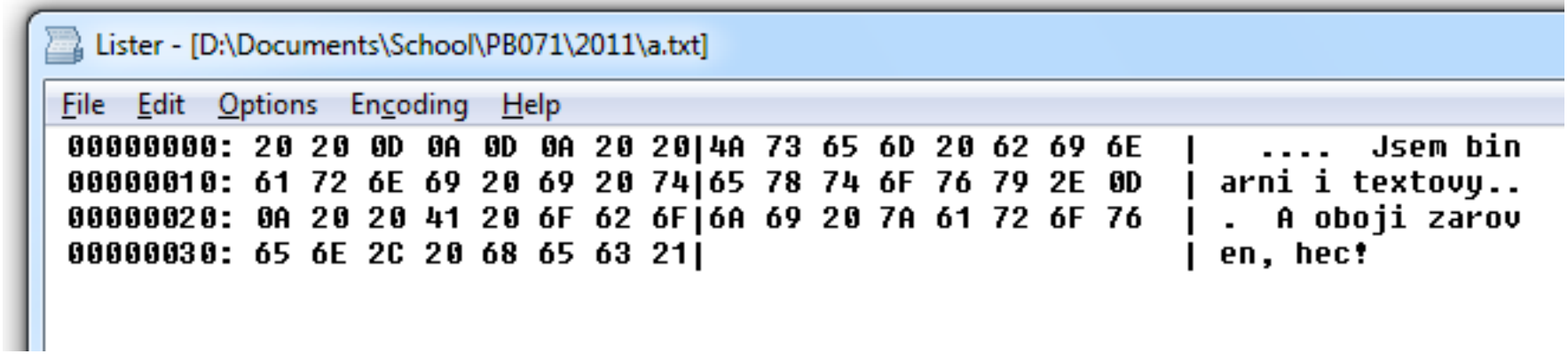
Typy souborů

- 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







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 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");
```

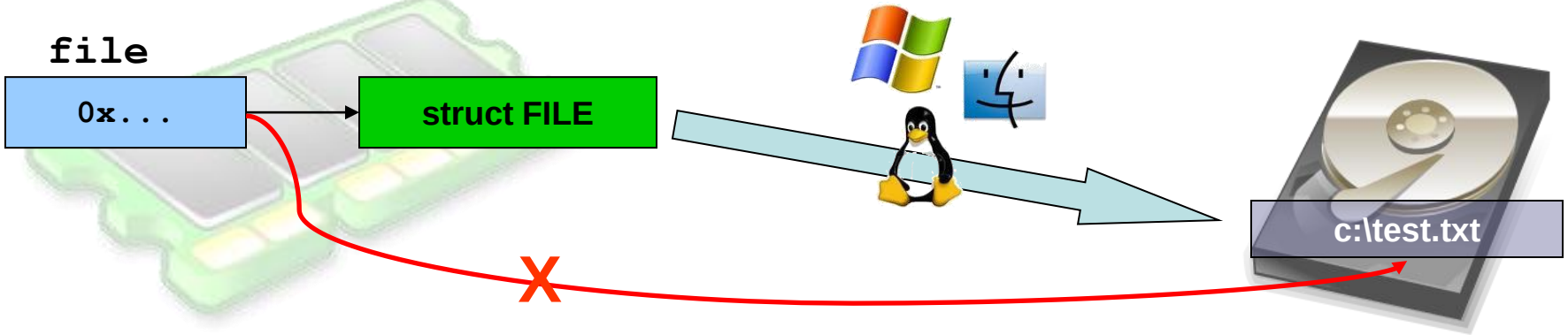
- **`FILE* fopen(const char* filename, const char* mode);`**
- filename obsahuje cestu k souboru
  - relativní: `test.txt, ../test.txt`
  - absolutní: `c:\test.txt`
- Pozor na znak '\' v řetězci obsahující cestu
  - C pokládá \ za speciální znak, nutno použít escape sekvenci \\
  - "`c:\test.txt`" vs. "`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)

Co je vlastně “handle”?

- Handle je reference na nějaký externí prostředek
  - Soubor, otevřené spojení na server nebo databáze…
  - Typicky při interakci s okolím programu (OS, DB, síť…)
  - Používá se, když nelze/nemá význam přímo adresa
- Handle obdržíme po zavolání příslušné funkce
  - Např. `handle = fopen("D:\\test.txt", "r");`
  - Poskytovatel handlu si handle asociuje s prostředkem (socket…)
  - Handle využíváme během následujících voláních, `fgetc(handle);`
- Hodnota „handlu“ je různá a typicky ji v programu nezkoumáme
  - Např. číselný index do tabulky, kde najde OS síťové spojení
- Externí prostředek musíme uvolnit (`fclose(handle);`)
  - Notifikace, že již prostředek nepotřebujeme
  - Např. záznam v tabulce se odstraní a spojení se zavře
- K handlu může být asociován stav, např. přístupová práva

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
  - Standardní knihovna obsah struktury FILE spravuje sama
    - při každém otevření/čtení/zápisu....

LINUX

