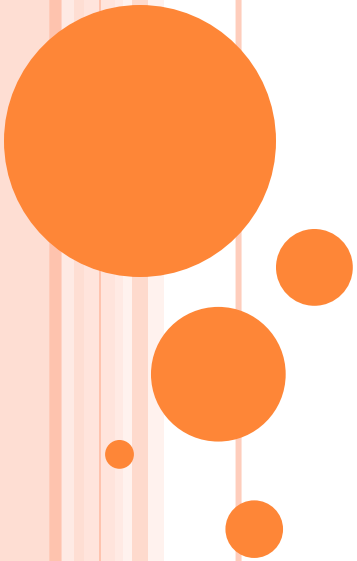


Programovacie jazyky pre vstavané systémy

Cvičenie 9

Náplň cvičenia

1. Hlavičkový súbor <assert.h>.
2. Build-systém.
3. Nekorektná práca s pamäťou.
4. Smerníky na funkcie.
5. Bitové operácie.
6. Úlohy.

Hlavičkový súbor <assert.h>

- Hlavičkový súbor <assert.h> definuje dve makrá, pomocou ktorých je možné testovať platnosť tvrdení:
  - assert(podmienka)
    - makro pracuje, len ak nie je definované makro NDEBUG pred #include<assert.h>,
    - ak má podmienka hodnotu 0, tak makro spôsobí **okamžité ukončenie programu** (cez abort() – abnormálne ukončenie aplikácie) s vypísaním informácie o mieste, kde došlo k chybe,
    - typické použitie – kontrola, či smerník, ktorý chceme dereferencovať neukazuje na NULL,
    - http://en.cppreference.com/w/c/error/assert,
  - static_assert(podmienka, sprava) (C11)
    - makro zodpovedá kľúčovému slovu _Static_assert (C11),
    - ak má podmienka hodnotu 0, tak makro spôsobí **chybu pri kompilácii**, pričom sa vypíše sprava,
    - typické použitie – kontrola, čí je kód prenositeľný na cieľovú platformu, t.j., či cieľová platforma spĺňa požiadavky, ktoré sú potrebné pre správnu činnosť programu (napr. 64 bitová architektúra, int má aspoň 4 bajty,...),
    - http://en.cppreference.com/w/c/language/_Static_assert.

Build-systém – make

- make – nástroj pre spúšťanie špeciálneho skriptu (tzv. Makefile), pomocou ktorých je možné pomerne ľahko udržiavať závislosti medzi súbormi vo väčších projektoch. Nástroj automaticky deteguje, ktoré súbory boli zmenené a na základe definovaných závislostí vykoná potrebné operácie (kompilácia a linkovanie).
- Návody na tvorbu Makefile:
  - http://kifri.fri.uniza.sk/~chochlik/frios/frios/sk/cvicenia/procesy/make.html
  - http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
- Ukážka Makefile pre synchrónnu klient-server aplikáciu z moodla:

```
1  OUTPUTS = k_s_server k_s_client
2  CC = gcc
3
4  all: $(OUTPUTS)
5
6  clean:
7          rm -f $(OUTPUTS)
8          rm -f *.o
9
10 .PHONY: all clean
11
12 %.o: %.c k_s_definitions.h
13         $(CC) -c -o $@ $<
14
15 k_s_server: k_s_definitions.o k_s_server.o
16         gcc $^ -o $@
17
18 k_s_client: k_s_definitions.o k_s_client.o
19         gcc $^ -o $@
20
```

Nekorektná práca s pamäťou

- V jazyku C je za správnu prácu s pamäťou zodpovedný programátor. Pri nekorektnej práci môžu vznikať ťažko detekovateľné chyby, ktoré sa niekedy prejavia a inokedy nie.
- K objaveniu takýchto chýb je možné využiť viaceré nástroje:
  - valgrind – pokročilý nástroj pre objavovanie problémov pri práci s pamäťou; len pre Unix-like systémy (http://valgrind.org/)
  - CMemLeak veľmi jednoduchý nástroj pre kontrolu správneho uvoľnenia všetkej alokovanej pamäte (http://www.codeguru.com/cpp/misc/misc/memory/article.php/c3745/Detecting-Memory-Leaks-in-C.htm)

