Procedurální programování

Dynamické datové struktury **1**
- Organizace dat v paměti RAM 1
- Struktury 1
- Operátory new a delete 1
- Vícerozměrná a dynamická pole 2

Práce se soubory **2**
- Soubory pomocí standardní knihovny C 2
- Soubory pomocí knihovny STL 3

Výpočty s čísly **3**
- Číselné datové typy 3
- Lokální a globální proměnné 4
- Operátory 4
- Matematické funkce 5
- Generování náhodných čísel 5

Práce s textem **5**
- C style strings 5
- STL strings 5
- Funkce pro práci se C style strings 6
- Funkce pro práci s STL strings - popis jednotlivých funkcí 6

Moduly v jazyce C++ **6**
- Význam rozhraní a implementace v C++ 6
- Oddělený překlad modulů 6
- Jmenné prostory 7

Dynamické datové struktury

Organizace dat v paměti RAM

1. Diagram rozložení prvků v paměti RAM a popis:
    a. Popis code bloku - zde je binární kód programu
    b. Data blok - zde jsou statická data, tedy deklarace globálních proměnných a static proměnných
    c. Halda, neboli dynamická paměť - slouží k alokaci prvků proměnlivé délky, nebo délky, známé až v runtime, nebo velkých prvků - operátory **new** a **delete**, předbíháme téma.
    d. Zásobník - pro automatické proměnné, tedy lokální proměnné ve funkcích a další údaje funkcí jako návratová adresa apod., jde adresově proti haldě
2. Předchozí bod je zdánlivé rozložení v adresovém prostoru, NE v paměti RAM.
3. Každý proces má vlastní adresový prostor.
4. Alokovaná paměť ve virtuálním adresovém prostoru rozdělena na stránky.
5. Stránky přes správce paměti OS přidělovány do rámců paměti RAM, rámec je prostor pro umístění stránky v RAM.
6. Stránky mohou být pak odloženy do odkládacího prostoru na disku.
7. Odkaz na Wikipedii na téma stránkování OS.

Pozn.: Titulek článku je záměrné zjednodušení, o existenci virtuálního adresového prostoru se čtenář dozví až na konci tohoto článku.

Struktury

1. Struktura - je uživatelsky definovaný datový typ k vyjádření složených dat. Slouží ke kombinaci různých datových typů.
2. Obsahují členské proměnné.
3. Ukázka definice struktury a popis.
4. Přístup k položkám a definice na zásobníku. Přístup přes operátor “tečka”.
5. Předávání struktur do funkcí hodnotou výchozím kopírovacím konstruktorem.
6. Ukazatele na struktury - přístupu položkám přes operátor ->
7. Předávání pomocí operátoru & ze zásobníku, preferovaný přístup kvůli zbytečnému kopírování hodnot. Ukázka tohoto přístupu.
8. Zarovnání struktur v paměti na adresu a velikosti, tedy velikost struktury může být větší než součet velikosti položek. Optimalizace kvůli přístupu.
9. Zásobník určen jen pro malé struktury, pro velké struktury => alokace na dynamické paměti - viz následující kapitola.

Operátory **new** a **delete**

1. Operátor **new**, vytvoření alokovaného prostoru na haldě.
    a. Důvod: často není známá velikost dat, až v runtime.

    b. Další důvod: Příliš velká velikost pro zásobník.
2. Alokace operátoru **new** vrací ukazatel na alokovanou oblast paměti.
3. Uvolnění pomocí **delete** operátoru s uvedením ukazatele, tedy předáním adresy na počátek alokované paměti.
4. Alokace polí na haldě, uvolnění operátorem **delete** [ ].
    a. Příklad a popis.
5. Alokace struktur na haldě a jejich uvolnění.
    a. Příklad a popis.
6. Opomenutí **delete**, vede k memory leakům.