```
typedef struct  {
        int             level;    /* fill/empty level of buffer */
        unsigned        flags;    /* File status flags          */
        int             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

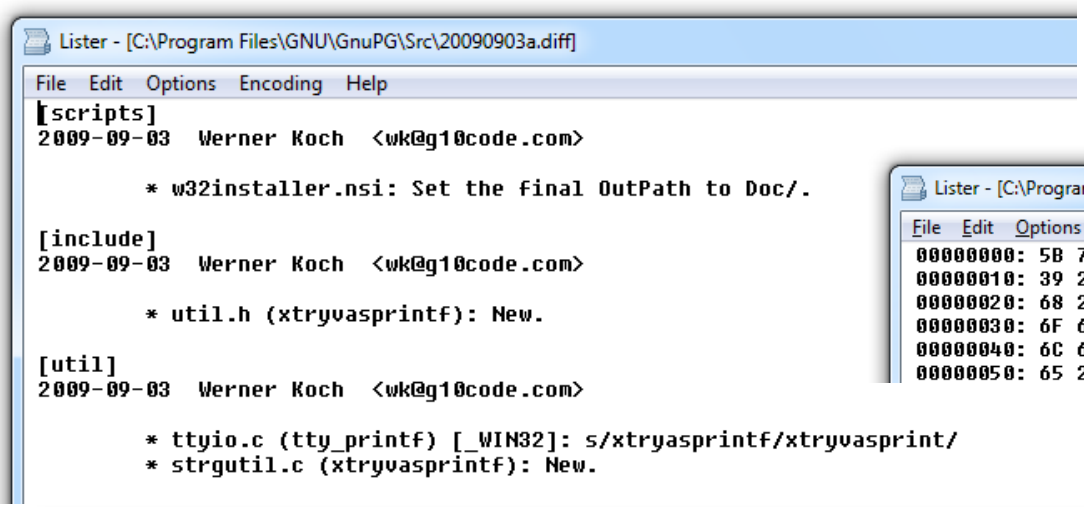
```
typedef struct _iobuf {
        char*   _ptr;
        int     _cnt;
        char*   _base;
        int     _flag;
        int     _file;
        int     _charbuf;
        int     _bufsiz;
        char*   _tmpfname;
} FILE;
```

