DUM č. 20 v sadě

35. Inf-11 Objektové programování v Greenfoot

Autor: Lukáš Rýdlo

Datum: 08.06.2014

Ročník: studenti semináře

Anotace DUMu: Filtr vyplňování souvislé oblasti barvou. Implementace dynamicky alokovaného zásobníku pro čísla a pro souřadnice bodů.

evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Projekt: konzolový BMP editor v C
Filtr výplň souvislé oblasti

Úvod

Tato lekce přináší praktickou aplikaci známého teoretického tématu dynamické alokace paměti a sice použití dynamicky implementovaného zásobníku. Filtr pro vyplňování souvislé oblasti barvou je poměrně náročné téma jednak pro „grafickou teorii“, druhak při implementaci nerekurzivním algoritmem. Kromě zásobníku můžeme použít samozřejmě i frontu nebo jiné dynamické struktury. Rozdíl nebude vidět na výsledku, ale na průběhu vyplňování. Zásobník bude vybírat vložené prvky v jiném pořadí a proto bude oblast vyplňovat v jiném pořadí bodů. Ideální by bylo použít množinu nebo haldu, jelikož se v takové struktuře nemohou opakovat vložené prvky, což by významně urychlilo činnost algoritmu. Implementace fronty ani zásobníku standardně umožňuje opakované vložení prvku, který již ve struktuře je. To se projeví při vyplňování obrázku poměrně často a poměrně velké množství pixelů tak bude „vybarvováno opakovaně“.

Samotný princip výplně souvislé oblasti je následující:

1. Zvolíme počáteční bod vyplňování a zapamatujeme si jeho barvu.
2. Bod přebarvíme na novou barvu.
3. Zkontrolujeme body v tzv. 4-růžici nebo 8-růžici. To jsou buď body vlevo, vpravo, nad a pod původním (4-růžice) nebo i další čtyři sousedící diagonálně (8-růžice). Pro výplň souvislé oblasti ohraničené tenkou čarou je vhodnější 4-růžice. Naopak pro přebarvení tenkých čar je vhodnější 8-růžice, protože projde i po diagonále.
4. Pokud některý z těchto bodů v růžici má původní barvu počátku, vložíme jej do zásobníku.
5. Je-li to možné, ze zásobníku vybereme jeden prvek – bod a vracíme se do bodu 2. Pokud je již zásobník prázdný, končíme.

Před započetím implementace filtru je nutné implementovat zásobník a to pro strukturu dvou souřadnic bodu. To není úplně jednoduchá úloha, proto budou první dva úkoly věnované právě implementaci zásobníku. Zásobník je struktura LIFO (last in, first out), obvykle se implementuje buď staticky pevně velkým polem nebo dynamicky pomocí struktury, která uchovává hodnotu a ukazatel na následníka. Na internetu je kromě Wikipedie (http://cs.wikipedia.org/wiki/Z%C3%A1sobn%C3%ADk_%28datov%C3%A1_struktura%29) i celá řada stránek přímo s implementací (např. http://amilasurendra.info/dynamic-stack-with-c-using-linked-lists/) v různých jazycích a s různou kvalitou. Doporučuji na začátku hodiny společně projít graficky přidávání a odebírání prvku a test prázdnosti. Je dobré studentům kreslit alokované struktury a šipkami znázorňovat ukazatele…

Úkoly s řešeními

1. Implementujte zásobník pro celá čísla. Program po spuštění bude načítat od uživatele čísla tak dlouho, dokud nezadá nulu nebo neselže alokace paměti. Pak program zobrazí tato čísla v opačném pořadí (stačí je vybírat ze zásobníku a vypisovat). Nachystejte funkce push, pop a isEmpty a datovou strukturu sData pro ukládání položky zásobníku (data jsou integery).

