Škola pre Mimoriadne Nadané Deti a Gymnázium, Teplická 7, 831 02 Bratislava

Mgr. Anino BELAN

GRAFIKA V JAZYKU C

(knižnica Allegro)

učebný text pre kvartu a kvintu osemročného gymnázia

BRATISLAVA
2003

Obsah

Úvod

Práve ste sa začítali do pokračovania kurzu jazyka C. Toto pokračovanie je venované knižnici Allegro. Knižnica Allegro je určená na programovanie hier. Na jej vytváraní sa podieľa asi 200 ľudí z celého sveta a jej funkcie vám dávajú možnosť pristupovať ku grafike, využívať vo vašich programoch zvukovú kartu, pracovať s myšou, klávesnicou a joystickom a programovať, čo sa vám páči.

Táto knižka nadväzuje na Kurz jazyka C. Bolo by dobré, aby ste skôr, než sa do nej pustíte, zvládli prvý diel, alebo sa naučili základy jazyka C nejako inak.

Prajem príjemnú zábavu.

Anino

1. lekcia
Už zase začiatok alebo "Hello, world!"

Jednou z najsilnejších čŕt jazyka C je jeho univerzálnosť. Môžete ho použiť pri tvorbe počítačovej hry, zložitého databázového systému, programu na modelovanie procesov v jadrovom reaktore alebo textového procesora. Nech programujeme čokoľvek, stále používame základné štruktúry C-čka ako napríklad `if`, `while`, `for` alebo `case`.

Je ale zrejmé, že na písanie rôznych druhov programov je treba mať k dispozícii rôzne prostriedky. Keď programujete internetový server, budete používať úplne iné funkcie, než keď programujete hru. A práve na to slúžia jazyku C knižnice. Knižnice obsahujú funkcie z jednotlivých oblastí a programátor môže použiť tie, ktoré potrebuje.

Knižníc pre prácu s grafikou je viacero. Niektoré sú jednoduchšie, niektoré zložitejšie, niektoré sú určené pre konkrétny operačný systém, niektoré sú schopné pracovať pod rôznymi systémami. V nasledujúcich lekciách sa budeme zaoberať knižnicou **Allegro**. Vybrali sme ju preto, lebo je jednoduchá (áno, v porovnaní s inými je skutočne jednoduchá), lebo môže fungovať pod všetkými operačnými a podobnými systémami, ktoré v škole máme (Linux, DOS, Windows) a pretože sa smie používať zadarmo.

Oproti doterajším zvyklostiam sme pristúpili k jednej zmene: budeme číslovať riadky v programe. Slúži to len na orientáciu v programe (programy budú dlhšie, než doteraz), tak to, prosím, nepíšte do zdrojákov.

Takže prvý grafický program (nazvite si ho `hello.c`):

```
 1  #include <allegro.h>
 2
 3  int main()
 4  {
 5     allegro_init();
 6     install_keyboard();
 7
 8     set_color_depth(16);
 9     if (set_gfx_mode(GFX_AUTODETECT_FULLSCREEN, 640, 480, 0, 0) != 0) {
10        set_gfx_mode(GFX_TEXT, 0, 0, 0, 0);
11        allegro_message("Zlyhala grafika\n%s\n", allegro_error);
12        return 1;
13     }
14
15     clear_to_color(screen, makecol(255, 255, 255));
16
17     text_mode(-1);
18     textout_centre(screen, font, "Hello, world!",
19              SCREEN_W/2, SCREEN_H/2, makecol(0,0,0));
20
21     readkey();
22     return 0;
23  }
24
25  END_OF_MAIN();
```

V prvom riadku načítame hlavičkový súbor knižnice allegro. Táto sa momentálne stará o všetko. Nemusíte načítavať nič ďalšie. Na začiatku `main`u v riadku 5 sa zavolá funkcia `allegro_init`. Táto funkcia zapína allegro a treba ju mať hneď na začiatku. Táto funkcia zistí typ operačného systému a nastaví niektoré základné premenné. Hneď po nej nasleduje funkcia `install_keyboard`, ktorá zapne klávesnicu.

A ideme naštartovať grafický režim. V riadku 8 sa nastavuje farebná hĺbka. Programu povieme, koľko farieb naraz budeme zobrazovať. Ak je farebná hĺbka 8, môžeme naraz zobraziť 256 farieb. Ak je farebná hĺbka 16, môžeme ich zobraziť až 65536. Farebnú hĺbku môžeme nastaviť aj na 24 alebo 32, ak to naša grafická karta zvládne.

V riadku 9 sa volá funkcia `set_gfx_mode`, ktorá štartuje grafiku. Má 5 parametrov. Prvý hovorí o spôsobe, ako sa má grafika naštartovať. `GFX_AUTODETECT_FULLSCREEN` znamená, že sa má na zobrazenie použiť celá obrazovka. Ďalšie dva hovoria o rozlíšení (v tomto prípade 640×480 bodov). Posledné dva parametre zatiaľ pre nás nie sú dôležité.

Ak sa funkcii `set_gfx_mode` podarí grafiku naštartovať, vráti ako výsledok nulu. Preto sa na riadku 9 kontroluje, či je výsledok naozaj 0 a ak nie je, v 10. riadku sa nastaví (opäť s pomocou funkcie `set_gfx_mode`) textový režim a na 11. riadku funkcia `allegro_message`, ktorá funguje presne rovnako ako funkcia `printf`, vypíše správu, že niečo nie je v poriadku. Potom na 12. riadku program skončí.

Na riadku 15 sa volá funkcia `clear_to_color`. Táto funkcia vie daný obrázok vyplniť danou farbou. Ako obrázok je zadaná `screen`, čiže celá obrazovka. Ako farba je zadaná farba `makecol(255,255,255)`. Farby sa určujú v RGB (red, green, blue) podobe. Každá farba je zmixovaná z týchto troch. Koľko je ktorej, určujú čísla od 0 do 255. Keď všetky tri hodnoty vypeckujeme na maximum, dostaneme najjasnejšiu možnú farbu – bielu. Funkcia `makecol` nám z tých troch čísel vyrobí tú správnu farbu pre zadanú farebnú hĺbku.

Keďže chceme vypisovať text, na riadku 17 nastavíme spôsob vypisovania. Ak funkcia `text_mode` dostane ako parameter farbu, nastaví ju ako farbu pozadia textu. Ak sa ale ako parameter zadá záporné číslo, pozadie bude priesvitné. Ostane bezo zmeny a vykresľovať sa budú iba písmenká.

Funkcia `textout_centre` vypíše do daného obrázku text. Má veľa parametrov. Prvý parameter hovorí, do ktorého obrázku sa má písať (znova je to `screen` – obrazovka). Druhý hovorí, akým fontom (typom písma) sa to má spraviť. (premenná `font` označuje štandardný systémový font.) Tretí parameter je text, ktorý sa má vypísať. (Klasické "Hello, world!") Štvrtý a piaty parameter určujú súradnice, kam sa má text vypísať. Použili sa premenné `SCREEN_W` a `SCREEN_H` v ktorých je šírka (width) a výška (height) obrazovky. Aby sa text vypísal do stredu obrazovky, z každého sa zobrala polovica. Posledný šiesty parameter je farba textu.

Na 21. riadku funkcia `readkey` čaká, kým niekto stlačí nejakú klávesu. Ak by sme ju tam nedali, program by hneď po vykreslení skončil a veľa by sme toho nevideli. (Špeciálne kvôli tejto funkcii sme na riadku 6 rozbehávali klávesnicu.) Na riadku 22 program skončí.

Na konci, až po skončení funkcie `main` je jeden podivný riadok – riadok číslo 25. Ten tam niektoré operačné systémy (napr. Windows ale v niektorých prípadoch aj Linux) potrebujú. Zatiaľ to berte ako zaklínadlo a píšte ho vždy tesne po funkcii `main`.

Na záver ešte pár slov o tom, ako to treba skompilovať. Všetky programy, s ktorými ste sa doteraz stretli, zatiaľ okrem štandardnej knižnice `libc` nepoužívali nič špeciálne. Tento program ale používa knižnicu `allegro`. Ako funkcie z tejto knižnice vyzerajú, sa kompilátor dozvie zo súboru `allegro.h`, ktorý ste includli v programe. Ale samotné funkcie sa nachádzajú v súbore `liballeg.dll`, `liballeg.a` alebo `liballeg.so`. Aby boli správne pripojené, treba zadať linkeru zvláštny parameter `-lalleg`. (Podobu parametra odvodíte tak, že od názvu knižnice odtrhnete koncovku a to „lib“ na začiatku a miesto neho dáte „l“.) Takže ak sa zdroják volal

`hello.c`, kompilovať sa bude (v prípade DOSu) príkazom

```
gcc -o hello.exe hello.c -lalleg
```

a v prípade linuxu (kde je knižnica `allegro` rozdelená na dve časti) príkazom

```
gcc -o hello hello.c -lalleg -lalleg_unsharable
```

Úloha č.1: Napíšte, pochopte, skompilujte.

Úloha č.2: Skúste na riadku 9 dať štartový režim `GFX_AUTODETECT_WINDOWED`. Čo sa zmenilo? Vyskúšajte farebné hĺbky 8 a 24 (riadok 8) a rozlíšenia 320×200, 800×600 a 1024×768. Kedy to funguje a kedy nie?

Úloha č.3: Skúste zmeniť farby na riadkoch 15, 17 a 19. Upravte program tak, aby na čiernom pozadí vypísal žltý text v modrom rámčeku.

Úloha č.4: Upravte program tak, aby sa text vypísal na vrchu obrazovky. Upravte program tak, aby sa text vypísal na spodku obrazovky.

2. lekcia
Čiaročky
alebo "Koho si Jáááánošíííik na paloš načiara..."

V minulej lekcii sme sa naučili, ako naštartovať grafický režim a ako vypísať text na miesto, ktoré si vyberiete. Teraz to bude o tom, ako sa dajú kresliť čiary, čiarky a čiaročky.

Grafický režim sa štartuje príkazom

```
set_gfx_mode(GFX_AUTODETECT_FULLSCREEN, 640, 480, 0, 0)
```

Čísla 640 a 480 znamenajú šírku a výšku obrazovky v bodoch. Súčasne s týmto príkazom sa na obrazovke vytvorí súradnicová sústava. Má ale jednu zvláštnosť: bod [0,0] sa nachádza vľavo hore a smerom nadol súradnica `y` rastie. Teda bod, ktorý sa nachádza na obrazovke vľavo dole má súradnice `[0,479]`. (Všimnite si, že druhá súradnica je o 1 menšia, než 480!!!) Bod vpravo hore má súradnice `[639,0]` a aké má súradnice bod vpravo dole, necháme na úvahu láskavého čitateľa. Mimochodom – aké súradnice má bod v strede obrazovky?

Na kreslenie úsečky slúži funkcia `line`. Ak chcem napríklad nakresliť na obrazovke zelenú úsečku z bodu `[10,40]` do bodu `[70,20]`, spravím to príkazom

```
line(screen, 10, 40, 70, 20, makecol(0, 255, 0));
```

Prvý parameter je obrázok, do ktorého budem kresliť (v našom prípade obrazovka). Ďalšie dva parametre určujú súradnice prvého bodu. Nasledujúce dva parametre určujú súradnice druhého bodu a posledný parameter je farba.

