1.1 C ++ v / s Java

C ++ jest jednym z najpotężniejszych języków i będzie używany przez długi czas w przyszłości pomimo pojawienia się Javy. C ++ działa bardzo szybko i jest 10 do 20 razy SZYBSZY niż Java. Java działa bardzo wolno, ponieważ jest językiem interpretowanym w języku bajtowym, działającym na "maszynie wirtualnej". Java działa szybciej z kompilatorem JIT (Just-In-Time), ale wciąż jest wolniejsza niż C ++. Zoptymalizowany program C ++ jest około 3 do 4 razy szybszy niż Java (z kompilatorem JIT). Dlaczego ludzie używają Javy? Ponieważ jest zorientowany obiektowo i jest łatwiejszy do programowania w Javie, ponieważ Java automatyzuje zarządzanie pamięcią, a programiści nie zajmują się bezpośrednio przydzielaniem pamięci. Ten dokument próbuje zautomatyzować zarządzanie pamięcią w C ++, aby uczynić go znacznie prostszym w użyciu. Biblioteka podana tutaj sprawi, że C ++ będzie wyglądać jak Java i umożliwi "C ++" konkurować z językiem Java. Ze względu na ręczne przydzielanie pamięci, debugowanie programów w C ++ zajmuje znaczną część czasu. W tym dokumencie znajdziesz lepsze pomysły i wskazówki, które pozwolą skrócić czas debugowania.

1.2 Który z języków Ada95, "C", "C ++" lub Java?

Wybór języka jest bardzo trudny. Istnieje zbyt wiele parametrów - ludzie, umiejętności ludzi, koszty, narzędzia, polityka (nawet polityka narodowa) i wpływ przedsiębiorców / firm komercyjnych. Najlepszy język oparty na zaletach technicznych nie zostaje wybrany tylko ze względu na decyzje polityczne! Java jest znacznie bliższa Ada95 niż C ++. Java pochodzi od Ada95. Ada95 otrzymuje maksymalną liczbę punktów według tabeli porównawczej Ada Davida Wheelera. Ada uzyskała 93%, Java 72%, C ++ 68%, a C - 53%. C ++ i Java są bliższe punktom (tylko 4% różnicy), stąd Java nie jest wielką rewolucją w porównaniu z C ++ , Ada to bardzo duża rewolucja i ulepszenie w stosunku do C ++. Wyniki są jak 4 studentów przystępujących do egzaminów, a student z najwyższym wynikiem to Ada (93%). Kto wie? Być może w przyszłości Ada95 zastąpi Javę !! Koszty rozwoju Ada to połowa C ++, jak na Stephena F. Zeiglera. Ada95 jest dostępny na stronie - · Ada home http://www.gnuada.org. · Indeks Google Ada Ponieważ programistów C ++ jest wiele, zaleca się programowanie w obiektowym "C ++" dla całego programowania aplikacji lub programowania ogólnego. Możesz w pełni korzystać z obiektów zorientowanych obiektowo C ++. Kompilator C ++ jest dużo bardziej skomplikowany niż kompilator "C", a programy w C ++ mogą działać nieco wolniej niż programy "C". Ale różnica prędkości pomiędzy "C" i "C ++" jest bardzo mała - może to być kilka milli-sekund, które mogą mieć niewielki wpływ na programowanie w czasie rzeczywistym. Ponieważ sprzęt komputerowy staje się coraz tańszy i szybszy, a pamięć RAM jest coraz szybsza i tańsza, warto robić kod w C ++, a nie w "C", ponieważ czas zapisany w przejrzysty sposób, a ponowne użycie kodu C ++ przesuwa w zwolnionym tempie. Opcje optymalizatora kompilatora, takie jak -O lub -O3, mogą przyspieszyć działanie C ++ / C, które nie jest dostępne w Javie. W dzisiejszych czasach język "C" jest używany przede wszystkim do "programowania systemów" w celu opracowania systemów operacyjnych, sterowników urządzeń itp.