Vícerozměrná a dynamická pole

1. Vícerozměrná pole
    a. souvislý blok paměti
    b. konstantní velikost všech položek
    c. Alokace najednou.
    d. Diagram alokovaného pole
2. Dynamické pole
    a. Nesouvislý blok paměti
    b. Proměnlivá velikost položek.
    c. Alokace každé položky zvlášť.
    d. Diagram dynamického pole.
3. Příklad na alokaci vícerozměrného pole.
    a. int (*)[**dimenze**] ukazatel
    b. a delete[ ]
    c. **dimenze** musí být konstantní výraz
    d. přístup pomocí operátoru indexování
    e. ukázka pointerové aritmetiky pro vícerozměrná pole
4. Příklad na postupnou alokaci dynamického pole
    a. int** ukazatel
    b. postupné uvolnění
    c. velikost dalších rozměrů nemusí být konstantní výraz
    d. přístup pomocí operátoru indexování
    e. ukázka pointerové aritmetiky pro dynamické pole

Práce se soubory

Soubory pomocí standardní knihovny C

1. otevření souboru fopen(...) popis parametrů
    a. zápis a čtení
    b. vytvoření nového či existující
    c. binární nebo textový režim
2. kontrola korektnosti otevření souboru: ferror
3. zápis a čtení formátovaného vstupu a výstupu: fprintf, fscanf

    a. popis základních formátů čtení a zápisu
4. čtení a zápis znaků: fputc a fgetc
5. čtení a zápis binárních dat pomocí bufferů: fread, fwrite
6. získání změna pozice v souboru pomocí fseek, ftellp
    a. od začátku, od konec, od současné pozice
7. zavření souboru fclose(...)

Soubory pomocí knihovny STL

1. otevření souboru ifstream, ofstream, fstream
    a. čtení a zápis
    b. vytvoření nového či existujícího
    c. binární či textový režim
2. kontrola korektnosti operací: good, eof, fail, bad
3. zápis a čtení formátovaný: operátory << a >>
4. čtení a zápis znaků: put() a get() a test dalšího znaku pomocí peek(...)
5. čtení a zápis binárních dat: read, write
6. získání a změna pozice kurzoru: tellg, seekg, tellp, seekp - rozdíl mezi pozicí čtení a zápisu
7. vyprázdnění bufferu: flush(...)
8. zavření souboru: automatickým uvolněním objektu

Výpočty s čísly

Číselné datové typy

1. rozdílnost datových typů char, signed char a unsigned char
2. dále pak wchar_t pro Unicode znaky
3. číselné datové typy
    - short int
    - int
    - long int
    - long long int
4. atribut znaménkovosti
    - signed a unsigned
5. čísla s pohyblivou řádovou čárkou
    - float
    - double
    - long double
6. standard C++ nedefinuje rozsahy těchto čísel, pouze, které jsou větší a menší
    - tabulka porovnání těchto datových typů: zdroj Stroustrup nebo C++ standard
7. zkratky datových typů
    - short, long, long long
    - unsigned
8. pokud je zájem o pevně definovanou bitovou velikost

- je zde knihovna <cstdint> a datové typy
- tabulka: uint8_t, int8_t, 16, 32, 64, apod.

9. číselné rozsahy datových typů, knihovna **<climits>**
    - INT_MAX, INT_MIN, UINT_MAX, LONG_MIN, LONG_MAX, ULONG_MAX
    - apod. tabulka

Lokální a globální proměnné

1. rozdíl mezi automatickými, globální a statickými proměnnými
2. statická proměnná je v podstatě charakterem globální proměnná
    - tedy hodnota přežívá opuštění těla funkce
    - je dostupná ale pouze ve scope, kde je deklarována
3. lokální proměnná
    - její platnost končí blokem, kde je deklarována
4. globální proměnná
    - má platnost neustále
    - lze ji adresovat pomocí operátoru pro globální jmenný prostor ::
        - vhodné při konfliktu jmen
5. stejně pojmenovaná lokální proměnná má přednost před globální
    - zastínění globální proměnné
    - zastínění lokální proměnné ve stejné funkci
    - nemělo by se používat

Operátory

1. odkazy na dřívější články
    1. Aritmetické operátory
    2. Relační operátory
    3. Logické operátory
2. doplnění následujících operátorů:
    1. ternární operátor ? :
        - přepis tohoto operátoru pomocí if klauzule pro názornost
    2. bitové operátory:
        - and
        - or
        - xor
        - not - bitový doplněk
        - shift vlevo
        - shift vpravo (rozdíl pro typy signed a unsigned)
    3. složené přiřazovací operátory pro bitové operátory:
        - jen tabulka: and, or, xor, left shift, right shift
    4. operátory přetypování:
        - static_cast
        - reinterpret_cast
        - dynamic_cast - popis v budoucnu u OOP, jen stručný popis
        - const_cast

Matematické funkce

- **Popis následujících funkcí**
    1. hlavičkový soubor <cmath>
    2. abs, fabs
    3. floor, round a ceil
    4. lround a llround
    5. pow a sqrt
    6. log, log10, exp
    7. sin, cos, tan
    8. asin, acos, atan