   Takovýto úkol je vhodný příklad na procvičení pro maturanty, ale zvládne jej i šikovnější mladší student. Hotové řešení vypadá takto:

```
#include<stdio.h>
#include<stdlib.h>

struct sData {
   □ * next;
   int value;
};

□ stack = NULL;

int pop() {
   struct sData * old;
   int value;

   if (isEmpty()) □;
   old   = stack;
   value = □;
   stack = □;
   free(old);

   return value;
}

int push(int data) {
   struct sData * new;
   new = (□) malloc(sizeof(□ sData));
   if (□) return 0;
   new->value = □;
   new->next  = □;
   stack      = new;
   return 1;
}

int isEmpty() {
   return stack==□;
}

int main() {
   int number;
```

```
   do {
      printf("Cislo: (0-konec) ");
      scanf("%i",&number);
      if (!□(number)) {
        printf(" To se nepovedlo vlozit!\n");
        break;
      }
   } while (number□0);

   printf("-- Vypis --\n");

   while(!□) {
      printf("Hodnota v zasobniku je %i \n",□());
   }

   return 0;
}
```

2. Upravte předchozí kód tak, aby zásobník přebíral jako data strukturu pojmenovanou s2D, která bude obsahovat souřadnice x a y bodu ve 2D. Funkce push, pop a isEmpty implementujte v souboru stack2D.c a deklarujte v stack2D.h, kde bude také deklarace struktury s2D a struktury sStack2D, která bude položkou zásobníku. Nezapomeňte použít direktivy a konstantu _STACK2D_H k vypodmínkování obsahu hlavičkového souboru. Můžete vytvořit soubor stackmain.c s upravenou funkcí main a načítat dvojice čísel. Všechny funkce nechť přebírají ukazatelem i ukazatel na vrchol zásobníku, aby bylo možno je používat bez deklarace globální proměnné pro zásobník, resp. k více různým zásobníkům.

   Implementace je podobná předchozímu příkladu, vlastně stačí nahradit „int“ za „s2D“. Hlavičkový soubor by mohl vypadat takto:

```
#include<stdio.h>
#include<stdlib.h>

#□ _STACK2D_H
#define □

typedef struct {
   unsigned int x;
   unsigned int y;
} s2D;
struct sStack2D {
   struct sStack2D □ next;
   □    value;
};

s2D pop(struct sStack2D ** stack);
int push(s2D data,struct sStack2D ** stack);
int isEmpty(struct sStack2D ** stack);
#endif
```

Samotný stack2D.c je implementovaný takto:

```
#include<stdio.h>
#include<stdlib.h>
#include "□"

s2D pop(struct sStack2D ** stack) {
   □ old;
   □ value;

   if (isEmpty(□)) {
      value.x = 0;
      value.y = 0;
      return value;
   }
   old   = □stack;
   value.x = □;
   value.y = □;
   *stack = □;
   free(old);

   return value;
}

int push(s2D data,struct sStack2D ** stack) {

   if (stack==□) return -1;

   □ new;
   new = (□) malloc(sizeof(struct sStack2D));
   if (new == □) return 0;
   new->value.x = □;
   new->value.y = □;
   new->next  = □stack;
   *stack      = □;
   return 1;
}

int isEmpty(struct sStack2D ** stack) {
   if (stack == NULL) return -1;
   return □stack==NULL;
}
```

Implementace zkušebního mainu by mohla být v souboru stackmain.c:

```
#include<stdio.h>
#include<stdlib.h>
#include "□"

int main() {
```

```
   □ point;
   □ stack=NULL;

   do {
      printf("Cisla: (0,0-konec) ");
      scanf("%i",□(point.x));
      scanf("%i",□(point.y));
      if (!□(point, □stack)) printf(" To se nepovedlo vlozit!\n");
   } while (point.x□0□point.y□0);

   printf("-- Vypis --\n");

   while(!isEmpty(□stack)) {
      point = □(□stack);
      printf("Hodnota %i %i\n",point.x,point.y);
   }

   return 0;
}
```

3. Includujte do bmpfilters.c podporu pro 2D zásobník. Zaveďte nový filtr v podmínkách spuštění programu (v mainu) s písmenem volby filtru „v“. Bude přebírat dva soubory (vstupní a výstupní) a navíc dvě čísla – souřadnici x a y počátečního bodu vyplňování. Chcete-li můžete přebírat ještě další 3 čísla udávající barevné složky barvy, kterou se bude vyplňovat. Nechejte zatím pouze vypisovat tyto hodnoty jako čísla. Jak lze převést string na integer v jazyce C? (Hledaná funkce má čtyři písmena a nachází se v stdlib.h.) V hlavičkovém souboru bmpfilters.h deklarujte filtr makeFill. Nechte jej navíc kromě hlaviček a obrazových dat přebírat int souřadnic a případně i barevných složek.