Uwaga: Za pomocą klas String, StringBuffer, StringTokenizer i StringReader podanych w tym podręczniku można kodować w C ++, który "dokładnie" wygląda jak Java. Ten dokument próbuje zlikwidować lukę między C ++ i Java, poprzez naśladowanie klas Java w C ++ Java jest językiem niezależnym od platformy, bardziej odpowiednim do rozwijania GUI działającego w przeglądarkach internetowych (aplety Java), ale działa bardzo wolno. Wolisz używać programowania po stronie serwera "Fast-CGI" z C ++ i HTML, DHTML, XML, aby uzyskać lepszą wydajność. Stąd złotą zasadą jest "programowanie po stronie serwera WWW za pomocą C ++ i programowanie po stronie klienta (przeglądarki) za pomocą apletów Java". Powód jest taki: system operacyjny po stronie serwera (Linux) jest pod kontrolą i nigdy się nie zmienia, ale nigdy nie dowiesz się, jaki jest system operacyjny przeglądarki WWW po stronie klienta. Może to być urządzenie urządzenia internetowego (wbudowany linux + netscape) lub komputery z systemem Windows 95/98 / NT / 2000 lub Linux, Apple Mac, OS / 2,

Netware, Solaris itp. Zaletą języka Java jest możliwość tworzenia " Aplety (GUI) ", które mogą działać na dowolnej platformie klienta. Java została stworzona w celu zastąpienia interfejsów API systemu Microsoft Windows 95 / NT, takich jak MS Visual Basic lub MS Visual C ++. Innymi słowy - "Java jest wieloplatformowym językiem interfejsów Windows GUI w następnym stuleciu". Wiele przeglądarek internetowych, takich jak Netscape, obsługuje aplety Java, a przeglądarka internetowa, taka jak Hot Java, jest napisana w samym języku Java. Ale cena, którą płacisz za przenoszenie na różne platformy to wydajność, aplikacje napisane w Javie działają bardzo wolno. W związku z tym Java działa na "kliencie", a C ++ na serwerach.

1.3 Problemy stojące przed obecnymi kompilatorami C ++

Ponieważ C ++ jest super-zbiorem C, ma wszystkie złe cechy języka "C". Ręczne przydzielanie i zwalnianie pamięci jest uciążliwe i podatne na błędy. W programowaniu "C" - przecieki pamięci, przepełnienia pamięci są bardzo powszechne ze względu na użycie takich funkcji jak - typ danych char * i char [] Funkcje łańcuchowe takie jak strcpy, strcat, strncpy, strncat itd. Funkcje pamięci takie jak malloc, realloc, strdup , itp. Używanie char * i strcpy powoduje straszne problemy z pamięcią spowodowane "przepełnieniem", "pomyłkami z przeszłości", "uszkodzeniem pamięci", "wkroczeniem na inny palec" (zranieniem lokalizacji pamięci innych zmiennych) lub " wycieki pamięci ". Problemy z pamięcią są bardzo trudne do debugowania i są bardzo czasochłonne do naprawy i rozwiązywania problemów. Problemy z pamięcią obniżają produktywność programistów. Dokument ten pomaga zwiększyć produktywność programistów za pomocą różnych metod rozwiązanych w celu usunięcia wad pamięci w "C ++". Błędy związane z pamięcią są bardzo trudne do złamania, a nawet doświadczeni programiści zajmują kilka dni lub tygodni w celu debugowania problemów związanych z pamięcią. Błędy pamięci mogą być ukryte w kodzie przez kilka miesięcy i mogą powodować nieoczekiwane awarie programu. Błędy pamięci spowodowane używaniem znaków char * i wskaźników w C / C ++ kosztują rocznie 2 miliardy dolarów w czasie utracone z powodu debugowania i przestojów programów. Jeśli używasz char * i wskaźników w C ++ to bardzo kosztowna sprawa, szczególnie jeśli rozmiar twojego programu przekracza 10 000 linii kodu. W związku z tym proponuje się następujące techniki w celu przezwyciężenia wad języka "C". Daj pierwszeństwo w następującej kolejności -

1. Użyj odnośników zamiast wskaźników.

2. Klasa String w stylu Java (podana w tym temacie) lub klasa ciągu STDLib.

3. Wskaźniki znaków (char *) w C ++ ograniczają użycie znaku char * do przypadków, w których nie można używać klasy String.

