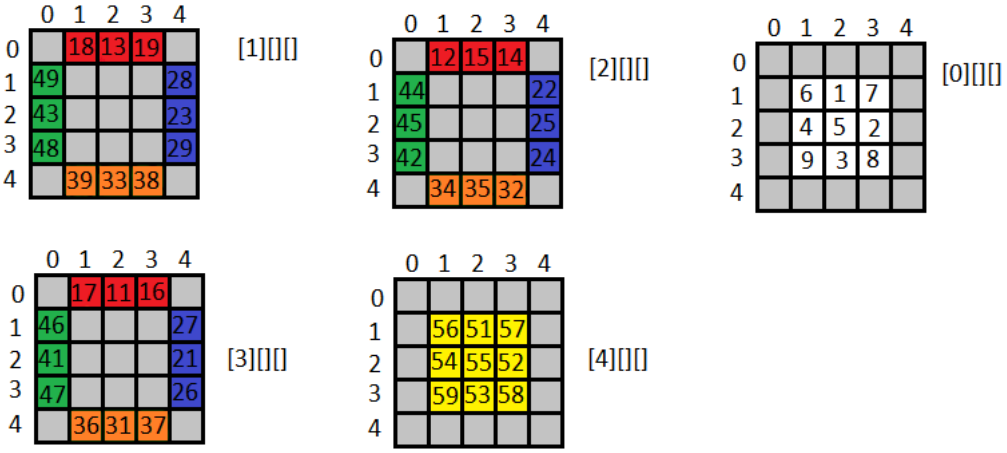
Technická dokumentace zápočtového programu
Skládání Rubikovky

První důležitou otázkou bylo navrhnout, jak bude Rubikovka uložená v paměti, co se bude dělat během jejího otáčení a jak značit jednotlivé dílky, aby se co nejvíce usnadnilo vyhledávání konkrétních dílků apod.

Nakonec jsem zvolil uložit Rubikovku do pole 5x5x5 s tím, že každá nálepka bude uložena ve vlastní buňce pole se svým vlastním identifikačním číslem, a zároveň tato identifikační čísla budou rozvrženy tak, aby při prohledávání tohoto pole bylo jednoduché napsat podmínku, díky které by program našel pouze pozice například hran s bílou barvou. Zde je nakresleno, jaké hodnoty se uloží do pole při inicializaci a jakou barvu reprezentují:



Inicializaci kostky provádí *void inicializace(char kostka[5][5][5])*.

Potom jsem potřeboval vytvořit funkci, která udělá zadaný tah tím, že přemístí hodnoty v jednotlivých buňkách do jiných buněk. Vytvořil jsem funkci *void tah(char ch, int kolikrat, char kostka[5][5][5], char vypsat)*. Ta přijme jako argumenty *char ch*, který má hodnotu podle toho, jaký tah se má udělat (např. U, r, x, …), *int kolikrat*, který značí, kolikrát se daný tah má udělat (0, pokud se nemá provést, 1, pokud se posune daná vrstva/y o 90° po směru hodinových ručiček, 2, pokud se posune o 180°, a 3, pokud se posune o 90° proti směru hodinových ručiček), odkaz na samotnou kostku v paměti a *char vypsat*, který má hodnotu 1, pokud provedení daného tahu chceme vypsat na obrazovku, a 0, pokud ne (což se později hodí v některých situacích, jak uvidíte).

Každý tah se dá složit z několika elementárních tahů, proto jsem udělal 9 funkcí, které provedou tahy U, D, R, L, F, B, M, E, S tak, že jsem natvrdo vypsal, hodnoty ve kterých buňkách se posunou na které souřadnice. *void tah* obsahuje jeden velký switch případů, který nastal tah, a podle případu provede dané elementární tahy; počet provedených tahů závisí na *int kolikrat*. Zároveň také vypíše daný tah na obrazovku (pokud se měl tah R provést 2krát, tak vypíše R2, ale pokud by se měl provést 3krát, tak vypíše R‘ s ohledem na notaci tahů).