Generování náhodných čísel

1. knihovny: <ctime> (funkce time()) a <cstdlib> (zbývající funkce kolem náhodných čísel)
2. funkce srand
3. funkce time, počet sekund od 1.1.1970
4. funkce rand
5. konstanta RAND_MAX
6. ukázka generování náhodných čísel v nějakém intervalu
    a. první od nuly až KONSTANTA
    b. druhé OD … DO

Práce s textem

C style strings

1. Datový typ char a wchar_t, rozdíl mezi ASCII a UNICODE, odkaz na to, co je UNICODE
2. Řetězec musí být ukončen nulovým znakem, proto v polích o jeden znak na konci navíc.
3. Znak v apostrofech, řetězec je v uvozovkách
4. Předávání stringů do funkcí, char*, a pro literály musí být const char*
5. Escape znaky - co jsou, proč jsou a jmenovat ty hlavní
6. Indexování pomocí operátoru pole, není kontrola rozsahu

STL strings

1. Datový typ string a wstring, rozdíl mezi ASCII a UNICODE, odkaz na to, co je UNICODE
2. Inicializace konstruktorem: literál anebo jiný string
3. Inicializace operátorem přiřazení, literál anebo jiný string
4. Spojování textů pomocí operátorů + a +=

5. Funkce length() a size() - synonyma
6. Indexování pomocí operátoru pole, není kontrola indexu kvůli rychlosti přístupu
7. existuje ale funcke **.at()**, kde je runtime kontrola indexu

Funkce pro práci se C style strings

1. vyskytuje se v hlavičce <cstring>
2. strcpy a strncpy
3. strcat a strncat
4. strcmp a strcoll
5. strchr a strstr
6. strtok
7. strlen

Funkce pro práci s STL strings - popis jednotlivých funkcí

1. substr
2. find
3. replace
4. push_back a pop_back
5. c_str()
6. odkaz na c++ referenční manuál pro další funkce

Moduly v jazyce C++

Význam rozhraní a implementace v C++

1. Rozhraní a implementace
    a. v C++ **má obecnější význam** než v klasickém chápání OOP
2. Rozhraní říká CO se dělá, implementace říká JAK se to dělá
3. Rozhraní má říkat klientovi implementace jasnou formou, jak implementaci použít bez toho, aniž by musel zkoumat, jak daná implementace vnitřně funguje, proto nutná vhodná volba i identifikátorů reprezentujcích dané rozhraní
4. Můžeme konkretizovat na:
    a. hlavičkový soubor je rozhraním pro CPP modul, který je jeho implementací
    b. rozhraní interface (implementace pomocí abstraktní třídy) je rozhraní třídy, která je její implementací
    c. veřejné metody třídy jsou rovněž rozhraní třídy, třída je jejich implementací, základní princip zapouzdření OOP, soukromé a chráněné metody a prvky jsou implementací

Oddělený překlad modulů

1. rozdělení definice a deklarací
    a. sdílené deklarace do hlavičkových souborů

    b. soukromé deklarace a definice do souborů CPP
2. hlavičkový soubor představuje **rozhraní** modulu
3. soubor CPP je **implementací** hlavičkového souboru
4. hlavičkové soubory jsou navazovány přes deklaraci **#include**
    a. má sloužit k použití jazykových entit některého z modulů
5. nejlepší přístup - maximalizovaná modularita
6. postup překladu
    a. preprocesor a překladač
    b. linker
7. provádí se linkování nejen
    a. přeložených objektových souborů
    b. ale i různých knihoven .LIB či .A

Jmenné prostory

1. Použití
    a. lokalizace platnosti identifikátorů z různých knihoven,
    b. bezpečnost pojmenovávání identifikátorů, není tedy nebezpečí jmenných konfliktů
    c. organizace větších programových celků
2. **namespace** deklarace, identifikátor a content ve složených závorkách
3. **using namespace** *název* - import názvů do globálního jmenného prostoru + definice co je globální jmenný prostor
4. plná kvalifikace identifikátorů
5. vnořené prostory jmen
    a. plná kvalifikace identifikátoru z vnořeného prostoru jmen
    b. **using namespace** deklarace vnitřního prostoru
    c. vysvětlit princip funkce zastínění identifikátorů
    d. automatické odkazování na identifikátory z vnějších jmenných prostorů, pokud nejsou zastíněny
    e. plná kvalifikace identifikátoru z globálního jmenného prostoru uvnitř jiného jmenného prostoru