4. Wskaźniki znaków (char *) w C używające specyfikacji łącza zewnętrznego, jeśli nie chcesz używać (char *) w C ++.

Aby użyć "C char *", umieścisz wszystkie swoje programy "C" w osobnym pliku i link do programów "C ++" za pomocą instrukcji link-statement extern "C" -

```
extern "C" {

#include <stdlib.h>

}

extern "C" {

comp ();

some_c_function ();
```

}

extern "C" jest specyfikacją powiązania i jest flagą, że wszystko wewnątrz otaczającego bloku (otoczonego klatką) wykorzystuje połączenie C, a nie wiązanie C ++. "Klasa String" wykorzystuje funkcje konstruktora i destruktora do zautomatyzowania zarządzania pamięcią i zapewnia dostęp do takich funkcji, jak ltrim, substring, itp. Zobacz także powiązaną "klasę strun" w kompilatorze C ++. Klasa string jest częścią standardowej biblioteki GNU C ++ i zapewnia wiele funkcji manipulowania strunami. Ponieważ biblioteka C ++ 'string class' i 'String class' udostępnia wiele funkcji manipulowania ciągami, mniej jest potrzeby użycia podejścia do wskaźnika znaków w celu napisania własnych funkcji ciągu. Ponadto należy zachęcać programistów C ++ do używania operatorów "nowych", "usuwania" zamiast używania "malloc" lub "free". "Klasa ciągów" robi wszystko, co char * lub char []. Może całkowicie zastąpić typ danych char. Dodatkową korzyścią jest to, że programiści nie muszą w ogóle martwić się o problemy z pamięcią i alokację pamięci.

1.4 COOP - C ++ Object-based Programming-language

Problem z C ++ polega na tym, że jest to nadzbiór C, i chociaż programiści mogą używać dobrych (obiektowo zorientowanych) cech C ++ i unikać złych cech C, nie ma niczego, co zmusza ich do tego. Tak więc, wiele programów napisanych w C ++ jest napisanych bez obiektów zorientowanych obiektowo i nadal używa złych cech C, które powinno pokonać użycie C ++. Dlatego proponuję, abyśmy stworzyli nową wersję C ++, która nie pozwala na użycie złych cech C. Proponuję, aby ta nowa wersja C ++ była nazywana COOP (powiedzmy koop), która jest akronimem dla programowania obiektowego C ++ "COOP" powinno być wymawiane jak kurnik (logo COOP jest grubym kwojem wewnątrz)! Proponuję, aby rozszerzenie pliku COOP było .coo, które nie będzie kolidowało z .c dla programów C lub .cpp dla programów w C ++. Zacznijmy od napisania COOP jako interfejsu do C ++, czyli COOP wstępnie przetwarza składnię kodu, a następnie używa standardowego kompilatora C ++ do kompilacji programu. COOP działa jako front-end dla kompilatora C ++. (Na początek, COOP będzie bardzo dobrym tematem projektu / pracy magisterskiej dla studentów uniwersytetu) Poniżej przedstawiono kilka innych proponowanych cech COOP:

* COOP pożyczy kilka najlepszych pomysłów od Microsoft C #, Microsoft włożył wiele wysiłku, a ty możesz po prostu je wykorzystać. Specyfikacje znajdują się w specyfikacji csharp i zobacz przegląd C #.