Další funkcí je funkce na tisknutí *void tisk(char kostka[5][5][5])* – chtěl jsem, aby se vypisovala síť krychle a dané nálepky na kostce byly reprezentovány barvami. Jelikož při převádění souřadnic 3-rozměrné krychle do 2-rozměrné sítě krychle se tak úplně nedá řídit nějakým konkrétním vzorcem, tak jsem natvrdo vypsal, na kterém místě se má vypsat která barva (před tím se ale vytvoří pole 3x3x3, kde se dané hodnoty na původní kostce zapíšou nějakým číslem, které po zadání do funkce měnící atributy textu změní barvu textu na barvu odpovídají barvě nálepky na reálné kostce (kromě oranžové, která ve Windowsácké konzoli není)). Takové vypisování natvrdo zároveň funguje optimalizačně – nemusí se volat další funkce, které by podle nějakého pseudo-vzoru vypisovala nějaké věci.

Zároveň jsem vytvořil funkci *void tisk2(char kostka[5][5][5])*, která místo barev tiskne písmenka reprezentující danou barvu, díky čemuž můj program lze spustit pod jinými OS než Windows po přejmenování této funkce na název primární funkce pro tisk. Návod na to je na prvních řádcích programu v komentáři.

Dále jsem vytvořil funkci *void generateRandomMoves(char kostka[5][5][5], int delka)*, která generuje náhodný sled zadaného počtu tahů a kostku podle toho rozloží. Při generaci náhodného tahu se vybírá z tahů R, U, F, L, D a B (ze kterých lze vytvořit libovolné rozložení kostky, ale z praktických důvodů (potřeboval jsem implementovat i rotaci kostky) jsem u *void tah* bral jako elementární tahy i M, E a S) a z čísel 1 – 3, které značí, kolikrát se má vybraný tah provést. Pamatuje si to i 2 poslední tahy, aby se zabránilo tomu, že se udělá vícekrát po sobě nějaký tah (např. R R‘ R2) nebo kombinace jako U D2 U‘.

Další funkce *void prectiScramble(char kostka[5][5][5])* dokáže přečíst sled tahů k rozložení kostky a provést dané tahy (není to zas tak naprosto triviální, protože např. tah R‘ musí interpretovat jako tah ‚R‘ a počet, kolikrát se má udělat, jako 3.

Na začátku programu se po inicializaci kostky vypíše úvodní text s instrukcemi k programu (*void uvodniRecicky()*) a spustí funkce *void zacatek(char kostka[5][5][5]*), která uživateli nabídne, zda si chce nechat vygenerovat scramble, nebo ho zadat sám, a podle toho zavolá patřičnou funkci.

Před popisem samotných funkcí, které skládají jednotlivé části kostky, ještě popíšu několik pomocných funkcí, které se budou hodit:

int delejTahDokud(char kostka[5][5][5], char t, char kosticka, char souradnice1, char souradnice2, char souradnice3) – jediná funkce „prohledávání“. Funkce vrací počet tahů *t* (*t* je char - označení tahu), které je potřeba udělat, aby *kosticka* (čili nějaká konkrétní nálepka) byla na zadaných souřadnicích. Funkce prostě zkouší udělat tah i-krát (i je od 0 do 3) a potom se vrátit zpět (na což se hodí dělání tahů s argumentem vypsat = 0), a v případě, že se po tomto počtu tahů nachází daná nálepka na zadaných souřadnicích, tak vrátí, kolikrát se tento tah má provést, v případě, že se daná nálepka po libovolném počtu tahu t nemůže nacházet na daných souřadnicích, vrátí -1 (což byl důležitý debugovací nástroj).

souradnice najdiSouradnice(char kostka[5][5][5], char kosticka) – vrací strukturu *souradnice* (což je jenom struktura obsahující 3 souřadnice), na které se nachází daná *kosticka* (čili konkrétní nálepka).

char hodnotaHrany(char kostka[5][5][5], char kosticka) – vrací hodnotu *kosticky*, která je na stejné hraně jako zadaná *kosticka* (hodí se u zjišťování poloh *kosticek* během vkládání hran u skládání 2. vrstvy).

void tiskBarvyDilku(char kosticka) – funkce přidaná později, je používaná u vypisování řešení, aby uživatel věděl, který dílek chce program aktuálně dát na správné místo.