Rafinovanejšie použitie tohto príkazu si môžete vyskúšať na nasledujúcom príklade:

```
 1  #include <allegro.h>
 2
 3  int main()
 4  {
 5     int i;
 6
 7     allegro_init();
 8     install_keyboard();
 9
10     set_color_depth(16);
11     if (set_gfx_mode(GFX_AUTODETECT_FULLSCREEN, 640, 480, 0, 0) != 0) {
12        set_gfx_mode(GFX_TEXT, 0, 0, 0, 0);
13        allegro_message("Zlyhala grafika\n%s\n", allegro_error);
14        return 1;
15     }
16
17     clear_to_color(screen, makecol(0, 0, 0));
18
19     for (i = 0; i <= 100; i++)
20        line(screen, 100, 0, 500, 4*i,
21                        makecol(255 - 2*i, 255 - 2*i, 255));
22
23     readkey();
24     return 0;
25  }
26
27  END_OF_MAIN();
```

Riadky 1 až 15 sú až na malú výnimku rovnaké ako riadky prvého programu, ktorý pracoval s knižnicou allegro, ktorý sme vyrábali minule. Tou výnimkou je deklarácia premennej `i` typu `int`, ktorú budeme potrebovať. Na riadku 17 zafarbíme celú obrazovku na čierno.

Na riadku 19 začína cyklus `for`, ktorý začína s hodnotou premennej `i` nula, zakaždým ju zväčší o 1 a beží až dovtedy, kým `i` neprerastie hodnotu 100. (Koľkokrát cyklus prebehne?) V každom behu cyklu sa zavolá funkcia `line`, ktorá nakreslí úsečku z bodu `[100,0]` do bodu `[500,4 × i]`. Keď má `i` hodnotu 0, ide o bod `[500,0]`, pri hodnote 1 je to bod `[500,4]` atď. až pri hodnote 100 sa nakreslí úsečka do bodu `[500,400]`. Takto v jednom cykle nakreslíme tých úsečiek pomerne dosť.

Premennú `i` použijeme ešte na inú vec. S jej pomocou budeme meniť farbu. Ak je `i` rovné nule, farba je `makecol(255,255,255)`, čo je biela. Hodnoty červenej a zelenej však v každom cykle zmenšíme a posledná úsečka sa už bude kresliť farbou `makecol(55,55,255)`, čo je kúsok svetlejšia modrá. A ten pomalý prechod od bielej k modrej je pekný. Však môžete si vyskúšať. Riadky 21 až 25 sú opäť rovnaké ako v prvom programe.
Iný zaujímavý obrázok dostanete ak cyklus na riadkoch 19 až 21 zameníte za cyklus

```
for(i = 0; i <= 400; i++)
   line(screen, 100, i, 500, 400-i, makecol(55+i/2,0,0) );
```

Prečo sa zmenil tvar?

Úlohy na dnes:

Úloha č.1: Napíšte, vyskúšajte a pochopte príklady.

Úloha č.2: Upravte prvý príklad tak, aby sa farba nemenila na modrú, ale na žltú.

Úloha č.3: Nakreslite domček so šikmou strechou.

Úloha č.4: Nakreslite mriežku 8×8 štvorčekov, každý so stranou 10.

3. lekcia

Omaľovánky
alebo "Toto prosím na zeleno."

Skôr, než začneme s niečím novým, by som rád urobil istý dohovor. Totiž v programoch, ktoré sme doteraz používali, sa niektoré veci opakujú a zaberajú miesto na papieri. Preto chcem zverejniť štandardnú verziu funkcie `main`. Budeme ju používať až do odvolania a bude vyzerať takto:

```
int main()
{
    /* Deklaracie */

    allegro_init();
    install_keyboard();
    set_color_depth(16);
    if (set_gfx_mode(GFX_AUTODETECT_FULLSCREEN, 640, 480, 0, 0) != 0)
    {
        set_gfx_mode(GFX_TEXT, 0, 0, 0, 0);
        allegro_message("Zlyhala grafika\n%s\n", allegro_error);
            return 1;
    }

    /* Hlavny kod */

    readkey();
    return 0;
}
END_OF_MAIN();
```

V ďalšom texte bude uvedené už iba aké premenné budeme používať (vloží sa to pod poznámku `Deklaracie`) a ako má vyzerať hlavný kód (vloží sa to pod poznámku `Hlavny kod`).

Vypĺňať plochu s pomocou čiar je možné (ako ste sa mohli presvedčiť v predošlej lekcii). Ak to však nerobíte pozorne, môže sa vám stať, že niektoré bodíky na obrazovke ujdú vašej pozornosti a nevyplníte ich. Môže z toho síce vzísť pekná vzorka (ako sa stalo pri prvom príklade predošlej lekcie), ale to nie je vždy to, čo si človek želá. Preto má Allegro funkcie určené špeciálne na vypĺňanie určených plôch.

Najjednoduchšie je vyplniť obdĺžnik. Napríklad obdĺžnik, ktorého protiľahlé vrcholy majú súradnice `[30,70]` a `[160,140]` sa vyplní príkazom

```
rectfill(screen, 30, 70, 160, 140, makecol(255, 198, 198));
```

Ako parametre uvedieme obrázok, do ktorého ideme čmárať (teda obrazovku), súradnice vrcholov a farbu. Podobne, ak chceme vyplniť trojuholník s vrcholmi `[130,30]`, `[420,70]` a `[270,342]`, môžeme zavolať funkciu

```
triangle(screen, 130, 30, 420, 70, 270, 342, makecol(255, 198, 198));
```

Situácia je trochu zložitejšia, keď chceme vyplniť viacuholník. Vtedy treba najprv vrcholy uložiť do nejakého poľa premenných typu `int`. Ak chceme vyplniť päťuholník s vrcholmi `[320,20]`, `[420,460]`, `[100,140]`, `[540,140]` a `[220,460]`, tak deklarácia bude vyzerať takto:

```
int bodiky[10] = { 320,  20, 420, 460, 100, 140,
                   540, 140, 220, 460};
```

V hlavnom kóde bude potom príkaz

```
polygon(screen, 5, bodiky, makecol(255, 0, 0));
```

Funkcii treba povedať, do ktorého obrázku ideme kresliť, koľko bodov ten mnohouholník má, v ktorom poli sú uložené a akou farbou sa má kresliť. Pole musí byť aspoň dvakrát také dlhé, ako počet vrcholov, pretože na každý vrchol potrebujeme dve súradnice. Ak hranica mnohouholníka pretína sama seba, vypĺňanie sa správa podivne – ako sa koniec koncov môžete presvedčiť na uvedenom príklade.

Posledný z vypĺňacích príkazov, ktoré budú v tejto lekcii zverejnené je príkaz `floodfill`. Slúži na vypĺňanie ohraničenej oblasti danou farbou. (Ak ste vo windowsovskom maľovaní použili niekedy tlačítko `vyplň`, určite viete o čom je reč.) Hlavný kód môže vyzerať napríklad takto:

```
1   line(screen, 290, 210, 350, 210, makecol(0, 255, 0));
2   line(screen, 290, 210, 290, 270, makecol(0, 255, 0));
3   line(screen, 290, 270, 350, 270, makecol(0, 255, 0));
4   line(screen, 350, 210, 350, 270, makecol(0, 255, 0));
5   line(screen, 260, 180, 380, 180, makecol(0, 255, 0));
6   line(screen, 260, 180, 260, 300, makecol(0, 255, 0));
7   line(screen, 260, 300, 380, 300, makecol(0, 255, 0));
8   line(screen, 380, 180, 380, 300, makecol(0, 255, 0));
9   floodfill(screen, 320, 200, makecol(0, 128, 0));
```

Na riadkoch 1 až 8 nakreslíme dva štvorce. Bod [320,200] leží medzi nimi, takže funkcia `floodfill` vyplní celý priestor medzi nimi trochu tmavšou zelenou.

Kto by mal záujem o ďalšie vypĺňacie funkcie, nájde ich v manuáli k Allegru v kapitole Drawing primitives.

Úloha č.1: Vyskúšajte uvedené príklady, posledný zmeňte tak, že vnútro menšieho štvorca vyplníte oranžovou a okolie vonkajšieho fialovou farbou.

Úloha č.2: Nakreslite červenú hviezdu bez diery v prostriedku.

Úloha č.3: Náhodné číslo[1] typu `int` vyrobíte s pomocou funkcie `random()`, náhodné číslo od 0 do 479 sa dá spraviť `random() % 480`. Nakreslite 1000 náhodných trojuholníkov náhodnej farby. (Ak by kompilátor hlásil, že nepozná funkciu `random()`, pridajte na začiatok súboru `#include <stdlib.h>`)

Úloha č.4: Nakreslite šachovnicu aj s bielymi a čiernymi poľami.

1 Čísla generované funkciou `random()` sa síce ako náhodné tvária, ale pri každom spustení programu budú rovnaké. Ak tomu chcete zabrániť, musíte nastaviť „semienko náhody“ (random seed) na inú než štandardnú hodnotu. Bežne sa používa aktuálny systémový čas. Takže kdesi k začiatku programu treba napísať príkaz

```
srandom(time(NULL));
```

Funkcia `time` je deklarovaná v hlavičkovom súbore `time.h`, takže ho nezabudnite tiež includovať, ak budete túto funkciu chcieť použiť.

4. lekcia
Obrázky
alebo "Monu Lisu vľavo hore prosím."

Doteraz sme pracovali s jediným obrázkom – s obrazovkou. Odvolávali sme sa na ňu s pomocou systémovej premennej `screen`. Táto premenná mala typ `BITMAP*` čiže smerník na bitmapu (bitmapa je len krycí názov pre obrázok). Dnes sa dozvieme, ako pracovať aj s inými obrázkami a ako obsah jedného dostať do druhého.

Sú dva druhy obrázkov. Tie, ktoré vidieť (obrazovka alebo jej časti) a tie, ktoré nevidieť. Zdalo by sa, že obrázky, ktoré nevidieť sú naprosto zbytočné. Je to prekvapivé, ale takéto obrázky (zvané pamäťové bitmapy) sa používajú veľmi často. Prvý dôvod je ten, že práca s obrazovkou je pomalá a práca s pamäťou rýchla. Takže často využívaný trik je, že sa nejaké zložitejšie kreslenie vykoná v pamäti a keď je to hotové, tak sa to potom celé skopíruje na obrazovku. Ďalšie použitie je, keď treba nejaký obrázok natiahnuť zo súboru. Taký obrázok sa odloží do pamäťovej bitmapy a potom sa môže kopírovať na obrazovku.
Na prvú ukážku je nutné, aby ste mali vo svojom počítači súbor `xicht.tga` v ktorom je uložený obrázok s rozmermi 100×100 bodov. Do deklarácie pridajte riadok

```
BITMAP* obrazok;
```

Hlavný kód bude obsahovať nasledujúce riadky:

```
1   obrazok = load_bitmap("xicht.tga",NULL);
2   blit(obrazok,screen, 0,0, 270,190, 100,100);
```

Prvý riadok obsahuje funkciu `load_bitmap`, ktorá má toho na starosti veľa. Musí prečítať súbor, ktorý sa jej povie (rozumie obrázkom typu BMP, LBM, PCX a TGA), vytvoriť pre neho miesto v pamäti, obrázok do toho miesta skopírovať a vrátiť adresu, kde sa nachádza. Táto adresa sa potom vloží do premennej `obrazok`. Funkcia má dva parametre. Prvý je reťazec s menom súboru, druhý sa používa iba pri 8bitovej grafike (vtedy je to smerník na paletu, ktorá sa má použiť), inak má hodnotu `NULL`.