* Jest podzbiorem języka C ++, ale zmusza programistę do używania programowania zorientowanego na przedmiot

* Czysty obiektowy język, ale zachowuje składnię C ++. Usuń wszystkie złe lub mylące funkcje C ++ w COOP, na przykład wielokrotne dziedziczenie, przeciążanie operatorów, ograniczanie użycia wskaźników itp.

* Zapobieganie pisaniu "C" jak programowanie w COOP, coś, co obecnie pozwala C ++. Usuń wszystkie funkcje C, które są uważane za złe lub nadmiarowe / duplikaty, takie jak printf, fprintf, malloc, struct, free etc ..

* Brak zgodności w dół z językiem "C".

* Kod napisany w COOP będzie łatwy w utrzymaniu i będzie łatwo zrozumiały / czytelny.

* Kod napisany w "COOP" będzie ponownie użyty (przez komponenty, moduły, obiekty). Obsługuje wielokrotnego użytku komponenty oprogramowania, ułatwiając Rapid Application Development.

* COOP jest prosty, solidny, OOP, ma nieistotną składnię mininum (unikanie mylących, nadmiarowych, dodatkowych konstrukcji C ++ dla np. Usunięcia klasy struct i use)

.

Pożyczyć pomysły od -

* Java - Sun Microsystem wkłada wiele wysiłku i możesz to po prostu wykorzystać.

* Connective C ++ na http://www.quintessent.com/products/cc++.

2. Odmiany klas string

Klasa string jest najważniejszym obiektem w programowaniu, a manipulacje łańcuchami są najszerzej stosowane i stanowią od 20 do 60% całego kodu. Istnieją 3 różne klasy ciągów. Oczywiście, możesz zbudować swoją własną klasę ciągów po prostu dziedzicząc z tych klas ciągów -

* Klasa String podana w tym dokumencie Dodatek A String.h

* Klasa String GNU

-GNU C ++ Library - Univ of Tech, Sydney

http://www.socs.uts.edu.au/doc/gnuinfo/libg++/libg++_18.html i podręcznik użytkownika

-lustrzana witryny Gesellschaft http://www-aix.gsi.de/doc/gnu/libg++_18.html#SEC23 i użytkownika

przewodnik

- lustrzana a strona Techno, Rosja

http://www.techno.spb.ru/~xbatob/FAQ/GNU/libg++_19.html#SEC27 i podręcznik użytkownika

- lustrzana strona Univ of Utah http://www.math.utah.edu/docs/info/libg++_19.html#SEC27 i poradnik użytkownika

* Klasa Qt String na http://doc.trolltech.com/qstring.html mirror at http://www.cs.berkeley.edu/~dmartin/qt/qstring.html

* Jeśli żadna z tych alternatyw nie jest odpowiednia, możesz zbudować własną klasę ciągów. Możesz zacząć od jednej lub więcej z wcześniej zbudowanych klas wymienionych powyżej (przy użyciu dziedziczenia pojedynczego lub wielokrotnego).