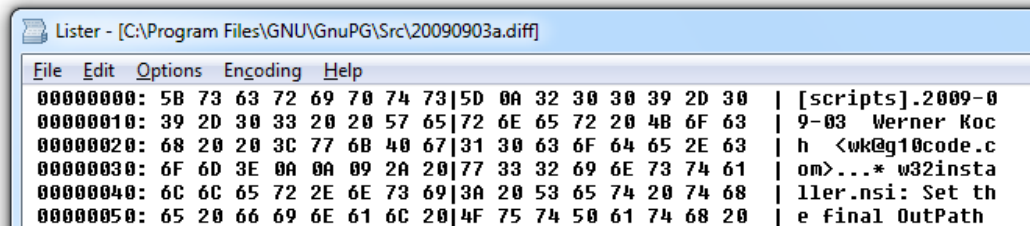
Poznámky k otevření souboru

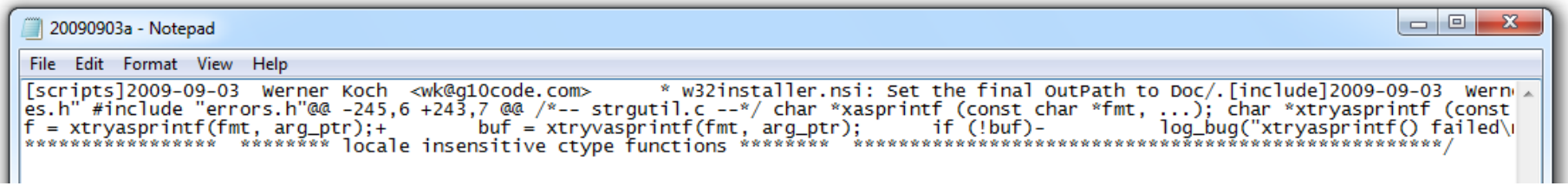
- Defaultně se soubor otevírá jako textový
  - na Unixu je textový i binární mód identický
  - na Windows se nahrazují konce řádků
- Pozor na smazání existujícího souboru
  - fopen("existuje.txt", "w") → existuje.txt velikost 0
- Pozor na situaci, kdy soubor neexistuje
  - fopen("neexistuje.txt", "r")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

Problém s koncem řádku

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







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 );`
  - **POZOR:** funguje jen na běžných souborech, ne na proudech (např. standardním vstupu stdin)

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 soub.
- Při ukončení programu jsou automaticky uzavřeny všechny otevřené soubory
  - Soubory ale přesto zavírejte sami (použití v knihovně…)
- Otevřené soubory nesou režii na straně OS
  - Může dojít k vyčerpání systémových prostředků
  - Již nepoužívané soubory zavírejte
  - Můžete využít detekci Valgrindem (memcheck)

Čtení ze souboru

- Čte se z aktuální pozice v souboru
  - po přečtení se pozice posune těsně za přečtená data
- Načtení jednoho znaku
  - int fgetc (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

```c
#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 fputc (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

```c
#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 pozici na začátek souboru
- **POZOR:** funguje jen na běžných souborech, ne na proudech (např. ne na stdin)

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 (pokud již nějaký s novým jménem existuje, tak se smaže)
- FILE* tmpfile **(void);**
  - otevře dočasný unikátní soubor
  - automaticky zaniká při konci programu

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 ...

Další práce se souborovým systémem

- Jak zjistit jména souborů v adresáři?
- Jak změnit aktuální adresář?
- Jak zjistit atributy souboru (čas, práva)?
- Jak...?