Několik proměnných, se kterými se budeme setkávat po celou dobu skládání:
char flag[] – pole flagů, jednotlivé flagy dostanou hodnotu 1, pokud je daná *kosticka*(čili nálepka na kostce) na své správné pozici. S tím souvisí i
int pocety – číslo, které returnují funkce na skládání kostky (a všechny kromě *int slozKriz* ji mají jako argument). Jde o to, že na začátku jsou středy dané barvy (ty mají vzájemnou polohu vždy stejnou) tak, jak na obrázku na začátku. Ale potom během skládání se otáčí s kostkou, prakticky pouze podle osy y (otáčení podle osy x se nachází v jednom algoritmu v posledním kroku, ale tam se během algoritmu kostka otočí po ose x zpátky). A po provedení několika tahů se testuje, zda jsou dané kostičky na správných místech (a podle toho nastavují *flagy* na 1). Což by moc nešlo, kdyby pokaždé byla kostička otočená jinak. Proto se před testováním na *flagy* otočí kostka (bez vypisování tohodle tahu) v závislosti na *pocety* tak, aby středy byly na stejných souřadnicích jako na začátku, a na začátku provádění série tahů se zase kostka otočí zpátky.
char i, j, k jsou iterátory.
int token – taková pomocná proměnná, která většinou slouží k zapamatování si, kolikrát nějaký konkrétní tah potom musíme udělat znovu, jeho využití je vždycky napsáno v komentáři.

Funkce na složení jednotlivých částí kostky: (jo, po skončení každé z těchto funkcí se vytiskne stav kostky)

int slozKriz(char kostka[5][5][5]) – funkce, která složí bílý kříž (program vždycky začíná od bílé barvy).
Nejdříve (*//posunuti prvni dobre hrany na bile stene*) program zkouší, zda lze vložit na správné místo nějakou bílou hranu pomocí tahu D. (během skládání máme stranu s bílým středem dole)
Potom v případě, že nějaká bílá hrana je v dolní vrstvě, ale její druhá barva (čili červená, zelená, oranžová nebo modrá) sousedí se středem jiné barvy, tak otočí kostku, abychom měli před sebou (ale pořád v dolní vrstvě) onu hranu, a provede několik tahů, aby se tato hrana dostala na správné místo.
Potom vkládá bílé hrany, které se nacházejí v horní vrstvě. (stejně jako v kroku předtím pomocí zkoušení souřadnic horní vrstvy, a pokud hodnota hrany na této souřadnici je mezi 1 a 4 (což jsou bílé nálepky, které se nachází na hranách), tak vytiskne barvu dané hrany, zjistí identifikační číslo nálepky, která je na stejné hraně (ale není bílá), zjistí, kolikrát otočit kostku u provést tah U, aby se hrana dostala ke středu barvy nálepky (která není bílá), a pomocí tahu F2 tuto hranu vloží)
Potom vkládá bílé hrany, které se nachází v prostřední vrstvě. Je to trochu komplikovanější, protože se najde střed, který má stejnou barvu jako nálepka na námi vkládané hraně (ale ne ta bílá), potom zjistí, kolikrát je potřeba udělat tah D, aby hrana, která je přímo pod

nalezeným středem, se dostala na pozici [1][4][2], a potom udělají určité tahy, jejichž počet je závislý na právě zjištěných skutečnostech.
Nakonec se vkládají bílé hrany z 1. a 3. vrstvy, podobným způsobem jako ty z prostřední vrstvy, protože stačí udělat tah F nebo F‘, a dostaneme se s touto hranou do prostřední vrstvy. Ale je potřeba si pamatovat, zda jsme tímto tahem náhodou nevyhodili bílou hranu, která už leží na správném místě. A taky jsem nechal rozlišit případ, kdy tah F nebo F‘ by posunul hranu na levou stranu a nálepka na této hraně (jiná než bílá) by sousedila se středem stejné barvy, tak v tom případě se provede ten tah F nebo F‘, který tu hranu posune nalevo, v opačném případě se to klasicky posune napravo.