2.1 Wiele dziedziczenia - przykładowa niestandardowa klasa String

Jak wspomniano powyżej, możesz zbudować własną niestandardową klasę ciągów z klas wstępnie zbudowanych przez dziedziczenie pojedyncze lub wielokrotne. W tej sekcji zbudujemy przykładową niestandardową klasę łańcuchów, używając dziedziczenia wielokrotnego, dziedzicząc z klasy łańcuchowej GNU i klasy łańcuchów przedstawionej w Dodatku H. Zacznij od pobrania przykładowego pliku "string_multi.h" z Dodatku A. Plik ten jest odtworzony poniżej:

```
// ************************************************************
// Sample program to demonstrate constructing your own string class
// by deriving from the String class and stdlib's "string" class
// ************************************************************
#ifndef __STRING_MULTI_H_
```

```
#define __STRING_MULTI_H_
#include <string>
#include "String.h"
// Important Notes: In C++ the constructors, destructors and copy
// operator are NOT inherited by the derived classes!!
// Hence, if the operators like =, + etc.. are defined in
// base class and those operators use the base class's contructors
// then you MUST define equivalent constructors in the derived
// class. See the sample given below where constructors mystring(),
// mystring(char[]) are defined.
//
// Also when you use operator as in atmpstr + mstr, what you are really
// calling is atmpstr.operator+(mstr). The atmpstr is declared a mystring
class mystring:public String, string
{
public:
mystring():String() {} // These are needed for operator=, +
mystring(char bb[]):String(bb) {} // These are needed for operator=, +
mystring(char bb[], int start, int slength):String(bb, start, slength) {}
mystring(int bb):String(bb) {} // needed by operator+
mystring(unsigned long bb):String(bb) {} // needed by operator+
mystring(long bb):String(bb) {} // needed by operator+
mystring(float bb):String(bb) {} // needed by operator+
mystring(double bb):String(bb) {} // needed by operator+
mystring(const String & rhs):String(rhs) {} // Copy Constructor needed by operator+
mystring(StringBuffer sb):String(sb) {} // Java compatibility
mystring(int bb, bool dummy):String(bb, dummy) {} // for StringBuffer class
int mystraa; // customizations of mystring
private:
int mystrbb; // customizations of mystring
};
```

#endif // __STRING_MULTI_H_

3. Najlepsze kompilatory C ++ dla MS Windows 2000 / NT / 95/98 / ME / XP

Ponieważ MS Windows jest dość popularny w C ++, biblioteka klas łańcuchów podana w tym dokumencie działa dobrze i działa bardzo dobrze we wszystkich wersjach MS Windows, tj. MS Win XP / 2000 / NT / 95/98 / ME. Kompilatory C ++ dla MS Windows to:

* GNU BloodShed na http://www.bloodshed.net/devcpp.html oceniony jako pierwszy (najlepszy spośród wszystkich)

* Kompilator Borland C ++ http://www.borland.com/bcppbuilder/freecompiler oceniono jako drugi

* Kompilator Microsoft Visual C ++ http://msdn.microsoft.com/visualc oceniono jako trzeci

* Kompilator MSDOS C ++ http://www.delorie.com/djgpp

Klasa String w tym dokumencie jest testowana ze wszystkimi powyższymi kompilatorami. Działa dobrze z kompilatorem MS Visual C ++ v6.0, Borland C ++ v5.2, Borland C ++ kompilatorem v5.5.1 i kompilatorem Bloodshed.

4. Pobierz string

Wszystkie programy, przykłady podano w załączniku do niniejszego dokumentu. Możesz pobrać jako pojedynczy plik ZIP, klasę String, biblioteki i przykładowe programy z · Przejdź tutaj i kliknij C ++ Programowanie howto.tar.gz plik http://www.aldev.8m.com Strony lustrzane znajdują się na stronie - http://aldev0.webjump.com, angelfire, geocities, virtualave, 50megs, theglobe, NBCi, Terrashare, Fortunecity, Freewebsites, Tripod, Spree, Escalix, Httpcity, Freeservers.

5. W jaki sposób mogę ufać klasie String Al Dev?

Możesz mieć pytanie o błędne zaufanie w oprogramowaniu klasy String. Aby zbudować pewność, istnieje metoda naukowa sprawdzania funkcjonalności klasy String Al Dev. W dzisiejszych czasach informatycy używają mocy procesora zamiast mocy ludzkiego mózgu do weryfikacji i sprawdzania poprawności oprogramowania. Ludzki mózg jest zbyt wolny i dlatego lepiej jest używać mocy komputera do testowania i sprawdzania poprawności oprogramowania. Program example_String.cpp ma moduł testu regresji, który możesz używać do automatycznego uruchamiania testów regresji kilka milionów razy. Po uruchomieniu testów regresji na klasie String można poświadczyć, że program klasy String jest programem ROCK SOLID i BULLET-PROOF. Przetestowałem klasę String z cyklem powtarzania = 50000 i uruchomiłem program bez awarii. Podczas działania nie zauważyłem żadnego wycieku pamięci. Na Linuksie użyłem / usr / bin / gtop, polecenia unix top,

KDEStart-> System-> System KDE Gaurd

i

KDEStart-> System-> Zarządzanie procesami do monitorowania wykorzystania procesora i pamięci.