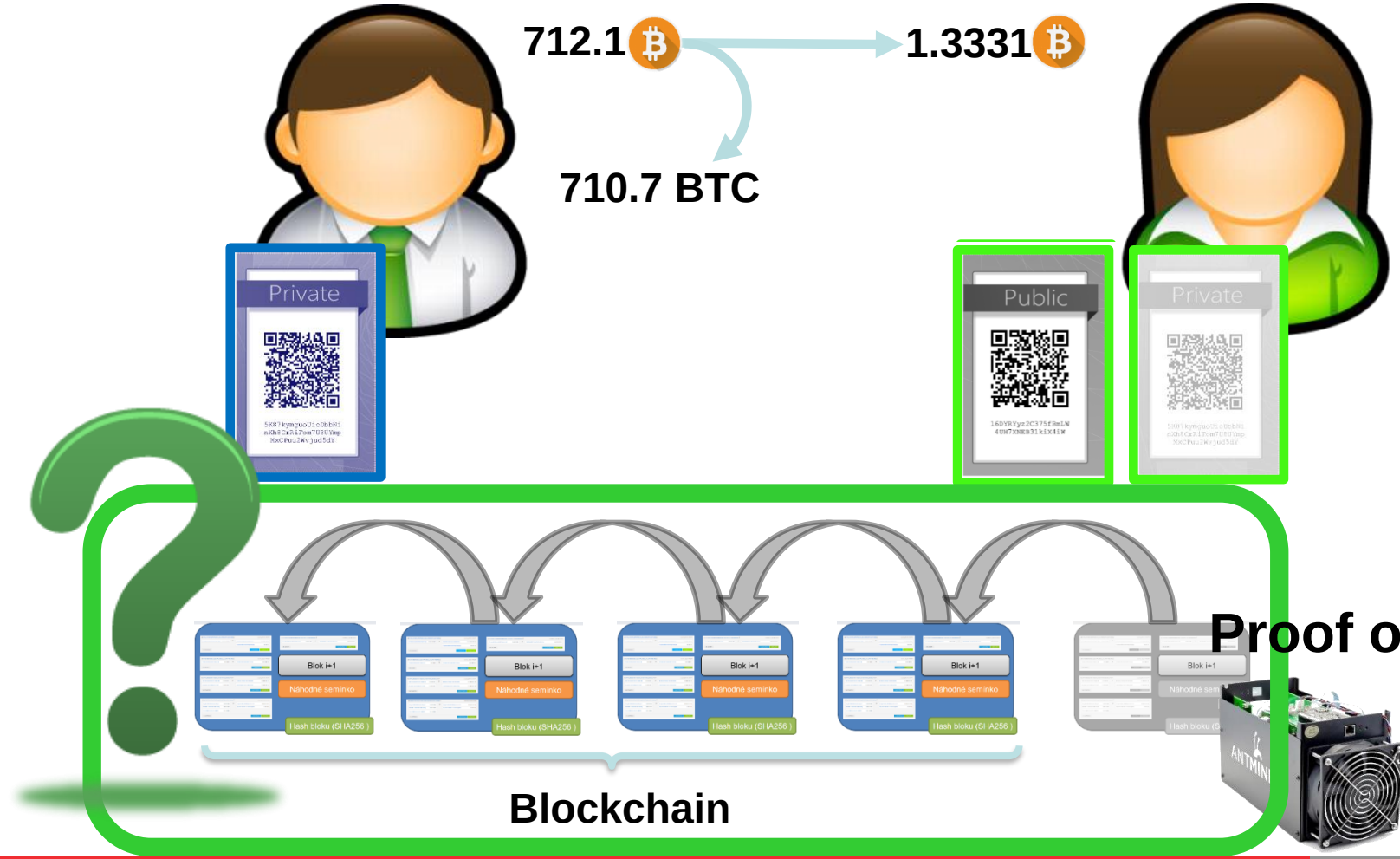
- Funkce nabízené standardní knihovnou C nestačí
- řešením je POSIX - později

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

KDE JE BITCOIN BLOCKCHAIN ULOŽENÝ?


31707ba1a1045044162879d790bd4b4a5ed05cd7ff029dce9f46122467460b83
mined Sep 29, 2017 4:49:20 PM
16FZb6QHFmMoxQfisMBDRvMPKo
712.10757837 BTC
1FnVsCiNE1Du3W4PqYjMHzn8GmR
710.75395577 BTC (U)
17MypmznvzdKBHuuukAdanPngbC
1.331 BTC (U)
FEE: 0.0226226 BTC
1 CONFIRMATIONS
712.08495577 BTC


712.1
1.3331
710.7 BTC
Private
Public
Private
Blok i+1
Náhodné seminko
Hash bloku (SHA256)
Blockchain
Proof of Work

Jak velký je blockchain?