   Úkol je jednodušší než předchozí (a následující), stačí hledat v dokumentaci potřebnou funkci (počítám s variantou s barvami):

```
makeFill(image, &fh, &ph, □toi(arg□[□]), □toi(arg□[□]), □toi(arg□[□]),
□toi(arg□[□]), □toi(arg□[□]), □toi(arg□[□]));
```

4. V souboru bmpfilters.c vytvořte následující pomocné funkce (netřeba je deklarovat v bmpfilters.h), budou sloužit jen jako pomocné pro makeFill.

```
int checkPointInImage(s2D point, picHeader * ph);
```

   Vrátí 1 pokud je bod na souřadnicích point uvnitř obrázku s obrazovou hlavičkou ph. Jinak vrátí nulu. Nezapomeňte, že bod se může dostat za okraj nejen vysokou, ale i příliš nízkou hodnotou.

```
void colorizePixel(sPixel ** image, s2D point, sPixel color, picHeader
* ph);
```

   Obarví bod na souřadnicích point v obrázku image barvou color. Hlavičku předáváme kvůli potřebě kontrolovat, zda bod je v obrázku (na což máme předchozí funkci).

```
int equalColor(s2D point, sPixel ** image, sPixel color, picHeader
*ph);
```

Vrátí 1, pokud bod o souřadnicích point na obrázku image má stejnou barvu, jako je color. Jinak vrací 0.

```
s2D newPoint(s2D point,int x, int y);
```

Vrátí nový 2D bod, který má souřadnice jako bod point, jen s přičtením x k x-souřadnici point a přičtením y k y-souřadnici point.

Tyto funkce jsou poměrně krátké, ale pomohou významně zpřehlednit kód funkce makeFill. Jejich implementace může vypadat takto (u „testovacích“ funkcí je za return logický výraz – podmínka…):

```
int checkPointInImage(s2D point, picHeader * ph) {
   return point.x□□ □ point.y□□ □ □□ph->width □ □□ph->height;
}

void colorizePixel(sPixel ** image, s2D point, sPixel color, picHeader
* ph) {
   if (□(point, ph)) {
      image[□][□].red   = □.red;
      image[□][□].blue  = □.blue;
      image[□][□].green = □.green;
   }
}

int equalColor(s2D point, sPixel ** image, sPixel color, picHeader
*ph) {
   if (□(point, ph)) {
      return image[□][□].red==□.red □
             image[□][□].green==□.green □
             image[□][□].blue==□.blue;
   }
   return 0;
}

s2D newPoint(s2D point,int x, int y) {
   □.x □= x;
   □.y □= y;
   return □;
}
```

5. Vytvořte ve bmpfilters.c (a ve bmpfilters.h deklarujte) funkci makeFill, která vyplní souvislou oblast se stejnou barvou jako má bod na předaných souřadnicích [x,y] nějakou (nejlépe předanou) novou barvou. Použijte 4-růžici, nerekurzivní implementaci pomocí zásobníku. Barva sousedních bodů musí být opravdu přesně stejná. Nakreslete si proto testovací obrázky ručně (na fotkách se budou odstíny byť drobně lišit). (V rámci testování filtru můžete počítat kolik kroků trvá vyplnění obrázku a jak se plní a vyprazdňuje zásobník. Ve finální verzi to ale nevypisujte.)

   Řešení tohoto úkolu samostatně zvládnou jen zdatnější studenti. Nemusí být na škodu vytvořit ho společně – učitel píše kód postupně na tabuli/dataprojektor a nechá si radit od studentů, případně opravuje chyby. Vzorové řešení je přiloženo:

```
void makeFill(sPixel ** image, fileHeader * fh, picHeader * ph, int x,
int y, int r, int g, int b) {
   s2D point;

   □ color;
   □ pixel;

   □ stack=NULL;

   color.red   = r;
   color.green = g;
   color.blue  = b;

   pixel.red   = □[x][y].red;
   pixel.green = □[x][y].green;
   pixel.blue  = □[x][y].blue;

   point.x=x;
   point.y=y;

   □(point,&stack);

   while(!□(&stack)) {
      point = □(&stack);
      □(image, point, color, ph);
      if (□(newPoint(point,-1,0),image,pixel,ph)) {
         □(newPoint(point,□,□),&stack);
      }
      if (□(newPoint(point,0,-1),image,pixel,ph)) {
         □(newPoint(point,□,□),&stack);
      }
      if (□(newPoint(point,1,0),image,pixel,ph)) {
         □(newPoint(point,□,□),&stack);
      }
      if (□(newPoint(point,0,1),image,pixel,ph)) {
         □(newPoint(point,□,□),&stack);
      }
   }
}
```