Zalecam rozpoczęcie testu regresji z powtarzaniem cyklu równym 10 milionów lub więcej. Im większa liczba powtórzeń, tym większa będzie Twoja pewność !! Rozpocznij test i idź na lunch i wróć, aby zobaczyć wyniki!

6. Wykorzystanie klasy String

Aby użyć klasy String, najpierw powinieneś zapoznać się z przykładowym programem "example_String.cpp" podanym w Dodatku A i klasą String podaną w Dodatku A. "Klasa String" jest

kompletnym zamiennikiem char i char * typ danych. Możesz użyć "klasy String" tak jak char i uzyskać znacznie więcej funkcjonalności. Powinieneś połączyć się z biblioteką 'libString.a', którą możesz zbudować z pliku makefile podanego w Dodatku A i skopiować bibliotekę do katalogu / usr / lib lub / lib, gdzie znajdują się wszystkie biblioteki "C ++". Aby użyć "libString.a" skompiluj swoje programy, takie jak -

g ++ example.cpp -lString

Zobacz przykładowy kod przykładowy jak podano poniżej -

String aa;

aa = "Creating an Universe is very easy, similar to creating a baby human.";

// You can use aa.val() like a 'char *' variable in programs

for (unsigned long tmpii = 0; tmpii < aa.length(); tmpii++)

{

//fprintf(stdout, "aa.val()[%ld]=%c ", tmpii, aa.val()[tmpii]);

fprintf(stdout, "aa[%ld]=%c ", tmpii, aa[tmpii]);

}

// Using pointers on 'char *' val ...

for (char *tmpcc = aa.val(); *tmpcc != 0; tmpcc++)

{

fprintf(stdout, "aa.val()=%c ", *tmpcc);

}

6.1 Operatory

klasa String zapewnia następujące operatory:

* Równy ==

* Nie równe ! =

* Przypisanie =

* Dodaj do siebie i Przypisz + =

* Łączenie ciągów lub dodawanie +

Na przykład, aby użyć operatorów –

String aa;

String bb("Bill Clinton");

aa = "put some value string"; // assignment operator

aa += "add some more"; // Add to itself and assign operator

aa = "My name is" + " Alavoor Vasudevan "; // string cat operator

if (bb == "Bill Clinton") // boolean equal to operator

cout << "bb is equal to 'Bill Clinton' " << endl;

if (bb != "Al Gore") // boolean 'not equal' to operator

cout << "bb is not equal to 'Al Gore'" << endl;

6.2 Funkcje

Funkcje udostępniane przez klasę String mają taką samą nazwę, jak klasa String języka Java. Nazwy funkcji i zachowanie są dokładnie takie same jak w klasie String języka Java. Dostępna jest również klasa StringBuffer. Ułatwi to przenoszenie kodu między Javą i C ++ (możesz wycinać i wklejać i wprowadzać minimalne zmiany w kodzie). Kod z treści funkcji Javy może być skopiowany do treści funkcji C ++, a przy bardzo małych zmianach kod skompiluje się pod C ++. Kolejną zaletą jest to, że programiści kodujący zarówno w Javie, jak iw C ++ nie muszą pamiętać dwóch różnych nazw składni lub funkcji. Na przykład, aby przekonwertować liczbę całkowitą na ciąg do –

String aa;

aa = 34; // The '=' operator will convert int to string

cout << "The value of aa is : " << aa.val() << endl;

aa = 234.878; // The '=' operator will convert float to string

cout << "The value of aa is : " << aa.val() << endl;

aa = 34 + 234.878;

cout << "The value of aa is : " << aa.val() << endl;

// The output aa will be '268.878'