V druhom riadku je použitá funkcia `blit`, ktorá vie kopírovať kus jedného obrázka do druhého. Prvý parameter je obrázok, z ktorého sa kopíruje, druhý je obrázok do ktorého sa kopíruje. Ďalšie dve čísla označujú súradnice bodu v zdrojovom obrázku, z ktorého kopírovanie začne. (Ak chceme kopírovať celý obrázok, musí to byť vždy [0,0].) Ďalšie dve čísla označujú miesto v cieľovom obrázku, kam chceme kopírovať. V určenom bode bude ľavý horný roh nášho veľdiela. Posledná dvojica čísel určuje šírku a výšku toho, čo sa bude kopírovať. (Ak chceme skopírovať celý obrázok s rozmermi 100×100 bodov, musí tam byť [100,100].)

Program by bol skompilovateľný a spustiteľný. Ale akonáhle vytvoríme pamäťovú bitmapu, preberáme za ňu zodpovednosť. To znamená, že keď ju už nebudeme potrebovať, musíme ju z pamäte vymazať. Preto tesne pred skončením (až po záverečnom `readkey`) musíme zavolať funkciu `destroy_bitmap`:

```
destroy_bitmap(obrazok);
```

Úloha č.1: Upravte uvedený program tak, aby sa kópia obrázku `xicht.tga` objavila v každom rohu obrazovky. Upravte ho tak, aby ste z obrázku skopírovali do stredu obrazovky len jeho ľavú hornú a pravú dolnú štvrtinu.

Obrázok je možné nielen kopírovať, ale aj naťahovať. Slúži na to funkcia `stretch_blit`. Nahraďte v predošlej ukážke druhý riadok nasledujúcim:

```
stretch_blit(obrazok,screen, 0,0,100,100, 170,190,300,100);
```

Parametre sú podobné ako pri funkcii `blit`. Najprv treba uviesť odkiaľ a kam sa kopíruje. Potom sa označí bod, od ktorého sa začne v zdrojovom obrázku a šírka a výška plochy, ktorá sa bude kopírovať (v našom prípade bod [0,0], šírka aj výška 100). Potom sa určí bod šírka a výška v cieľovom obrázku. (Cieľová šírka je v našom prípade 300 a celý obrázok sa teda trikrát natiahne do šírky.)

Úloha č.2: Natiahnite obrázok `xicht.tga` na celú plochu obrazovky.

Úloha č.3: Vydláždite obrázkom `xicht.tga` obrazovku.

Úloha č.4: Nakreslite tisíckrát náhodne umiestnený a natiahnutý obrázok `xicht.tga`.

Úloha č.5: Ak funkcia `load_bitmap` zlyhá (napríklad preto, lebo tam súbor s obrázkom nie je, alebo je súbor poškodený), vráti hodnotu NULL. Upravte program tak, aby v prípade, že obrázok nenačíta, správne skončil a podal o chybe správu.

5. lekcia
Škriatkovia
alebo "Obrázky s dierami."

Určite ste niekedy hrali niektorú z klasických 2D počítačových hier ako napríklad Super Mario alebo Prince of Persia. Skrátka hry, v ktorých pobehujú nejaké postavy, povaľujú sa nejaké predmety a hlavného hrdinu treba niekam bez ujmy na zdraví dopraviť. Tiež si možno pamätáte na korytnačky z Imaginu, ktoré boli miestami priesvitné a ktoré mohli pobehovať po pracovnej ploche bez toho, aby ju poškodili. V oboch týchto prípadoch (a v mnohých ďalších) prichádzajú ku slovu **sprity** (čítaj sprajty).

Sprite (po anglicky škriatok alebo strašidlo) je obrázok, ktorý nemusí mať nutne obdĺžnikový tvar a ktorý môžete vykresliť kdesi na obrazovku. Sprity, ktoré používa Allegro sú obyčajné bitmapy, na ktorých vykreslenie sa použila špeciálna funkcia `draw_sprite`. Miesta, ktoré na obrázku majú jasnofialovú farbu (`makecol(255,0,255)`) sa pokladajú za diery a pri kreslení sa vynechávajú.

Na nasledujúcom príklade si ukážeme, ako sa dá s pomocou spritov spraviť jednoduchá animácia. Okrem programu budete potrebovať päť obrázkov sln1.tga, sln2.tga, sln3.tga, sln4.tga a sln5.tga (všetky majú rozmer 100×100 bodov). Prečítajte si pozorne program a najmä komentár, ktorý po ňom nasleduje.

Takže deklarácie budú vyzerať takto:

```
 1   BITMAP *obrazok[5];
 2   BITMAP *kuspozadia, *vysledok;
 3   int i;
```

a hlavný kód takto:

```
 4   obrazok[0] = load_bitmap("sln1.tga",NULL);
 5   obrazok[1] = load_bitmap("sln2.tga",NULL);
 6   obrazok[2] = load_bitmap("sln3.tga",NULL);
 7   obrazok[3] = load_bitmap("sln4.tga",NULL);
 8   obrazok[4] = load_bitmap("sln5.tga",NULL);
 9   kuspozadia = create_bitmap(100,100);
10   vysledok = create_bitmap(100,100);
11
12   for( i = 0; i < 320; i++)
13           line(screen, 0, i, 640, i, makecol(128 + i/5, 128 + i/5, 255));
14   for( i = 320; i < 640; i++)
15           line(screen, 0, i, 640, i, makecol(0, 255 + 320 - i, 0));
16   blit(screen, kuspozadia, 400, 100, 0, 0, 100, 100);
17   i = 0;
18   while( ! keypressed())
19   {
20         blit(kuspozadia, vysledok, 0, 0, 0, 0, 100, 100);
21         draw_sprite(vysledok, obrazok[i], 0, 0);
22         blit(vysledok, screen, 0, 0, 400, 100, 100, 100);
23         i = (i + 1) % 5;
24         rest(100);
25   }
26
27   for(i = 0; i < 5; i++)
28         destroy_bitmap(obrazok[i]);
29   destroy_bitmap(kuspozadia);
30   destroy_bitmap(vysledok);
```

V deklaráciach máme pole piatich smerníkov na bitmapy `obrazok[0]` až `obrazok[4]` do ktorých uložíme sprity zo súborov, smerníky na dve pomocné bitmapy `kuspozadia` a `vysledok` a jednu pomocnú celočíselnú premennú `i`.

V hlavnom kóde najprv načítame jednotlivé bitmapy (riadky 4–8). Bitmapy `kuspozadia` a `vysledok` musia byť pred použitím vytvorené funkciou `create_bitmap`. V riadkoch 12–13 nakreslíme oblohu (vodorovné čiary, prechod z bledomodrej do eštebledšomodrej) a v riadkoch 15–16 trávnik (prechod od zelenej do tmavozelenej).

Keď sme si takto vytvorili pozadie, môžeme sa venovať spritom. Päť spritov, ktoré máte k dispozícii vytvára animáciu slniečka. Keby sme ich však na obrazovku plácali jeden cez druhý, tie staré by nezmizli a bol by z toho chaos. Preto to treba spraviť šikovnejšie. Na riadku 16 si do bitmapy `kuspozadia` uložím kus pozadia vpravo dole od bodu `[400,100]`. Kreslenie potom bude prebiehať tak, že si `kuspozadia` skopírujeme do bitmay `vysledok`, pridáme tam patričný sprite a celý výsledok skopírujeme naspäť na obrazovku.

Hlavný cyklus je na riadkoch 18–25. Beží až kým niekto nestlačí nejakú klávesu (zisťuje mi to funkcia `keypressed()`). Na riadku 20 skopírujeme `kuspozadia` do `vysledok`. Na riadku 21 nakreslíme do bitmapy `vysledok` sprite `obrazok[i]` (`i` sa meni od 0 do 4). Funkcii `draw_sprite` treba povedať do ktorého obrázku sa ide kresliť, aký sprite a na ktoré miesto. Na riadku 22 skopírujeme výsledok na správne miesto na obrazovke. Keď je všetko hotové, zväčšíme `i` o 1 a osekneme modulo 5, aby po štyrke nasledovala nula a aby sme nabudúce kreslili ďalší sprajt v poradí (riadok 23) a počkáme 100 milisekúnd (funkcia `rest`) aby to nebežalo príliš rýchlo (riadok 24).

Akonáhle niekto stlačí klávesu, cyklus skončí. Na riadkoch 27–30 zlikvidujeme bitmapy a program môže skončiť tiež.

Úloha č.1: Napíšte, skompilujte, pochopte.

Úloha č.2: Ako to pobeží, ak vynecháte riadok 24? Čo to spraví, ak budete výsledok v riadku 22 kopírovať na pozíciu `[400,350]` miesto na `[400,100]`?

Úloha č.3: Prerobte program tak, že sa budú sprity miesto tej obchádzky cez dve bitmapy vykresľovať priamo na obrazovku. Ako to bude vyzerať?

Úloha č.4: Vykreslite na obrazovku 40 náhodne umiestnených slniečok zo súboru `sln1.tga`. Dávajte si pozor, aby ste ich neumiestnili na trávnik.

6. lekcia
Jemný úvod do myšovania alebo "Myšie strašidlo"

V tejto lekcii sa dozviete niečo o práci s myšou. Nedozviete sa ale ani zďaleka všetko. Napríklad taký detail, ako urobiť, aby bolo polohu myši na obrazovke aj vidieť si necháme až na budúcu lekciu. Takže dnes síce budeme pracovať s myšou, ale bude neviditeľná.

Predtým, než sa k myši dostaneme, treba však povedať pár slov o bitových maskách. Predstavte si, že si potrebujete zapamätať jednu logickú hodnotu. Vyhradíte si na ňu pravdepodobne premennú typu `int`. Ak je v nej 0, logická hodnota je `nepravda`, ak je v nej hocičo okrem nuly, logická hodnota je `pravda`. Ak si chcete zapamätať jednu logickú hodnotu, tak je to v poriadku. Ale keby ste si ich chceli zapamätať osem, je takýto prístup mrhaním. Dá sa to spraviť šikovnejšie. A pomôže nám k tomu dvojková sústava.

Napríklad číslo 75 sa v dvojkovej sústave napíše ako `01001011`. Každá jeho číslica sa dá použiť na zapamätanie jednej logickej hodnoty 1 - pravda, 0 - nepravda. To by bolo fajn, ale aby sa s tým pohodlne robilo, bolo by treba nájsť spôsob, ktorým sa dá jednoducho zistiť, aká je tretia cifra zprava a spôsob ako rýchlo vyrobiť číslo, ktoré má prvú, druhú a piatu cifru jedničky a ostatné nuly.

Na prvú z týchto vecí sa používa operátor `&`. Funguje podobne ako operátor `&&` v logických podmienkach (výsledok je pravda len ak sú pravdivé oba vstupy) iba že to spraví pre jednotlivé bity vstupných čísel. Takže napríklad `01101010 & 11010110` bude `01000010` pretože iba na druhej a siedmej cifre zľava majú obe čísla jednotky. Ako teda zistiť, či je v premennej `a` na tretej cifre zľava jednotka? Jednoducho. Ak tam jednotka nie je, tak výraz `(a & 4)` bude nula (pretože 4 v dvojkovej sústave je `00000100`). Ak tam jednotka je, výraz bude nenulový.[2]

Takže naspäť ku knižnici `allegro`. Ak chceme používať myš, treba jej podporu zapnúť. Teda pod riadok v ktorom sa inštaluje klávesnica treba pridať riadok

```
install_mouse();
```

Od toho okamihu máte k dispozícii premenné `mouse_x` a `mouse_y`, v ktorých budete mať uloženú aktuálnu pozíciu myši. Okrem toho je tu ešte premenná `mouse_b` v ktorej je ako v bitovom poli uložené, ktoré tlačidlo na myši je stlačené. Ak je nenenulové `(mouse_b & 1)`, je stlačené ľavé tlačidlo. Ak `(mouse_b & 2)`, je stlačené pravé tlačidlo a ak je nenulové `(mouse_b & 4)`, je stlačené stredné tlačidlo (ak ho myš má).