int vlozRohy(char kostka[5][5][5], int pocety) – vkládá bílé rohy na správná místa.
V prvním vnořeném for cyklu vyhledá bílé rohy, kde bílá nálepka je v 3. vrstvě, a podle toho, jestli je v pravém horním nebo levém horním rohu, ji pomocí algoritmu vloží na správné místo.
V druhém vnořeném for cyklu vyhledá bílé rohy z horní vrstvy, přesune je mezi středy barev takových, co jsou na dané hraně (kromě bílé) a pomocí algoritmu vloží
V posledních 2 vnořených for cyklech to najde rohy, které jsou vložené ve špatném slotu (tzn. Rohy, které mají bílou v dolní nebo 1. vrstvě), vysune je z toho slotu a přejde do těla prvního vnořeného for cyklu (tzn. převede to na již vyřešený případ).

int vlozHrany(char kostka[5][5][5], int pocety) – vkládá všechny ostatní nežluté hrany na správná místa (takže skládá 2. vrstvu).
Nejdříve se to dívá po (nežlutých) hranách, které se nachází v 3. vrstvě a vkládá je podle vzájemné polohy středu nálepky, která je v 3. vrstvě, a nálepky, která je v horní vrstvě (2 zrcadlově stejné algoritmy).
Potom se to podívá, zda se v 2. vrstvě nachází hrana, která je vložená, ale naopak. V tomto případě ji vyndá (resp. tam vloží nějakou jinou hranu) a přejde do těla minulého vnořeného for cyklu (převede na minulý případ).

int OLL(char kostka[5][5][5], int pocety) – orientuje horní vrstvu ve 2 krocích.
V prvním kroku se orientují hrany. Existují 4 případy, jakou můžou mít žluté hrany pozici:
1) ani jedna hrana nemá žlutou nálepku v horní vrstvě
2) v horní vrstvě jsou 2 žluté nálepky naproti sobě
3) v horní vrstvě jsou 2 žluté nálepky vedle sebe
4) všechny 4 nálepky v horní vrstvě jsou žluté
V případě 4) není potřeba nic dělat, pro případy 2) a 3) existují jednoduché algoritmy, a případ 1) se řeší pomocí algoritmu pro případ 2) a algoritmu pro případ 3). korát u případů 2) a 3) je ještě třeba několikrát provést tah U, aby se hrany dostaly do správné pozice.
V druhém kroku se udělá tah U tolikrát, aby nahoře v pravém dolním rohu nebyla žlutá nálepka, a opakuje se čtveřici tahů do doby, než tam nebude. Potom se zase udělá tah U atd. až všechny rohy budou mít žlutou nálepku v horní vrstvě.

int PLL(char kostka[5][5][5], int pocety) – provede několik algoritmů, které permutují dílky v 3. vrstvě tak, aby byly na správných místech.
Nedříve zjistí, kolik dvojic nálepek na rozích (vždy na stejné straně) je stejných (pokud je kostka složená, tak všechny 4), podle toho (pokud jich je méně než 4) buď provede

algoritmus zvaný „A perm“ (ten prohazuje pozice 3 rohů), anebo najde, na kterém místě je správná dvojice nálepek, udělá tolik tahů U, aby se tato nacházela vzadu, a udělá „A perm“. Občas ho bude muset udělat 2krát. Tímto způsobem budou všechny rohy správně zpermutované.
Potom se provede U tak, aby nálepky na rozích barevně ladily s nálepkami na hranách, které jsou pod nimi.
Nakonec si zjistí, kolik hran v 3. vrstvě má nálepku, která ladí s nálepkami sousedních rohů. Ve *flaghrany* si ukládá, kolikrát musí udělat tah U, aby hrana ladící s okolními rohy byla na přední stěně. V případě, že ani jedna hrana neladí s rohy, tak se provede „U perm“ (a zapíše se hodnoty flaghrany), a potom se (každopádně) provede tah U *flaghrany*-krát a (pokud je pouze 1 hrana ladící s rohy) provede se „U perm“ (občas se bude muset provést 2krát). A kostka je složena.

Poté se program zeptá uživatele, zda chce skládat znova, a buď pojede od začátku, nebo skončí.