* BONUS: Jako domácí úkol pro nejschopnější studenty lze zkusit upravit funkci makeFill (nebo její pomocné) tak, vyplňovala souvislou oblast s určitou tolerancí ke změně barvy, tedy aby byla aplikovatelná na fotky a pokud např. barva mraků variuje s drobnou změnou kolem jednoho odstínu modré, aby funkce vyplnila celou oblast mraků, resp. oblast, kde se barevný odstín jen málo mění. Další užitečná úprava by spočívala v tom, že tato oblast s variujícím odstínem barvy by nebyla vyplněna souvisle jedním odstínem přebarvovací barvy, ale že by i její odstín mírně kolísal tak, jak kolísal odstín původní barvy.

  Tento rozšiřující úkol je složitější po stránce znalostí grafiky a programátorsky není příliš přínosný. Proto nebudu přikládat ani jeho řešení. Jako tip pro studenty, kteří by ho rádi zkusili: toleranci ke změně odstínu lze implementovat na úrovni funkce equalColor, která by měla tolerovat odchylku – testovat ne rovnost, ale přítomnost v intervalu. Je však třeba si rozmyslet, jak se toho docílí. Zda počítat odchylku vůči sousednímu pixelu nebo středovému nebo jinému. Vyplnění se změnou odstínu vyplňovací barvy vyžaduje při vyjmutí bodu ze zásobníku nejprve zjistit, jak se jeho odstín liší od původního (počátečního) bodu a stejným způsobem upravit i odstín vyplňovací barvy. I zde je potřeba mít na paměti, aby při přepočítávání hodnoty barvy nedošlo k přetečení číselného rozsahu.

* BONUS: Implementujte další jiné filtry, např.: obarvení fotky na jeden odstín, otočení, převrácení, rozmazání, nahrazení barvy jinou apod.

* BONUS: Implementujte vyplňování souvislé oblasti jinou strukturou, například frontou. Nebo upravte implementaci zásobníku tak, aby v případě vložení položky o souřadnicích, které již v zásobníku jsou, bod znovu nevkládala (ale nevyvolala chybu). Můžete také zkusit (u menších obrázků) po každém kroku vyplňování oblasti barvou uložit obrázek pod číslovaným názvem a z jednotlivých obrázků udělat animaci nebo video postupného vyplňování (ale pozor na velký – řádově i více než desetitisíce – počet kroků vyplňování i u poměrně malých obrázků).

Úkoly s řešeními

1. Implementujte zásobník pro celá čísla. Program po spuštění bude načítat od uživatele čísla tak dlouho, dokud nezadá nulu nebo neselže alokace paměti. Pak program zobrazí tato čísla v opačném pořadí (stačí je vybírat ze zásobníku a vypisovat). Nachystejte funkce push, pop a isEmpty a datovou strukturu sData pro ukládání položky zásobníku (data jsou integery).

2. Upravte předchozí kód tak, aby zásobník přebíral jako data strukturu pojmenovanou s2D, která bude obsahovat souřadnice x a y bodu ve 2D. Funkce push, pop a isEmpty implementujte v souboru stack2D.c a deklarujte v stack2D.h, kde bude také deklarace struktury s2D a struktury sStack2D, která bude položkou zásobníku. Nezapomeňte použít direktivy a konstantu _STACK2D_H k vypodmínkování obsahu hlavičkového souboru. Můžete vytvořit soubor stackmain.c s upravenou funkcí main a načítat dvojice čísel. Všechny funkce nechť přebírají ukazatelem i ukazatel na vrchol zásobníku, aby bylo možno je používat bez deklarace globální proměnné pro zásobník, resp. k více různým zásobníkům.