S týmto vybavením už môžeme urobiť prvý nesmelý pokus o jednoduché kreslítko. Do hlavného kódu dajte nasledujúci kus programu:

```
while( ! keypressed())
{
   if (mouse_b & 1)
      putpixel(screen, mouse_x, mouse_y, makecol(255,255,0));
}
```

2 Na vytváranie bitových polí sa používa operátor `|`. Ak chceme číslo, ktoré má jednotky na prvej, druhej a štvrtej cifre zľava, môžeme ho vyrobiť ako výraz `( 8 | 2 | 1 )`. (Samozrejme, môžeme si to zrátať aj ručne a rovno tam napísať 11.)

Ten robí iba to, že kým nestlačíte niečo na klávesnici, opakovane zisťuje, či je stlačené ľavé tlačidlo na myši a ak áno, tak tam, kde je myš, spraví žltú bodku.

Úloha č.1: Vyskúšajte, pochopte.

Úloha č.2: Upravte program tak, aby keď je stlačené ľavé tlačidlo kreslil žlté bodky a keď je stlačené pravé tlačidlo kreslil červené bodky.

Úloha č.3: Upravte program tak, aby pri stlačení ľavého tlačidla kreslil súvislú čiaru. Urobte to tak, aby sa čiara prerušila, keď ľavé tlačidlo pustíte. (Pozor, je to pomerne náročné. Dobre si premyslite, čo idete urobiť a ako to bude fungovať.)

Úloha č.4: Upravte program tak, aby tam, kde kliknete, nakreslil slniečko (sprite `sln1.tga`).

7. lekcia
Pokračovanie s myšou alebo "Hlodavec, nezavadzaj!"

V predošlej lekcii sme sa naučili, ako pracovať s myšou. Vieme zistiť jej pozíciu aj to, či sú na nej stlačené niektoré tlačidlá. Ale stále to malo jednu nevýhodu. Myš nebolo vidieť. Ona klasická malá šipka, na ktorú sme zvyknutí, po obrazovke nebehala a pozíciu myši sme mohli iba hádať.

Ak chceme, aby myš bolo vidieť, musíme urobiť dve veci. Za prvé musíme nainštalovať modul `timer`. Na čo je taký modul dobrý sa dozvieme v niektorej z ďalších lekcií. Zatiaľ len toľko, že ak chceme vidieť myš, musí byť nainštalovaný. Spravíme to tak, že tam, kde sme inštalovali ostatné veci pridáme riadok `install_timer();`[3]

Druhá vec ktorú musíme dodržať je, že vždy predtým, než na obrazovku budeme niečo kresliť, musíme myš zhasnúť, pretože inak nám bude robiť na obrazovku šmuhy. Hneď ako dokreslíme, samozrejme myši povieme, aby sa opäť objavila. Myš sa objavuje na obrazovke príkazom `show_mouse(screen);` a zhasína sa príkazom `show_mouse(NULL);` Na ukážku spravíme program, ktorý bude „razítkovať“ slniečka, rovnaký ako v predošlej lekcii, ale s viditeľnou myšou. V deklaráciach bude

```
1   BITMAP* obrazocek;
```

a v hlavnom kóde

```
2   obrazocek = load_bitmap("sln1.tga",NULL);
3   show_mouse(screen);
4
5   while( ! keypressed())
6   {
7      if (mouse_b & 1)
8      {
9         show_mouse(NULL);
10        draw_sprite(screen, obrazocek, mouse_x-50, mouse_y-50);
11        show_mouse(screen);
12     }
13  }
```

V cykle `while` sa kontroluje, či je stlačené ľavé tlačidlo na myši a ak áno, myš sa skryje, nakreslí sa slniečko a myš sa znova zapne. Celý tento cyklus sa opakuje až kým niekto nestlačí klávesu na klávesnici (a výraz `(! keypressed())` tak nadobudne hodnotu „nepravda“). Mimochodom – prečo sú vo funkcii `draw_sprite` súradnice pozície myši zmenšené o päťdesiat?

Iste ste už pracovali s viacerými programami, v ktorých sa kurzor myši rôzne menil. Aj v allegre môžete dať myši tvar ľubovoľného spritu, ktorý si vyrobíte. Slúži na to funkcia `set_mouse_sprite`. V prípade, že kurzor myši zmeníte, je dôležité určiť, ktorý jeho bod je aktívny. (Aktívny bod je ten, ktorého pozícia sa ukladá do premenných `mouse_x` a `mouse_y`. Pri klasickom tvare myši je to špička šipky.) Aktívny bod sa určuje s pomocou funkcie

3 Keď som to doma na linuxe skúšal, fungovalo to aj bez timera. Je ale možné, že na niektorých iných platformách je to nutnosť.

`set_mouse_sprite_focus`. Ako parametre sa použijú súradnice aktívneho bodu na myšovom sprite. Ak túto funkciu nepoužijete, za aktívny sa pokladá ľavý horný roh myšového spritu – aktívny bod je nastavený na `[0,0]`.

Ak teda chceme, aby nám po obrazovke miesto klasickej šipky pobehovalo slniečko a odtlačilo sa tam, kde klikneme, musíme medzi riadky 2 a 3 pridať nasledujúci kód:

```
set_mouse_sprite(obrazocek);
set_mouse_sprite_focus(50,50);
```

Úloha č.1: Vyskúšajte a pochopte obe ukážky (s obyčajnou myšou aj so slniečkom).

Úloha č.2: Napíšte program (s viditeľnou myšou), ktorý na mieste kliknutia nakreslí kružnicu s polomerom 30.

(Na kreslenie kružníc sa používa funkcia `circle(bitmapa,x,y,polomer,farba)`.)

Úloha č.3: Spravte program s viditeľnou myšou, ktorý v prípade, že je stlačené ľavé tlačidlo na myši, nakreslí úsečku z bodu `[0,0]` do bodu, kde sa práve myš nachádza. Skúste meniť farbu úsečky podľa pozície myši.

Úloha č.4: (ťažká) Upravte program so slniečkom z úlohy č.1 tak, aby na obrazovke zostal vždy len posledný odtlačok slniečka.

8. lekcia

Opakovanie grafiky alebo "Picasso by sa divil."

Úloha č.1: Nakreslite snehuliaka

Úloha č.2: Nakreslite kráľovskú korunu.

Úloha č.3: Nakreslite 10 štvorcov s rozmermi 100×100 bodov na náhodných miestach vyplnených náhodnými farbami.

Úloha č.4: Nakreslite 15 schodov.

Úloha č.5: Urobte program, v ktorom sa na začiatku vľavo na obrazovke objaví kruh a potom sa pomaly presunie doprava (treba ho vždy zmazať a o kúsok ďalej znovu nakresliť).

Úloha č.6: Nakreslite lienku. (Túto úlohu vymyslela moja žena, takže je za dva body.)

Úloha č.7: Nakreslite obrázok `xicht.tga` zväčšený na 200×200 bodov presne do stredu obrazovky a okolo neho spravte zelený rámik hrubý 10 bodov.

Úloha č.8: Vytvorte pamäťovú bitmapu 20×20 bodov, do nej nakreslite červený trojuholník. Keď je sltačené tlačítko myši, odtláčajte tento trojuholník na obrazovku. Myš by malo byť vidieť.

Úloha č.9: Vytvorte pamäťovú bitmapu 20×20 bodov, vyplňte správnou farbou (aby to bol sprit) a nakreslite do nej zelený trojuholník. Túto bitmapu nastavte ako kurzor myši s aktívnym bodom v jednom z vrcholov trojuholníka.

Úloha č.10: Nakreslite 15 štvorcov so spoločným stredom a so stranami 10, 20,..., 150

Úloha č.11: V predošlej úlohe vyplňte priestor medzi štvorcami náhodnými farbami.

Úloha č.12: Vyplňte celú plochu obrazovky štvorčekmi 5×5 bodov, ktoré budú mať náhodné farby.

Úloha č.13: Spravte program znázorňujúci výbuch školy.

9. lekcia
Timery
alebo "Časovaná nálož"

Počítače majú rôznu rýchlosť. Historický exemplár XTčka je asi tisíckrát pomalší, než Pentium III. Vývoj ide dopredu a ľudia sa z toho väčšinou tešia. Však kto by sa netešil, keď mu program behá tisíckrát rýchlejšie. Problémy ale môžu nastať, ak ten program je napríklad Tetris alebo iná počítačová hra. Ak si niekto svoju zamilovanú verziu Tetrisu pre XTčko pustí na nejakom rýchlom stroji, môže iba s hrôzou sledovať, ako sa mu na najpomalšom leveli zaplní v priebehu pol sekundy celý stĺpec bez toho, aby vôbec stihol siahnuť na klávesnicu.

Aby sa tento problém vyriešil, boli vymyslené timery. Sú to hodiny, ktoré má program možnosť sledovať a správať sa podľa nich. Pri ich použití pod niektorými operačnými systémami však treba dodržať niektoré pomerne náročné pravidlá. V nasledujúcej ukážke vôbec neinicializujeme grafiku, používame iba funkcie knižnice `allegro` týkajúce sa timerov:

```
 1   #include <allegro.h>
 2
 3   volatile int sekundy = 0;
 4
 5   void dalsia()
 6   {
 7       sekundy++;
 8   }
 9   END_OF_FUNCTION(dalsia);
10
11
12   int main()
13   {
14      allegro_init();
15      install_keyboard();
16      install_timer();
17
18      LOCK_VARIABLE(sekundy);
19      LOCK_FUNCTION(dalsia);
20
21      install_int(dalsia,1000);
22
23      while(sekundy <= 10)
24          printf("%d s.\n",sekundy);
25
26      return 0;
27   }
28   END_OF_MAIN();
```

S prvou zaujímavosťou sa stretávame už na riadku 3. Premenná `sekundy` má v svojej deklarácii uvedené slovo `volatile`. Toto slovíčko znamená, že premenná môže nečakane a náhle zmeniť hodnotu zásahom „zvonka“. (Všimnite si, že ak by premenná `sekundy` túto vlastnosť nemala, cyklus na riadkoch 23–24 by nikdy nemohol skončiť!) To, že nejaká premenná bude mať

túto vlastnosť, treba dať kompilátoru vedieť.[4]

Na riadkoch 5–9 sa nachádza funkcia `dalsia`. Jej jedinou úlohou je zvýšiť hodnotu premennej `sekundy`.

Riadky 14-16 sú klasické allegrovské inicializačné rutiny. Volanie funkcie `install_timer` nám umožní používať timery a veci s nimi zviazané.

Celé tajomstvo timerov sa nachádza na riadku 21. Funkcia `install_int` nám taký timer vyrobí. Ako prvý parameter treba uviesť funkciu, ktorú timer bude volať (v našom prípade je to funkcia `dalsia`). Táto funkcia nesmie mať parametre ani vracať hodnotu. Ako druhý parameter sa uvedie časový interval v milisekundách, v ktorom sa uvedená funkcia bude volať. Na riadku 21 teda hovoríme nášmu programu, že okrem toho, že vykonáva naše príkazy, tak „pomimo“ ešte každú sekundu (sekunda je 1000 milisekúnd) musí spustiť funkciu `dalsia`.