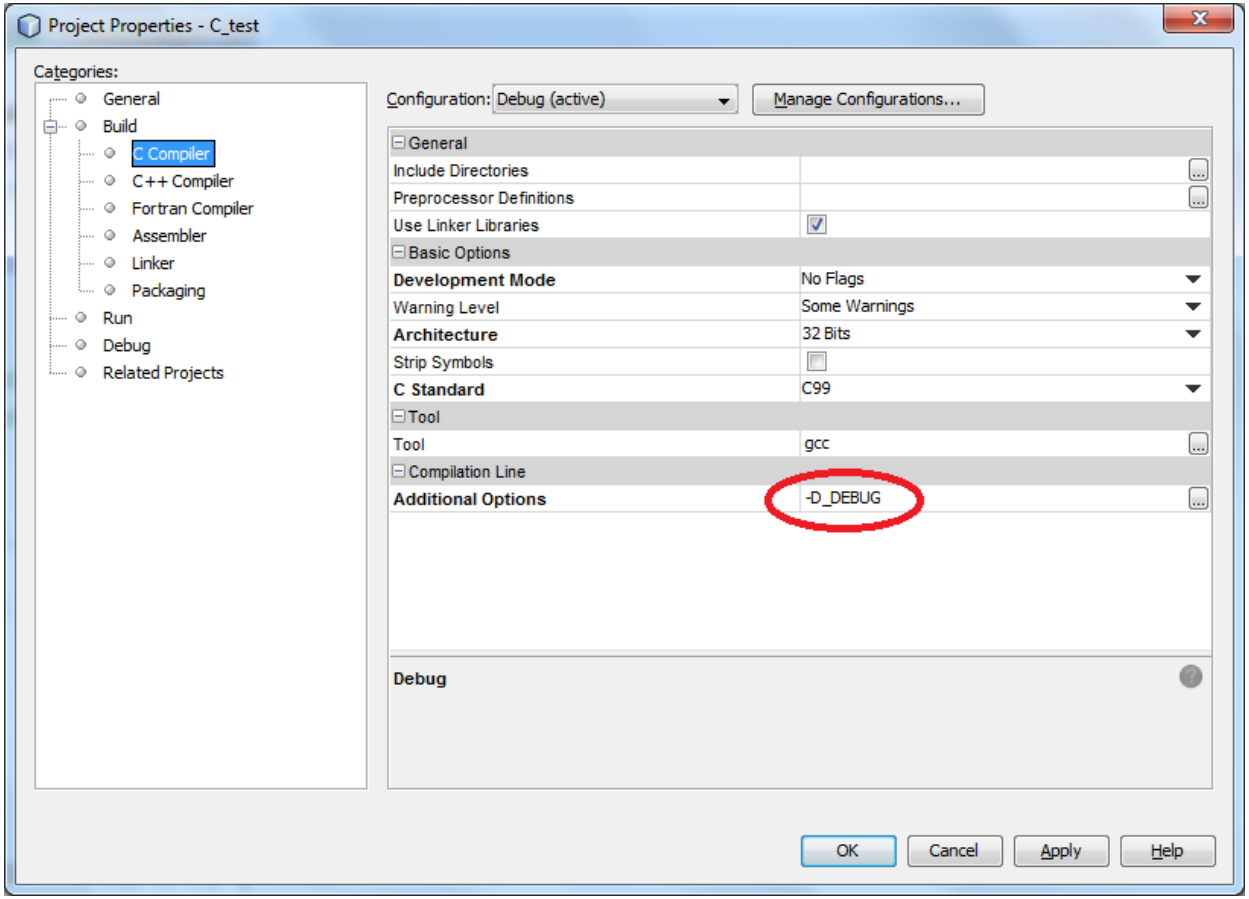
CMemLeak (1)

1. Do projektu pridajte nasledujúce súbory:
   - CMemLeak.h
   - CMemLeak.c
2. Do každého .c súboru pridajte (v takomto poradí):
   - #include <stdio.h>
   - #include "CMemLeak.h"

```
1  #include <stdlib.h>
2  #include "CMemLeak.h"
3
4  int main(int argc, char** argv) {
5      int* pole = malloc(10*sizeof(int));
6      free(&pole);
7
8      return 0;
9  }
```

CMemLeak (2)

3. Knižnicu CMemLeak aktivujete nastavením makra _DEBUG



CMemLeak (3)

4. Po korektnom skončení aplikácie vznikne súbor CMemLeak.txt. Popis možných chýb nájdete na http://www.codeguru.com/cpp/misc/misc/memory/article.php/c3745/Detecting-Memory-Leaks-in-C.htm

```
1  FNH: &pole deallocated ukazky_9.c: 6
2
3  Final Report
4  MLK: 0x80028cd0 40 bytes allocated ukazky_9.c: 5
5  Total allocations     : 1
6  Max memory allocation: 40 (0K)
7  Total leak            : 40
8
```