- V průměru každých 10 minut vznikne nový blok
  - Maximálně 1MB blok, po aktivaci SegWit (2017) o něco více (<4MB)
  - Blok může být menší (málo nových transakcí, nečeká se na naplnění)
- Aktuální velikost blockchainu je (1.4.2023):
  - 783459 bloků, 469.38 GB

Blocks

Height	Pool	Timestamp	Mined	Health	Reward	Fees	TXs	Size
783459	Foundry USA	2023-04-01 14:24	9 minutes ago	99.96%	6.48 BTC	0.23 BTC	2,864	1.61 MB
783458	Foundry USA	2023-04-01 14:11	22 minutes ago	96.27%	6.35 BTC	0.10 BTC	970	2.62 MB
783457	Binance Pool	2023-04-01 14:09	24 minutes ago	99.95%	6.43 BTC	0.18 BTC	2,089	2.19 MB
783456	Foundry USA	2023-04-01 14:03	30 minutes ago	100%	6.48 BTC	0.23 BTC	1,217	1.24 MB
783455	AntPool	2023-04-01 13:57	37 minutes ago	93.11%	6.34 BTC	0.09 BTC	1,241	3.24 MB
783454	F2Pool	2023-04-01 13:53	41 minutes ago	97.18%	6.38 BTC	0.13 BTC	2,263	2.1 MB
783453	AntPool	2023-04-01 13:47	47 minutes ago	100%	6.52 BTC	0.27 BTC	3,371	1.57 MB
783452	Poolin	2023-04-01 13:33	1 hour ago	99.39%	6.57 BTC	0.32 BTC	2,979	1.58 MB
783451	AntPool	2023-04-01 13:13	1 hour ago	95.96%	6.42 BTC	0.17 BTC	1,700	2.26 MB
783450	Foundry USA	2023-04-01 13:06	1 hour ago	90.79%	6.33 BTC	0.08 BTC	849	2.91 MB
783449	Foundry USA	2023-04-01 13:04	1 hour ago	94.2%	6.30 BTC	0.05 BTC	1,212	2.11 MB

Kde je blockchain uložený?

- Každý plný Bitcoin uzel se zapojuje do P2P Bitcoin sítě
  - Defacto optimalizovaný bittorrent na blockchain
  - Zároveň jeden z klíčových prvků bezpečnosti (validuje všechny příchozí bloky i transakce)
- Nový uzel se připojí k existujícím a začne si žádat postupně nové bloky
  - Počáteční synchronizace, pak nově vytěžené
  - Zároveň poskytuje ostatním bloky, o které si řeknou
- Každý uzel (full node) obsahuje kopii blockchainu a může ji lokálně analyzovat
- Kde najít soubory s lokální kopií?
  - \Bitcoin\blocks\blk00xxx.dat
  - Začně vzniká automaticky po spuštění Bitcoin klienta (i regtest)

Jak ověřit správnost staženého blokchainu?

- Co když si stáhnu celý blockchain třeba z uloz.to?
  - Může být modifikovaný vůči tomu „originálnímu“ z Bitcoin P2P sítě
  - Každý si může vygenerovat svůj vlastní (regtest)
  - Nebo zkrátit ten originální (chybí provedená transakce)
  - Prodloužit originální o další bloky, kde je útočník jediný miner
  - …
- Jak ověřit správnost staženého blokchainu?
  - Verifikace všech položek od prvního (Genesis) bloku
  - Získávat bloky z náhodně vybraných uzlů z Bitcoin P2P sítě
- Jak získat důvěryhodně první blok?
  - Je „zakompilován“ v Bitcoin aplikaci (např. Bitcoin Core)
  - /src/chainparams.cpp:CreateGenesisBlock()