Na riadkoch 23–24 sa dokolečka vypisuje obsah premennej `sekundy`. Nebyť timera, hodnota premennej `sekundy` je stále 0 a cyklus sa nikdy neskončí. Lenže náš timer (bez ohľadu na to, že program práve vykonáva cyklus) každú sekundu zavolá funkciu `dalsia`, ktorá hodnotu premennej `sekundy` zväčší. Takže stačí počkať 11 sekúnd a cyklus a s ním celý program skončí.

Vyzerá to pekne. Lenže v niektorých prípadoch nastávajú problémy. Niektoré prostredia (napríklad DJGPP pod DOSom) sú pomerne prefíkané a ak majú málo pamäte, tak kus z nej si dočasne odložia na disk (tomu sa hovorí memory swap). A niektoré prostredia (opäť napríklad DJGPP pod DOSom) používajú k obsluhe timerov systémové prerušenia. No a ak sa obsluha systémového prerušenia odswapuje na disk, tak v niektorých prostrediach (príklad už ani uvádzať nebudem...) to spôsobí, že systém v hrozných kŕčoch zdochne. Takže sa treba postarať o to, aby sa obsluha prerušenia ani žiadna premenná, ktorú používa neodswapovala – treba ich zamknúť. Preto na riadkoch 18–19 voláme makrá `LOCK_VARIABLE` a `LOCK_FUNCTION`, ktoré patričnú premennú aj funkciu zamknú a na riadku 9 makro `END_OF_FUNCTION`, ktoré označuje, kde zamykaná funkcia končí.

Z dôvodu problémov zo zamykaním (a nielen z tohto) sa odporúča, aby funkcia obsluhujúca timer bola čo najmenšia a najjednoduchšia.

Úloha č.1: Pochopte, napíšte, skompilujte a vyskúšajte uvedený príklad.

Úloha č.2: Napíšte program stopky, ktorý vám s presnosťou na stotinu sekundy odmeria čas medzi dvoma sltačeniami klávesy `<Enter>`.

Úloha č.3: Napíšte program, ktorý na obrazovku raz za sekundu nakreslí krúžok náhodnej farby, umiestnenia a polomeru. Krúžok nekreslite vo funkcii obsluhujúcej timer ale kontrolujte si, či sa hodnota premennej, v ktorej máte uložený čas od posledného kreslenia, zmenila!

4 Ak by kompilátor túto informáciu nedostal, mohol by sa pokúsiť optimalizovať kód kvôli zrýchleniu a pritom narobiť škodu. Napríklad by mohol zistiť, že v tele cyklu na riadkoch 23–24 sa premenná `sekundy` nemení a preto by kvôli zrýchleniu testoval podmienku cyklu iba na začiatku.

10. lekcia

Dátové súbory alebo "Organizácia skladu"

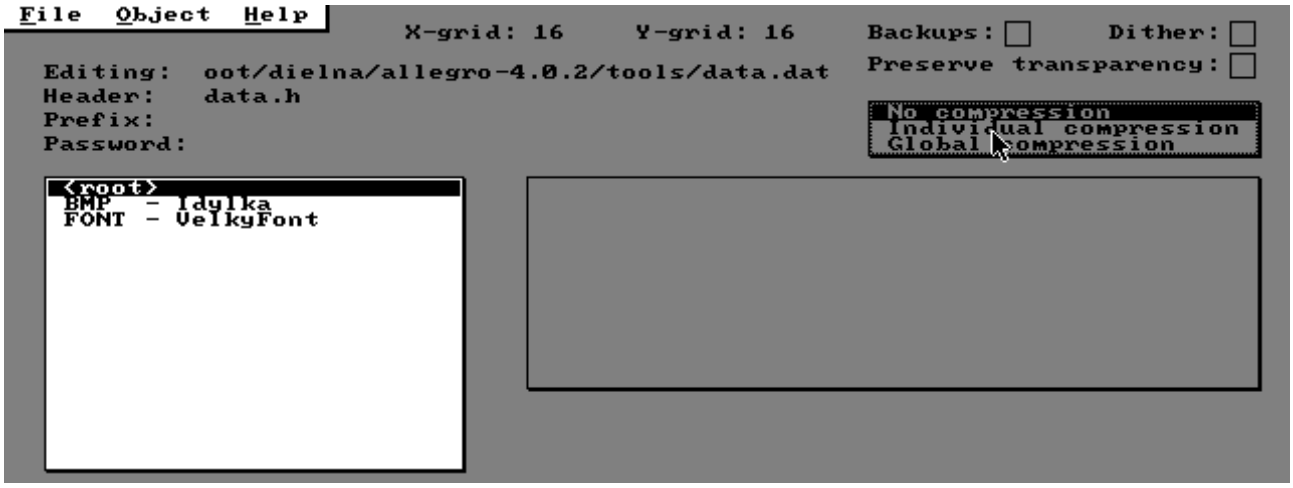
Keď človek píše nejakú hru alebo iný multimediálny projekt, väčšinou k tomu potrebuje množstvo súborov. Bitmapy, fonty, animácie, zvukové sekvencie alebo MIDI súbory sú podstatnou súčasťou programu a ich vytvorenie často stojí veľkú námahu. Okrem ich vytvárania ale môže byť problémom aj ich rozumná organizácia. Napchať tisíc súborov do jedného adresára je jednak neprehľadné pre autora, jednak to robí problémy pri distribúcii a okrem toho je zbytočne pracné pri načítavaní päťsto bitmáp päťstokrát volať funkciu `load_bitmap` vždy s iným menom súboru.

Tento problém rieši Allegro s pomocou dátových súborov. Dátový súbor je súbor, do ktorého všetky vami vytvorené multimediálne súbory naukladáte. Takýto súbor potom v programe načítate naraz a nemusíte postupne otvárať všetky jednotlivé súbory.

Na vytváranie dátových súborov slúži špeciálny program `grabber`. Ak chcete začať vytvárať nový dátový súbor (napríklad s názvom `data.dat`), musíte mať samozrejme vytvorené nejaké súbory, ktoré do neho chcete uložiť. Potom napíšte

```
grabber data.dat
```

Objaví sa vám niečo podobné, ako to, čo vidíte na obrázku.



Vo vašom prípade ale nebude vyplnená kolónka `Header` a v bielom obdĺžniku vľavo budete mať iba položku `<root>`. Do kolónky Header dajte meno hlavičkového súboru prislúchajúceho k dátovému súboru, ktorý práve vyrábate. Ten potom budete musieť includovať v programe, ktorý ten dátový súbor bude používať.

Nové položky do dátového súboru pridáte tak, že pravým tlačítkom myši kliknete na `<root>` v bielom obdĺžniku. Presuňte sa myšou nad `New` a potom vyberte, čo nového idete do dátového súboru pridať. Pre začiatok vyberte `Bitmap`. Grabber sa vás spýta, ako sa vaša bitmapka bude volať. Týmto menom sa budete odvolávať na bitmapku v programe. Zadajte `Idylka`. Keď meno zadáte, objaví sa vám v obdĺžniku nová položka. Zatiaľ ale obsahuje úplne inú bitmapu, než

chcete. Kliknite pravým tlačítkom na novopribudnutú položku a vyberte `Grab`. Vyskočí vám dialógové okno, v ktorom si vyberiete, ktorý že to obrázok má byť tá Idylka (tento). To, čo ste si vybrali, uvidíte vpravo od bieleho obdĺžnika. Nová bitmapa je súčasťou dátového súboru. Rovnakým spôsobom sme pridali font `VelkyFont` (tento).

Keď už v dátovom súbore máte všetko, čo potrebujete, uložte ho na disk. (Postup je štandardný – File, Save.) Potom môžete grabber opustiť príkazom `Ctrl-Q`.

Program, ktorý takto vytvorený dátový súbor využíva, môže vyzerať napríklad takto:

```
 1   #include <allegro.h>
 2   #include "data.h"
 3
 4   int main()
 5   {
 6      DATAFILE *datafile;
 7
 8      allegro_init();
 9      install_keyboard();
10      set_color_depth(16);
11      set_gfx_mode(GFX_AUTODETECT, 320, 200, 0, 0);
12
13      datafile = load_datafile("data.dat");
14
15      blit(datafile[Idylka].dat, screen, 0, 0, 85, 10, 150, 150);
16      textout_centre(screen, datafile[VelkyFont].dat,
17         "Lodicka", 160, 170, makecol(180, 180, 255));
18
19      readkey();
20      unload_datafile(datafile);
21
22      return 0;
23   }
24   END_OF_MAIN();
```

Na riadku 2 sa načíta súbor `data.h`. Je v úvodzovkách (musí sa preto nachádzať v tom istom adresári, ako zdrojový súbor) a vďaka nemu môžeme potom používať názvy ako `Idylka` a `VelkyFont`. Na riadku 6 je deklarovaný smerník na začiatok poľa, v ktorom sú uložené jednotlivé položky typu `DATAFILE`. Na riadkoch 8–11 sa štartuje Allegro (tento kód je zostručnený na maximálnu možnú mieru, nerobíme žiadne kontroly, ktoré by sa robiť mali). Na riadku 13 sa konečne načíta náš dátový súbor. Jednotlivé jeho položky sú potom prístupné ako prvky poľa `datafile`. Keďže štruktúra `DATAFILE` je zložitejšia (ak chcete vedieť detaily, pozrite si manuál), smerník na samotné dáta prvej položky získate ako `datafile[0].dat`. Na riadku 15 takto vykreslíme našu bitmapu `Idylka`. (Vďaka hlavičkovému súboru `data.h` náš program vie, že `Idylka` má hodnotu 0. Pozrite si ten súbor, aby ste videli, ako je to urobené.) Na riadkoch 16–17 podobne našim fontom vypíšeme nápis „Lodicka“. Na riadku 20 vypraceme dátový súbor z pamäti.

Úloha č.1: Pochopte a vyskúšajte.

Úloha č.2: S použitím funkcie `stretch_blit` natiahnite obrázok na celú obrazovku

Úloha č.3: Pridajte do súboru `data.dat` niektorú z predošlých bitmáp a vykreslite ju. Ako sa zmenil súbor `data.h`?

Úloha č.4: Upravte vzorový program z lekcie 5 tak, aby používal dátové súbory.

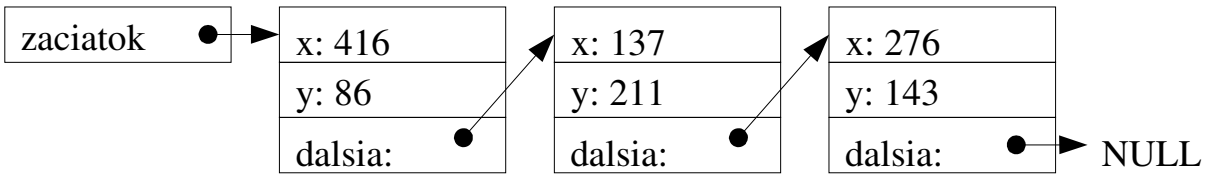
11. lekcia
Zoznamy
alebo "Ide vláčik ši, ši, ši"

Táto lekcia bude istým spôsobom výnimočná. Napriek tomu, že sa nachádza v cykle o knižnici Allegro, nebudeme túto knižnicu tentokrát používať. Vystačíme si s obyčajným C-čkom. Totiž – v mnohých programoch je potrebné ukladať si informácie o rôznych objektoch do pamäte. Ak je počet objektov stále rovnaký, je možné vytvoriť si pole a do neho informácie ukladať. Ale je veľmi pravdepodobné, že počet ufónov, ktorí budú útočiť na vašu základňu sa bude meniť. A tu je vhodnejšie použiť iné dátové štruktúry ako napríklad **zoznamy**.