Ukazovateľ na funkciu

- Ukazovateľ nemusí uchovávať len adresu objektu, ale môže uchovávať aj adresu funkcie. Napr.:

```
int min(int a, int b) {
    return a < b ? a : b;
}

int (*f1)(int a, int b) = min;
int (*f2)(int, int) = &min;
int (*f3)(int, int);
f3 = min;

int vys1 = f1(10, 20);
int vys2 = (*f1)(10, 20);
int vys3 = (***f1)(10, 20);

typedef int (*FUN)(int a, int b);
FUN f4 = min;
```

Bitové operácie

- Pomocou bitových operátorov môžeme manipulovať s bitmi v **celočíselných** dátových typoch.
- Bitové operácie sa používajú na tvorbu bitových masiek, pomocou ktorých je možné získať alebo nastaviť hodnotu konkrétneho bitu nejakej l-hodnoty.
- Pri použití bitových operátorov sa na operandy aplikujú implicitné konverzie typu „usual arithemtic conversions“.
- Určte typ nasledujúcich výrazov:
  - ~1
  - ~(char)1
  - (char)~1

Operátor	Význam	Príklad	Výsledok
~	bitová negácia	~0000 1111	1111 0000
<<	bitový posun doľava	1110 0011 << 2	1000 1100
>>	bitový posun doprava	1110 0011 >> 2	0011 1000
&	bitový súčin	1110 0011 & 0000 1111	0000 0011
^	bitový xor	1110 0011 ^ 0000 1111	1110 1100
|	bitový súčet	1110 0011 | 0000 1111	1110 1111

Úlohy – smerníky na funkcie

- Vytvorte program pre utriedenie poľa:
  - objektov typu int,
  - objektov typu double,
  - reťazcov.

  Triedenie realizujte pomocou funkcie qsort(), ktorej popis nájdete na http://en.cppreference.com/w/c/algorithm/qsort.

- K štruktúre LinZoz (cvičenie 6) prirobte funkcie:
  - void process(const struct LinZoz *linZoz, void (*processItem)(void *item, void *data), void *data) – funkcia dostane ako parameter smerník na funkciu, pomocou ktorej bude možné spracovať jednu položku zoznamu (ak má funkcia niečo vypočítať, vráti to cez parameter data);
  - implementujte niekoľko funkcií, ktoré budú môcť byť poslané do funkcie process(), napr.:
    - void printItem(void *item, void *data);
    - void setItem(void *item, void *data);
    - void sum(void *item, void *data);
    - void product(void *item, void *data);
    - ...

Úlohy – množina

- Pomocou bitových operácií implementujte množinu celých čísel, ktorá môže obsahovať celé čísla z intervalu <0, 255> (tzv. bázová množina).
- Pri definícii dátového typu množina využite celočíselné dátové typy s **pevnou veľkosťou** (C99). Dátové typy sú definované v hlavičkovom súbore <stdint.h> (http://en.cppreference.com/w/c/types/integer):
  - uint8_t, uint16_t, uint32_t, uint64_t,...
- Nad množinou implementujte nasledujúce operácie:
  - void insert(struct set *A, unsigned char value) – vloží prvok s hodnotou value do množiny A;
  - void remove(struct set *A, unsigned char value) – odstráni prvok s hodnotou value z množiny A;
  - bool contains(const struct set *A, unsigned char value) – zistí, či sa prvok s hodnotou value nachádza v množine A;
  - void print(const struct set *A) – vypíše obsah množiny A na štandardný výstup;
  - struct set* intersection(const struct set *A, const struct set *B, struct set *C) – množinu C naplní tak, aby predstavovala prienik množín A a B a vráti ukazovateľ na ňu;
  - struct set* union(const struct set *A, const struct set *B, struct set *C) – množinu C naplní tak, aby predstavovala zjednotenie množín A a B a vráti ukazovateľ na ňu;
  - struct set* difference(const struct set *A, const struct set *B, struct set *C) – množinu C naplní tak, aby predstavovala rozdiel množín A a B a vráti ukazovateľ na ňu;
  - struct set* complement(const struct set *A, struct set *C) – množinu C naplní tak, aby predstavovala doplnok množiny A do bázovej množiny a vráti ukazovateľ na ňu;
  - struct set* complement(const struct set *A, struct set *C) – množinu C naplní tak, aby predstavovala doplnok množiny A do bázovej množiny a vráti ukazovateľ na ňu.