3. Includujte do bmpfilters.c podporu pro 2D zásobník. Zaveďte nový filtr v podmínkách spuštění programu (v mainu) s písmenem volby filtru „v“. Bude přebírat dva soubory (vstupní a výstupní) a navíc dvě čísla – souřadnici x a y počátečního bodu vyplňování. Chcete-li můžete přebírat ještě další 3 čísla udávající barevné složky barvy, kterou se bude vyplňovat. Nechejte zatím pouze vypisovat tyto hodnoty jako čísla. Jak lze převést string na integer v jazyce C? (Hledaná funkce má čtyři písmena a nachází se v stdlib.h.) V hlavičkovém souboru bmpfilters.h deklarujte filtr makeFill. Nechte jej navíc kromě hlaviček a obrazových dat přebírat int souřadnic a případně i barevných složek.

4. V souboru bmpfilters.c vytvořte následující pomocné funkce (netřeba je deklarovat v bmpfilters.h), budou sloužit jen jako pomocné pro makeFill.

```
int checkPointInImage(s2D point, picHeader * ph);
```

Vrátí 1 pokud je bod na souřadnicích point uvnitř obrázku s obrazovou hlavičkou ph. Jinak vrátí nulu. Nezapomeňte, že bod se může dostat za okraj nejen vysokou, ale i příliš nízkou hodnotou.

```
void colorizePixel(sPixel ** image, s2D point, sPixel color, picHeader
* ph);
```

Obarví bod na souřadnicích point v obrázku image barvou color. Hlavičku předáváme kvůli potřebě kontrolovat, zda bod je v obrázku (na což máme předchozí funkci).

```
int equalColor(s2D point, sPixel ** image, sPixel color, picHeader
*ph);
```

Vrátí 1, pokud bod o souřadnicích point na obrázku image má stejnou barvu, jako je color. Jinak vrací 0.

```
s2D newPoint(s2D point,int x, int y);
```

Vrátí nový 2D bod, který má souřadnice jako bod point, jen s přičtením x k x-souřadnici point a přičtením y k y-souřadnici point.

5. Vytvořte ve bmpfilters.c (a ve bmpfilters.h deklarujte) funkci makeFill, která vyplní souvislou oblast se stejnou barvou jako má bod na předaných souřadnicích [x,y] nějakou (nejlépe předanou) novou barvou. Použijte 4-růžici, nerekurzivní implementaci pomocí zásobníku. Barva sousedních bodů musí být opravdu přesně stejná. Nakreslete si proto testovací obrázky ručně (na fotkách se budou odstíny byť drobně lišit). (V rámci testování filtru můžete počítat kolik kroků trvá vyplnění obrázku a jak se plní a vyprazdňuje zásobník. Ve finální verzi to ale nevypisujte.)

* BONUS: Jako domácí úkol pro nejschopnější studenty lze zkusit upravit funkci makeFill (nebo její pomocné) tak, vyplňovala souvislou oblast s určitou tolerancí ke změně barvy, tedy aby byla aplikovatelná na fotky a pokud např. barva mraků variuje s drobnou změnou kolem jednoho odstínu modré, aby funkce vyplnila celou oblast mraků, resp. oblast, kde se barevný odstín jen málo mění. Další užitečná úprava by spočívala v tom, že tato oblast s variujícím odstínem barvy by nebyla vyplněna souvisle jedním odstínem přebarvovací barvy, ale že by i její odstín mírně kolísal tak, jak kolísal odstín původní barvy.

* BONUS: Implementujte další jiné filtry, např.: obarvení fotky na jeden odstín, otočení, převrácení, rozmazání, nahrazení barvy jinou apod.

* BONUS: Implementujte vyplňování souvislé oblasti jinou strukturou, například frontou. Nebo upravte implementaci zásobníku tak, aby v případě vložení položky o souřadnicích, které již v zásobníku jsou, bod znovu nevkládala (ale nevyvolala chybu). Můžete také zkusit (u menších obrázků) po každém kroku vyplňování oblasti barvou uložit obrázek pod číslovaným názvem a z jednotlivých obrázků udělat animaci nebo video postupného vyplňování (ale pozor na velký – řádově i více než desetitisíce – počet kroků vyplňování i u poměrně malých obrázků).