Predstavte si, že si potrebujete uložiť údaje o viacerých potvorách, ktoré vo vašej hre budú útočiť na chudáka hráča. Pre jednoduchosť si budeme pamätať iba ich súradnice (v reálnej hre môže byť tých údajov oveľa viac). Najprv si vyrobíme novú štruktúru určenú pre jednu potvoru. Môže vyzerať napríklad takto:

```
typedef struct potvora
{
    int x;
    int y;
    struct potvora *dalsia;
} POTVORA;
```

Štruktúra `POTVORA` má tri zložky. Prvé dve – `x` a `y` budú obsahovať súradnice, na ktorých sa bude potvora nachádzať. Tretia zložka je smerník na ďalšiu štruktúru typu `POTVORA`. To nám umožní zoradiť všetky potvory ktoré sú práve na hracej ploche do vláčiku, ktorý bude vyzerať napríklad takto:

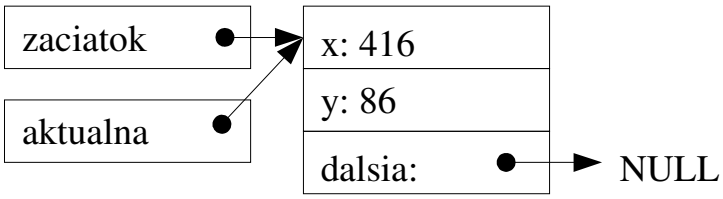


Dá sa to spraviť napríklad nasledujúcim spôsobom. Najprv si deklarujeme premennú `i` typu `int` a tri smerníky na štrukúru `POTVORA`, ktoré nazveme `zaciatok`, `aktualna` a `nova`. S pomocou funkcie `malloc` si v pamäti alokujeme prvú potvoru. Smerník `aktualna` nastavíme tak, aby na ňu ukazoval tiež. Hodnoty `x` a `y` nastavíme na náhodné hodnoty a smerník `dalsia` predbežne nastavíme na `NULL`.

```
int i;
POTVORA *zaciatok, *aktualna, *nova;

zaciatok = (POTVORA *) malloc(sizeof(POTVORA));
aktualna = zaciatok;
aktualna -> x = random() % 640;
aktualna -> y = random() % 480;
aktualna -> dalsia = NULL;
```

Keď tento kód prebehne, situácia bude približne takáto:



A teraz môžeme pridávať ďalšie vagóniky napríklad týmto spôsobom:

```
for( i = 0; i < 10; i++ )
{
    nova = (POTVORA *) malloc(sizeof(POTVORA));
    aktualna -> dalsia = nova;
    aktualna = nova;
    aktualna -> x = random() % 640;
    aktualna -> y = random() % 480;
    aktualna -> dalsia = NULL;
}
```

Smerník `aktualna` ukazuje na poslednú potvoru v zozname. V prvom príkaze v tele cyklu alokujeme v pamäti miesto pre novú potvoru. V druhom príkaze ju pripojíme na koniec vláčiku. Potom smerník `aktualna` nastavíme na novopridaný prvok a naplníme jednotlivé položky. V každom behu cyklu pridáme jeden vagónik. Keď ho teda necháme zbehnúť dvakrát, bude situácia vyzerať presne ako na prvom obrázku, ak prebehne desaťkrát, vagónikov bude 11.

Dôležité upozornenie: Ak neopatrne zmeníte hodnotu smerníka `zaciatok`, zabudnete, kde vláčik začína a ten začiatok sa nenávratne stratí v hlbinách pamäte počítača. Preto si dávajte pozor pri manipulácii s týmto smerníkom.

Ak chcete prejsť všetky prvky zoznamu, spraví sa to jednoducho. Nasledujúci cyklus vypíše súradnice všetkých zúčastnených potvor:

```
aktualna = zaciatok;
while (aktualna != NULL)
{
    printf("[%d, %d]\n", aktualna -> x, aktualna -> y);
    aktualna = aktualna -> dalsia;
}
```

Na začiatku smerník `aktualna` nastavíme na začiatok. Vypíšeme údaje potvory, na ktorú ukazuje a smerník `aktuálna` nastavíme na ďalšiu. Toto opakujeme dovtedy, kým smerník `aktuálna` ukazuje na niečo zmysluplné.

Keď prácu so zoznamom skončíme, patrí sa uvoľniť pamäť. Spravíme to napríklad takto:

```
while (zaciatok != NULL)
{
    aktualna = zaciatok -> dalsia;
    free((void *) zaciatok);
    zaciatok = aktualna;
}
```

Funguje to tak, že si v premennej `aktualna` uložíme druhý prvok v zozname, potom zmažeme začiatok zoznamu a smerník `zaciatok` nastavíme na nový začiatok vláčiku. Toto opakujeme, kým je čo mazať.

Uf. Je toho hodne na pochopenie. Takže si to pre istotu prečítajte ešte raz a potom sa pustite do úloh.

Úloha č.1: Pochopte, poskladajte z toho zmysluplný program a vyskúšajte.

Úloha č.2: Napíšte funkciu, ktorá v hotovom zozname vynuluje súradnice všetkých potvor.

Úloha č.3: Napíšte funkciu, ktorá zo zoznamu vymaže piaty prvok, ak taký existuje. Vypíšte obsah zoznamu pred a po zavolaní funkcie.

Úloha č.4: Vytvorte zoznam, v ktorom sa dá pohybovať oboma smermi. (Každý vagónik obsahuje okrem šipky na ďalšiu potvoru aj šipku na predošlú.) Vyrobte zoznam s desiatimi prvkami a vypíšte ho odpredu aj odzadu.

12. lekcia
Klávesnica alebo "Dajako to riadiť treba"

Pri tvorbe počítačovej hry je jednou z dôležitých častí spracovanie vstupu od užívateľa. Keby sme ho totiž nespracovávali, nejednalo by sa o počítačovú hru, ale o film. Ak si odmyslíme špeciálny hardware ako volanty a snímače polohy, vstup sa deje tromi štandardnými cestami – cez klávesnicu, cez myš alebo cez joystick.

O zisťovaní polohy myši sme už hovorili v predošlých lekciách. V tejto lekcii sa budeme venovať klávesnici. Ako už viete, jej použitie treba zapnúť funkciou `install_keyboard()`. Allegro vie pristupovať ku klávesnici dvoma rôznymi spôsobmi. Jeden prístup sa nazýva **zásobníkový**. To znamená, že stlačené klávesy sa ukladajú do zásobníka a keď ich chcete odtiaľ vytiahnuť, zavoláte si patričnú funkciu. (Pozn.: Tento prístup používa aj linuxovský shell. Keď napríklad spustíte editor bez použitia znaku &, všetko, čo v okne príkazového riadku stlačíte sa ukladá do zásobníka a shell tomu venuje pozornosť až keď prácu s editorom skončíte.)

Čo sa zásobníkového spracovania týka, už sme sa stretli s funkciou `keypressed()` ktorá vracala hodnotu `pravda` ak v zásobníku niečo čakalo a hodnotu `nepravda` ak bol prázdny. Znak zo zásobníka môžeme získať príkazom `readkey()`. Ak v zásobníku nič nie je, funkcia čaká dovtedy, kým tam niečo nebude. Na to, aby sme získali ASCII kód stlačeného znaku, treba ale výstup z funkcie trochu upraviť (pridaním záhadnej sekvencie `& 0xff`, ktorá z daného čísla ponechá posledné dva bajty), ako sa môžete presvedčiť na nasledujúcom príklade.

```
if ((readkey() & 0xff) == 'd')
   printf("Stlacili ste 'd'\n");
```

Niekedy treba obsah zásobníka vymazať. Na to slúži funkcia `clear_keybuf()`.

Pri programovaní hier sa ale častejšie uplatní **priamy** prístup. Vtedy chceme vedieť, ktoré klávesy sú na klávesnici v danom momente stlačené a ktoré nie. Tento cieľ sa vďaka spôsobu, akým sú klávesnice skonštruované nedá celkom splniť, ale Allegro robí čo môže, aby ste z klávesnice dostali maximum možných informácií.

Keď inicializujete klávesnicu, máte k dispozícii automaticky definované pole `key`, s pomocou ktorého viete zistiť, či je určitá klávesa stlačená. Ak chceme zistiť, či je stlačená klávesa `ESC`, spravíme to napríklad takto:

```
if (key[KEY_ESC])
   printf("ESC je stlacene.\n");
else
   printf("ESC nie je stlacene.\n");
```

Na jednotlivé klávesy sa odvolávame skratkami `KEY_A ... KEY_Z, KEY_0 ... KEY_9, KEY_0_PAD ... KEY_9_PAD` a špeciálne klávesy ako napríklad klávesy `KEY_ESC, KEY_SPACE, KEY_ENTER, KEY_LEFT, KEY_RIGHT, KEY_UP, KEY_DOWN` atď. (Úplný zoznam je v dokumentácii ku knižnici Allegro.)

Použitie môžete vidieť v nasledujúcom príklade, kde sa s pomocou kláves pohybuje oranžovým krúžkom:

```
 1   #include <allegro.h>
 2
 3   main()
 4   {
 5      int x=0,y=0;
 6      int oldx, oldy;
 7
 8      allegro_init();
 9      install_keyboard();
10      install_timer();
11
12      set_color_depth(16);
13      set_gfx_mode(GFX_AUTODETECT_FULLSCREEN, 640, 480, 0, 0);
14
15      circlefill(screen, x, y, 10, makecol(255,127,0));
16
17      while(! key[KEY_ESC])
18      {
19         oldx = x;
20         oldy = y;
21
22         if (key[KEY_LEFT] || key[KEY_4_PAD])
23            x--;
24         if (key[KEY_RIGHT] || key[KEY_6_PAD])
25            x++;
26         if (key[KEY_UP] || key[KEY_8_PAD])
27            y--;
28         if (key[KEY_DOWN] || key[KEY_2_PAD])
29            y++;
30
31         if (x < 0) x = 0;
32         if (x > 640) x = 640;
33         if (y < 0) y = 0;
34         if (y > 480) y = 480;
35
36         if (x != oldx || y != oldy)
37         {
38             circlefill(screen, oldx, oldy, 10, makecol(0,0,0));
39             circlefill(screen, x, y, 10, makecol(255,127,0));
40         }
41         rest(1);
42      }
43   }
44   END_OF_MAIN();
```

V premenných `x` a `y` uchovávame aktuálnu polohu krúžku. V riadkoch 8–13 inicializujeme veci potrebné pre Allegro. Na riadku 15 nakreslíme krúžok, ktorým budeme hýbať. Samotný program sa vykonáva v cykle na riadkoch 17 až 42, ktorý sa ukončí stlačením klávesy ESC. Na začiatku sa do premenných `oldx` a `oldy` uloží aktuálna pozícia krúžku. Na riadkoch 22–29 sa upraví aktuálna pozícia krúžku (ak je napríklad stlačená šipka vľavo `KEY_LEFT`, súradnica `x`

sa zmenší o 1). Riadky 31–34 dbajú o to, aby sa krúžok nedostal mimo obrazovky. Ak sa súradnice krúžku zmenili (skontroluje sa na riadku 36), na jeho miesto sa nakreslí čierny krúžok a nový krúžok sa nakreslí na novej pozícii, čo vyvoláva zdanie pohybu.