// You must cast String to convert

aa = (String) 34 + " Can create infinite number of universes!! " + 234.878;

cout << "The value of aa is : " << aa.val() << endl;

// The output aa will be '34 Can create infinite number of universes!! 234.878'

7. Plik String.h

W języku C ++ (lub dowolnym języku zorientowanym obiektowo) wystarczy przeczytać "strukturę danych klasy" (tj. Interfejs), aby rozpocząć korzystanie z tego obiektu. Trzeba tylko zrozumieć interfejs, a nie implementację interfejsu. W przypadku klasy String wystarczy przeczytać i zrozumieć klasę String w pliku String.h. Nie musisz czytać całej implementacji (String.cpp), aby użyć klasy String. Klasy zorientowane obiektowo są oszczędne w czasie rzeczywistym i bardzo starannie ukrywają implementację. (W języku Java zorientowanym obiektowo istnieje odpowiednik nazywany "interfejsem", który ukrywa szczegóły implementacji.)

//

// Autor: Al Dev E-mail: alavoor@yahoo.com

```
// Użyj klasy string lub klasy String
//
// Aby zapobiec wyciekom pamięci - klasa char do zarządzania zmiennymi znakowymi
// Zawsze wolisz używać klasy String lub string
// zamiast char [] lub char *
//
#ifndef __STRING_H_ALDEV_
#define __STRING_H_ALDEV_
// nie używaj programu iostream, ponieważ program staje się nieporęczny ...
#ifdef NOT_MSWINDOWS
#include <iostream>
#jeszcze
#include <iostream.h> // niektóre kompilatory takie jak .h tutaj. Czemu???
#endif // NOT_MSWINDOWS
#include <stdio.h> // dla FILE i sprintf ()
// # include <list.h> // dla listy
// Do kompilatorów MS Windows 95 VC ++ lub Borland C ++ -
// zobacz plik d: \ program files \ CBuilder \ include \ examples \ stdlib \ list.cpp i include \ list.h
// # include <list> // dla listy
// używanie przestrzeni nazw std;
const short INITIAL_SIZE = 50;
const short NUMBER_LENGTH = 300;
const int MAX_ISTREAM_SIZE = 2048;
// class StringBuffer;
// Skompilowałem i przetestowałem tę klasę napisów w systemie Linux (Redhat 7.1) i
// MS Windows Borland C ++ wersja 5.2 (win32). To też powinno działać
// przy użyciu kompilatora MS Visual C ++
klasa String
{
publiczny:
Strunowy();
```

```
String (const char bb []); // potrzebne operatorowi +

String (const char bb [], int start, int slength); // podzbiór znaków

String (int bb); // potrzebne operatorowi +

String (unsigned long bb); // potrzebne operatorowi +

String (długi bb); // potrzebne operatorowi +

String (float bb); // potrzebne operatorowi +

String (double bb); // potrzebne operatorowi +

String (const String & rhs); // Kopiuj Konstruktor potrzebny operatorowi +

// String (StringBuffer sb); // Kompatybilność Java - ale powoduje problem kompilacji w łańcuchu (int bb, bool manekin); // dla klasy StringBuffer

virtual ~ String (); // Wykonane wirtualnie, tak aby po usunięciu klasy bazowej

// następnie wywoływany jest destruktor klasy pochodnej.

char * val () {return sval;} // Nie jest bezpiecznie udostępniać publicznie

// Funkcje poniżej naśladują obiekt String języka Java

unsigned long length ();

char charAt (int where);

void getChars (int sourceStart, int sourceEnd,

char target [], int targetStart);

char * toCharArray ();

char * getBytes ();

bool równa się (String str2); // Zobacz także operator ==

bool jest równy (char * str2); // Zobacz także operator ==

bool equalsIgnoreCase (String str2);

bool regionMatch (int startIndex, String str2,

int str2StartIndex, int numChars);

bool regionMatch (bool ignoreCase, int startIndex,

String str2, int str2StartIndex, int numChars);

String toUpperCase ();

String toLowerCase ();

bool startsWith (String str2);

bool startsWith (char * str2);
```

```
bool endsWith (String str2);
bool endsWith (char * str2);
int compareTo (String str2);