/src/chainparams.cpp:CreateGenesisBlock()

```
/**
 * Build the genesis block. Note that the output of its generation
 * transaction cannot be spent since it did not originally exist in the
 * database.
 *
 * CBlock(hash=000000000019d6, ver=1, hashPrevBlock=00000000000000, hashMerkleRoot=
 *   CTransaction(hash=4a5e1e, ver=1, vin.size=1, vout.size=1, nLockTime=0)
 *     CTxIn(COutPoint(000000, -1), coinbase 04ffff001d0104455468652054696d65732030
 *     CTxOut(nValue=50.00000000, scriptPubKey=0x5F1DF16B2B704C8A578D0B)
 *   vMerkleTree: 4a5e1e
 */
static CBlock CreateGenesisBlock(uint32_t nTime, uint32_t nNonce, uint32_
{
    const char* pszTimestamp = "The Times 03/Jan/2009 Chancellor on brin
    const CScript genesisOutputScript = CScript() << ParseHex("04678afdb
    return CreateGenesisBlock(pszTimestamp, genesisOutputScript, nTime, 
}
```

Jak vypadá první blok?

- Binární soubor
  - FILE* genesis = fopen("blk00000.dat", "rb");

Lister - [c:\Bitcoin\blocks\blk00000.dat]

File Edit Options Encoding Help

```
00000000: F9 BE B4 D9 1D 01 00 00|01 00 00 00 00 00 00 00  | ù¾´Ù............
00000010: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
00000020: 00 00 00 00 00 00 00 00|00 00 00 00 3B A3 ED FD  | ............;£íý
00000030: 7A 7B 12 B2 7A C7 2C 3E|67 76 8F 61 7F C8 1B C3  | z{.²zÇ,>gv.a.È.Ã
00000040: 88 8A 51 32 3A 9F B8 AA|4B 1E 5E 4A 29 AB 5F 49  | ..Q2:.¸ªK.^J)«_I
00000050: FF FF 00 1D 1D AC 2B 7C|01 01 00 00 00 01 00 00  | ÿÿ...¬+|........
00000060: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
00000070: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 FF FF  | ..............ÿÿ
00000080: FF FF 4D 04 FF FF 00 1D|01 04 45 54 68 65 20 54  | ÿÿM.ÿÿ....EThe T
00000090: 69 6D 65 73 20 30 33 2F|4A 61 6E 2F 32 30 30 39  | imes 03/Jan/2009
000000A0: 20 43 68 61 6E 63 65 6C|6C 6F 72 20 6F 6E 20 62  |  Chancellor on b
000000B0: 72 69 6E 6B 20 6F 66 20|73 65 63 6F 6E 64 20 62  | rink of second b
000000C0: 61 69 6C 6F 75 74 20 66|6F 72 20 62 61 6E 6B 73  | ailout for banks
000000D0: FF FF FF FF 01 00 F2 05|2A 01 00 00 00 43 41 04  | ÿÿÿÿ..ò.*....CA.
000000E0: 67 8A FD B0 FE 55 48 27|19 67 F1 A6 71 30 B7 10  | g.ý°þUH'.gñ¦q0·.
000000F0: 5C D6 A8 28 E0 39 09 A6|79 62 E0 EA 1F 61 DE B6  | \Ö¨(à9.¦ybàê.aÞ¶
00000100: 49 F6 BC 3F 4C EF 38 C4|F3 55 04 E5 1E C1 12 DE  | Iö¼?Lï8ÄóU.å.Á.Þ
00000110: 5C 38 4D F7 BA 0B 8D 57|8A 4C 70 2B 6B F1 1D 5F  | \8M÷º..W.Lp+kñ._
00000120: AC 00 00 00 00 F9 BE B4|D9 D7 00 00 00 01 00 00  | ¬....ù¾´Ù×......
00000130: 00 6F E2 8C 0A B6 F1 B3|72 C1 A6 A2 46 AE 63 F7  | .oâ..¶ñ³rÁ¦¢F®c÷
```

DÍKY ZA VÁŠ ČAS A V PONDĚLÍ ZASE NA VIDĚNOU