Úloha č.1: Prečítajte, pochopte, napíšte, skompilujte a vyskúšajte. Čo sa stane, ak naraz stlačíte šipky vpravo a dole? Prečo sa to deje?

Úloha č.2: Upravte program tak, aby po stlačení klávesy `F` krúžok náhodne zmenil farbu.

Úloha č.3: Upravte program tak, aby sa po obrazovke hýbal sprit slniečka.

Úloha č.4: Upravte program z úlohy č.2 tak, aby sa po stlačení klávesy `N` krúžok presunul na nové náhodné miesto na obrazovke.

Úloha č.5: (pre guruov, aby sa neflákali) Upravte program č.3 tak, aby sa stále menili fázy slniečka. Ak to budete mať, upravte to tak, aby sa slniečko hýbalo po nejakom viacfarebnom pozadí a nezničilo ho. (To už by mohol byť celkom sľubný základ pre nejakú hru.)

13. lekcia
Zvukové efekty
alebo "Chcelo by to výbuch"

Málo hier sa dnes už zaobíde bez zvukov. Postavy hovoria, výbuchy vybuchujú, spoza dvier je počuť tajuplné kroky a na pozadí hrá podmanivá hudba. Zvuky hre dodávajú atmosféru a zážitok z ozvučenej hry je oveľa väčší než z nemého „filmu“.

Zvuky ku hre je treba najprv získať. Nahráte ich v profesionálnom štúdiu alebo stokorunovým mikrofónom alebo stiahnete z internetu. Formátov, v ktorých sa dajú uložiť je nepreberné množstvo, Allegro však vie pracovať iba s dvoma formátmi.[5] Pre mono nahrávky používa formát `.voc` a pre stereo formát `.wav`

Keď takýto zvuk chceme natiahnuť do Allegra, buď to spravíme cez dátové súbory, ktoré boli popísané v 10. lekcii, alebo si deklarujeme smerník na štruktúru `SAMPLE` a zvuk načítame do pamäte s pomocou funkcie `load_sample` ako sme to ukázali v ukážkovom príklade.

Ak chceme používať zvukový podsystém, musíme ho, ako všetko v Allegre, inicializovať. Na to slúži funkcia `install_sound`, ktorá má tri parametre, z ktorých je momentálne zaujímavý iba prvý (druhý sa týka nastavenia prehrávania MIDI súborov a tretí je tam vyložene z historických dôvodov). Takže zvuk nainštalujete príkazom

```
install_sound(DIGI_AUTODETECT, MIDI_NONE, NULL);
```

Keď chceme nejaký zvuk zahrať, slúži na to funkcia `play_sample`. Táto má parametrov päť a tentokrát sú všetky dôležité. Prvý parameter hovorí, ktorý to zvuk treba zahrať. Druhý parameter je číslo od 0 do 255, ktoré hovorí, ako hlasno sa to má zahrať. Tretí parameter hovorí, z ktorej strany bude zvuk počuť. 255 znamená pravý reproduktor, 0 znamená ľavý, ak to chceme mať v strede, treba použiť 128. Štvrtý parameter je o tom, ako vysoko to chceme zahrať. Hodnota 1000 znamená „rovnako vysoko, ako to bolo nahraté“, 2000 je dvojnásobná frekvencia, 500 polovičná... Posledný parameter je buď `TRUE` alebo `FALSE`. Ak je `FALSE`, zvuk sa zahrá raz a hotovo. Ak je ale `TRUE`, zvuk sa opakuje stále dookola až kým nezavoláme funkciu `stop_sample` so smerníkom na zvuk, alebo kým neskončí program. Toto sa používa hlavne vtedy, keď chcete pustiť do pozadia nejakú hudbu, ktorú necháte hrať stále dokola, kým neskončí daný level.

Takže ukážkový program môže vyzerať takto:

```
 1   #include "allegro.h"
 2
 3   int main()
 4   {
 5      SAMPLE *vybuch;
 6
 7      allegro_init();
 8      install_keyboard();
 9      install_timer();
10      install_sound(DIGI_AUTODETECT, MIDI_NONE, NULL);
11
```

5 Samozrejme, Allegro sa dá presvedčiť, aby vedelo pracovať aj s inými formátmi (napr. mp3) ale treba na to nainštalovať ďalšie knižnice.

```
12      vybuch = load_sample("bum.wav");
13
14      set_gfx_mode(GFX_SAFE, 320, 200, 0, 0);
15      play_sample(vybuch, 255, 128, 1000, TRUE);
16
17      readkey();
18      stop_sample(vybuch);
19      destroy_sample(vybuch);
20  }
21  END_OF_MAIN()
```

Nič nečakaného sa tam nedeje. Na riadku 10 inicializujeme zvuk, na riadku 12 načítame obsah súboru `bum.wav` do pamäte a do smerníka `vybuch` vložíme, kde ten zvuk v pamäti je. Na riadku 15 to necháme hrať stále dookola až kým sa niečo na klávesnici nestlačí. Potom ten randál vypneme a vymažeme z pamäte a všetko skončí.

Úloha č.1: Pochopte a vyskúšajte (na počítači so zvukovkou, požičajte si slúchadlá alebo zapnite reproduktory).

Úloha č.2: Experimentujte s parametrami funkcie `play_sample`. Spustite ju päťkrát za sebou tak, aby sa zvuk postupne presúval zprava doľava, spustite zvuk s dvojnásobnou a polovičnou frekvenciou a vôbec, skúšajte, čo to dá.

14. lekcia
Dialógy
alebo "Stlačte OK!"

Ak programujete hru (alebo čokoľvek iné), niekedy sa ocitnete v situácii, kedy je potrebné užívateľovi poskytnúť možnosť niečo si nastaviť. A na to je užitočné mať k dispozícii systém tlačidiel, menu, editovacích boxov a vôbec všetkých možných vymožeností, ktoré sa označujú GUI (General User Interface) a ktoré dôverne poznáte z väčšiny okienkoidných programov.

Na programovanie dialógov sa častejšie používajú objektovo orientované jazyky ako C++, Java, Delphi či Smalltalk. Knižnica Allegro ale poskytuje možnosť naprogramovať jednoduchý dialóg aj v jazyku C. V tejto lekcii sa nachádza stručný náčrt toho, ako sa to robí. Ide však o pomerne rozsiahlu tému a tak čo sa týka ďalších detailov odkazujeme čitateľa na dokumentáciu ku knižnici Allegro.

Pri tvorbe dialógu sú dôležité niektoré pojmy: **Objekt** je vec na obrazovke, ktorú vidíte, môžete na ňu klikať, niečo do nej písať alebo si vyberať z ponúknutých možností. Objekty sú napr. button (tlačidlo) alebo editbox. **Udalosť** je vec, ktorá sa môže objektu prihodiť. V systéme, ktorý používa Allegro to môže byť napríklad `MSG_DRAW` ktorá objektu povie, že sa má prekresliť, alebo `MSG_CLICK`, ktorá objektu povie, že sa na neho kliklo myšou. Rôznych udalostí je veľa a ich úplný zoznam nájdete v manuáli. Posledným dôležitým pojmom je **správca udalostí** (alebo **message handler**). Je to funkcia, ktorá má daný objekt na starosti – ak objekt dostane správu, že sa má prekresliť, funkcia to obstará, ak dostane správu, že sa na neho kliklo, funkcia zaistí patričnú odozvu. Správcov niektorých typov udalostí už Allegro obsahuje, iných si môžete vyrobiť sami.

Každý dialóg v Allegre je reprezenvovaný poľom štruktúr typu `DIALOG`, ktorého posledná položka obsahuje samé nuly. Štruktúra `DIALOG` vyzerá takto:

```
typedef struct DIALOG
{
   int (*proc)(int, DIALOG *, int); - správca udalostí
   int x, y, w, h;                  - pozícia a veľkosť objektu
   int fg, bg;                      - farba popredia a pozadia
   int key;                         - ASCII klávesová skratka
   int flags;                       - príznaky stavu objektu
   int d1, d2;                      - čokoľvek, čo sa vám hodí
   void *dp, *dp2, *dp3;            - smerníky na ďalšie dáta
                                      týkajúce sa objektu
} DIALOG;
```

Prvá zložka štruktúry je handler daného objektu. Potom nasledujú súradnice ľavého horného rohu, šírka a výška objektu, farba popredia a pozadia, klávesová skratka (písmeno, ktoré keď stlačíte, objekt obdrží patričnú akciu), maska príznakov (tu sa dá objektu povedať, že má byť neaktívny (`D_DISABLED`) alebo že po stlačení tlačidla sa má celý dialóg skončiť (`D_EXIT`), ďalšie podrobnosti v manuáli.) Posledných päť parametrov slúži na odovzdávanie dát objektu. Napríklad reťazec, ktorý má byť napísaný v tlačidle sa uvedie ako parameter `dp`.

Handler objektu je funkcia, ktorá musí mať nasledujúcu podobu:

```
int nejaky_moj_handler(int msg, DIALOG *d, int c);
```

Je to teda funkcia, ktorá má tri parametre. Prvý je udalosť, ktorá volanie funkcie vyvolala, druhý je smerník na objekt, ktorého sa udalosť týka (to je užitočné zvlášť vtedy, ak je jedna funkcia handlerom viacerých objektov; vtedy si vďaka tomuto smerníku vie funkcia zistiť, kto ju vlastne volal). Tretí parameter sa týka len udalostí vyvolaných klávesnicou a momentálne sa mu nebudeme venovať. Handler podľa typu udalosti urobí to, čo má a vráti hodnotu. Táto je obyčajne `D_O_K`. Ak treba celý dialóg prekresliť vráti hodnotu `D_REDRAW` a ak jeho činnosť treba ukončiť, hodnotu `D_CLOSE` (sú ešte ďalšie možnosti – znovu je odporúčané čítať manuál). Niektoré handlery už knižnica Allegro obsahuje. Sú to napríklad `d_box_proc`, ktorý vie nakresliť obdĺžnik, `d_button_proc`, ktorý vie vyrobiť a spravovať tlačidlo a `d_edit_proc`, ktorý spravuje editovateľný riadok.

Takže toľko teória, poďme sa pozrieť, ako to všetko vyzerá v praxi.

```
 1   #include <allegro.h>
 2
 3   char retazec[20] = "Juchacha!";
 4   int farba = 65535;
 5
 6
 7   int tlacidielko(int msg, DIALOG *d, int c);
 8
 9
10   DIALOG dialog[] = {
11   /* proc,  x,  y,  w, h, fg, bg,
12                       key, flag, d1, d2, dp, dp2, dp3 */
13
14   {d_box_proc, 200, 100, 240, 280, 65535, 0,
15                              0, 0, 0, 0, NULL, NULL, NULL},
16
17   {tlacidielko, 210, 110, 90, 20, 31, 0,
18                     'c', 0, 1, 0, "&Cervena", NULL, NULL},
19
20   {tlacidielko, 330, 110, 90, 20, 2016, 0,
21                      'z', 0, 2, 0, "&Zelena", NULL, NULL},
22
23   {tlacidielko, 275, 140, 90, 20, 65535, 0,
24                    's', 0, 3, 0, "&Standard", NULL, NULL},
25
26   {d_edit_proc, 210, 200, 190, 20, 65535, 0,
27        0,  0,  sizeof(retazec)-1, 0, retazec, NULL, NULL},
28
29   {d_button_proc, 275, 240, 90, 20, 65535, 0,
30                    'o', D_EXIT, 0, 0, "&OK", NULL, NULL },
31
32   {NULL,              0,   0,  0,  0,  0,  0,
33                              0,  0, 0, 0, NULL, NULL, NULL}
34   };
35
36
37   int tlacidielko(int msg, DIALOG *d, int c)
38   {
39       if (msg == MSG_CLICK || msg == MSG_KEY)
40       {
```

```
41            switch (d -> d1)
42            {
43            case 1:
44                clear_to_color(screen,makecol(255,0,0));
45                farba = makecol(255,0,0);
46                break;
47            case 2:
48                clear_to_color(screen,makecol(0,255,0));
49                farba = makecol(0,255,0);
50                break;
51            case 3:
52                clear_to_color(screen,makecol(128,128,128));
53                farba = makecol(255,255,255);
54            }
55            return D_REDRAW;
56        }
57        else
58            return d_button_proc(msg, d, c);
59    }
60
61
62    main()
63    {
64       allegro_init();
65       install_keyboard();
66       install_mouse();
67       install_timer();
68       set_color_depth(16);
69       set_gfx_mode(GFX_SAFE, 640, 480, 0, 0);
70
71       clear_to_color(screen,makecol(128,128,128));
72       popup_dialog(dialog,-1);
73       clear_to_color(screen,makecol(0,0,0));
74       textout_centre(screen,font,retazec,320,240,farba);
75       while(!key[KEY_ESC]);
76    }
77    END_OF_MAIN();
```

Dialóg v uvedenom príklade je definovaný na riadkoch 10–34. Prvá položka je rámik. Ako handler je použitá funkcia `d_box_proc`, ktorá sa stará hlavne o jeho vykreslenie. Vo vnútri rámiku budú tri tlačítka na zmenu farby (druhá, tretia a štvrtá položka poľa). Text na tlačítkach bude `Cervena, Zelena` a `Standard`. Písmená, pred ktorými je znak & (teda `C`,`Z` a `S`) budú podčiarknuté. Handlerom týchto tlačidiel bude funkcia `tlacidielko`. Táto v podstate všetku prácu zvalí na funkciu `d_button_proc`, iba v prípade, že sa jedná o akciu `MSG_CLICK` (kliknutie myšou) alebo `MSG_KEY` (zavolanie cez klávesovú skratku), funkcia si podľa položky `d1` zistí ktoré tlačidlo ju vlastne zavolalo a podľa toho nastaví hodnotu premennej farba a celú obrazovku premaľuje. V tomto prípade musí funkcia vrátiť hodnotu `D_REDRAW`, aby sa premaľovaný dialóg znovu zobrazil.

Štvrtá položka dialógu je editovací riadok. Spravuje ho funkcia `d_edit_proc`. V tomto riadku sa dá meniť obsah reťazca `retazec`. Ako prvok `d1` treba uviesť maximálnu dĺžku textu (a treba pamätať aj na ukončovaciu nulu reťazca) a ako prvok `dp` treba uviesť smerník na reťazec, ktorý sa bude meniť.

Piata položka je tlačidlo `OK`. V parametri `flag` má uvedené `D_EXIT`, aby po jeho stlačení dialóg ukončil činnosť. Handler je rovno `d_button_proc`.

Vo funkcii `main` sa naštartuje a prichystá všetko možné a potom sa dialóg spustí funkciou `popup_dialog`. (Táto funkcia vracia index objektu, ktorý spôsobil ukončenie dialógu, teda v našom prípade 5.) Keď práca dialógu skončí, zmaže sa obrazovka a zvolenou farbou sa vypíše zvolený text. Na konci je prázdny cyklus, ktorý sa cyklí až kým niekto nestlačí klávesu `ESC`.

Úloha č.1: Prečítajte trikrát (najmenej trikrát, lebo je to ťažké!!!), pochopte a vyskúšajte. Čo to spraví ak vo funkcii tlačidielko vrátite miesto hodnoty `D_REDRAW` hodnotu `D_O_K`?

Úloha č.2: Doplňte ďalšie tlačítko s prefarbovaním na bledomodro.

Úloha č.3: Doplňte tlačítko `Cancel` po ktorého stlačení program rovno skončí.

Úloha č.4: Doplňte tesne pred koniec programu riadok

```
alert("E pericoloso sporgersi",NULL,NULL,"OK",NULL,0,0);
```

Čo to spraví? Ako sa to zmení, ak miesto `NULL` nasledujúcom po `"OK"` vložíte `"Cancel"`?

15. lekcia
Záverečná lekcia alebo "Čo ďalej?"

Absolvovali ste kurz jazyka C. Naučili ste sa základné štruktúry jazyka, poznáte niektoré základné funkcie a viete čiastočne používať grafickú knižnicu Allegro. Ste v podobnej situácií, ako keby ste sa z nejakého jazyka (napríklad svahilčina) naučili gramatiku, niekoľko slovíčok a prečítali niekoľko článkov v časopise. Teda viete už dosť na to, aby ste sa pustili do nejakej serióznej roboty. Ale ak vaše vedomosti nebudete používať, ak nezačnete čítať nejakú svahilskú knižku, nezačnete prekladať do svahilčiny a nebudete sa pokúšať s niekym, kto po svahilsky hovorí dorozumieť, vyjde námaha, ktorú ste doteraz vynaložili navnivoč.

Podobne je to aj s programovaním a konkrétne aj s jazykom C. Ak nebudete v C-čku niečo rozumné programovať, ak nebudete čítať manuály, ak si nebudete prehlbovať svoje znalosti a rozvíjať techniku a programátorský štýl, spoznávať nové metódy programovania a učiť sa nové finty, vaša programátorská prítomnosť sa rýchle stane minulosťou.

Úlohou tejto lekcie je jednak povedať, kde hľadať ďalšie informácie, jednak vám dať návrhy na to, čo robiť a do akých projektov sa pustiť. Takže najprv k tým informáciam. Čo sa týka samotného jazyka C, niektoré učebniece idú viac do hĺbky, než náš kurz. Zvlášť odporúčame knižku **Pavel Herout: Učebnice jazyka C** ktorá je podľa nás najlepšou učebnicou C-čka v češtine alebo slovenčine. Ak máte práve málo peňazí, treba sa rozhliadnuť po internete. Na adrese `http://aslan.smnd.sk/anino/programming/c/saloun` je jeden relatívne úplný kurz jazyka C. Ak si ho prečítate, doplníte si oblasti, ktorým sme sa v našom kurze venovali len okrajovo.

Mnoho úloh z programovania už bolo vyriešených a nie vždy je účelné programovať ich odznova. Pozrieť sa po internete, či už niekto danú úlohu nevyriešil alebo či nie je k dispozícii funkcia, ktorú by ste potrebovali, ušetrí neraz čas aj robotu. Mnohé funkcie, ktoré sa môžu zdať užitočné, sú obsiahnuté už v klasickej knižnici `libc`, ktorá sa pri linkovaní s kompilátorom `gcc` pripája automaticky. Manuály ku knižnici `libc` nájdete na adrese `http://ftp.groovy.gr/docs/gnu_manuals/libc/html_node/index.html` Manuály sú po anglicky, ale čím skôr si zvyknete čítať anglickú dokumentáciu, tým lepšie.

Rovnako môže byť užitočné prejsť si manuály ku knižnici Allegro. Dodávajú sa priamo s knižnicou, prípadne si ich môžete stiahnuť priamo zo stránky Allegra `http://alleg.sourceforge.net`. Okrem nich existuje dokument pre začiatočníkov v programovaní v knižnici Allegro nazvaný Allegro Vivace. Slovenský preklad sa dá nájsť na adrese `http://aslan.smnd.sk/anino/moje/vivace`. Okrem úvodu ku knižnici Allegro tu môžete nájsť ukážky mnohých programátorských techník bežne používaných aj v iných oblastiach, než je počítačová grafika a tvorba počítačových hier.

Existujú mnohé iné špecializované stránky venované programovaniu (za mnohé uveďme len `http://ll.cz/programovani`). Keď viete o konkrétnej veci, ktorú hľadáte, tak sa treba obrátiť na vyhľadávací server (napr. Google je dobrý kamarát... `http://www.google.com`) a je veľmi pravdepodobné, že nájdete niečo, čo vám pomôže.

Toľko teda, čo sa týka veci „ako programovať“. Ďalšou nemenej dôležitou otázkou je „čo programovať“. Začnem opäť linkou na internete. Na stránke `http://www.ksp.sk` nájdete korešpondenčný seminár z programovania, zadania matematickej olympiády kategórie P a linky na ďalšie programátorské súťaže. Úlohy v nich nie sú náročné, čo sa týka znalosti jazyka (vaša znalosť jazyka C je úplne postačujúca), ale sú ťažké čo sa týka hľadania konkrétnych algoritmov. Rozhodne

sa pri ich riešení môžete mnohému priučiť.

Ďalšou z možností je pustiť sa do samostatného projektu – najlepšie do nejakej hry. (Predtým sa vrelo odporúča prečítať Allegro Vivace!!!) Na začiatok to pravdepodobne bude remake niečoho klasického. Ťahovky typu Gnobots (pozrite si túto hru pod Linuxom) patria medzi folklór a nie sú z programátorského hľadiska náročné. Podobne je mnoho matematických hier (napríklad hry typu NIM), ktoré majú vyhrávajúcu stratégiu a dajú sa ľahko programovať a počítač v nich môže vystupovať v úlohe protihráča.

Pokročilejší programátori sa môžu pustiť aj do zložitejších hier založených na animáciach a pohybe (ono aj vyššie spomenutým hrám pekná animácia dodá na kráse). Dali by sa spraviť napríklad žaby (Na obrazovke je 6 vodorovných úrovní po ktorých v nepravidelných intervaloch plávajú lekná, žaba musí preskákať zo spodku až nahor a nesmie pritom spadnúť do vody, ani sa leknom nechať vyniesť z obrazovky). Ďalšia klasická hra je Space Invaders – z rôznych miest na vrchu obrazovky sa začnú rojiť rôznou rýchlosťou inváziechtiví marťania, dole pobehuje raketka, ktorá ich ostreľuje a môžu jej uniknúť maximálne dvaja. Dala by sa spraviť aj nejaká ďalšia verzia PacMana – Hladnej potvorky pobehujúcej po bludisku a vyjedajúcej bodky, zatiaľ čo ju prenasledujú nejaké ďalšie príšery.

Chce to nápad a chuť pustiť sa do roboty. Inšpiráciou môžu byť staré Sinclairovské hry, ktorých sa spolu s emulátormi po sieti povaľuje veľké množstvo, na vylepšenia a samostatné nápady prídete isto aj sami. Len si na začiatok nevezmite veľmi veľké sústo a začnite od jednoduchších vecí.

Na záver jeden program v jazyku C. Napíšte (prípadne skopírujte) a spustite :) Mnoho šťastia v programovaní aj inak.

Anino

```
#include <stdio.h>
main(t,_,a)
char *a;
{
return!0<t?t<3?main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a)):
1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?
main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-72?main(_,t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l+,/n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n')()#}w'){){nl]'/+#n';d}rw' i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \
}'+}##(!!/")
  :t<-50?_==*a?putchar(a[31]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
    :0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a,
"!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);
}
```