int compareTo (char * str2);
int compareToIgnoreCase (String str2);
int compareToIgnoreCase (char * str2);
int indexOf (char ch, int startIndex = 0);
int indexOf (char * str2, int startIndex = 0);
int indexOf (String str2, int startIndex = 0);
int lastIndexOf (char ch, int startIndex = 0);
int lastIndexOf (char * str2, int startIndex = 0);
int lastIndexOf (String str2, int startIndex = 0);
Podciągi łańcuchowe (int startIndex, int endIndex = 0);
Zastępowanie ciągu (oryginalny znak, zamiana znaków);
Zastąp ciąg (char * original, char * replacement);
String trim (); // Zobacz także przeciążone wykończenie ()
String concat (String str2); // Zobacz także operator +
String concat (char * str2); // Zobacz także operator +
String concat (int bb);
String concat (unsigned long bb);
String concat (float bb);
String concat (double bb);
String reverse (); // Patrz także overloaded reverse ()
String deleteCharAt (int loc);
String deleteStr (int startIndex, int endIndex); // Java "delete ()"
String valueOf (char ch)
{char aa [2]; aa [0] = ch; aa [1] = 0; return String (aa);}
String valueOf (char chars []) {return String (chars);}
String valueOf (char chars [], int startIndex, int numChars);
String valueOf (bool tf)
{if (tf) return String ("true"); else return String ("false");}
```

```
String valueOf (int num) {return String (num);}
String valueOf (long num) {return String (num);}
String valueOf (float num) {return String (num);}
String valueOf (double num) {return String (num);}
// Zobacz także klasę StringBuffer w tym pliku podaną poniżej
// ---- Koniec Java jak funkcje obiektu String -----
//////////////////////////////////////////////////////////////////////////// ////
// Lista dodatkowych funkcji spoza java
//////////////////////////////////////////////////////////////////////////// ////
String ltrim();
void ltrim(bool dummy); // Directly changes object. dummy to get different signature
String rtrim();
void rtrim(bool dummy); // Directly changes object. See also chopall().
// dummy to get different signature
void chopall(char ch='\n'); // removes trailing character 'ch'. See also rtrim()
void chop(); // removes one trailing character
void roundf(float input_val, short precision);
void decompose_float(long *integral, long *fraction);
void roundd(double input_val, short precision);
void decompose_double(long *integral, long *fraction);
void explode(char *separator); // see also token() and overloaded explode()
String *explode(int & strcount, char separator = ' '); // see also token()
void implode(char *glue);
void join(char *glue);
String repeat(char *input, unsigned int multiplier);
String tr(char *from, char *to); // translate characters
String center(int padlength, char padchar = ' ');
String space(int number = 0, char padchar = ' ');
String xrange(char start, char end);
String compress(char *list = " ");
String left(int slength = 0, char padchar = ' ');
```

```
String right(int slength = 0, char padchar = ' ');
String overlay(char *newstr, int start = 0, int slength = 0, char padchar = ' ');
String at(char *regx); // matches first match of regx
String before(char *regx); // returns string before regx
String after(char *regx); // returns string after regx
String mid(int startIndex = 0, int length = 0);
bool isNull();
bool isInteger();
bool isInteger(int pos);
bool isNumeric();
bool isNumeric(int pos);
bool isEmpty(); // same as length() == 0
bool isUpperCase();
bool isUpperCase(int pos);
bool isLowerCase();
bool isLowerCase(int pos);
bool isWhiteSpace();
bool isWhiteSpace(int pos);
bool isBlackSpace();
bool isBlackSpace(int pos);
bool isAlpha();
bool isAlpha(int pos);
bool isAlphaNumeric();
bool isAlphaNumeric(int pos);
bool isPunct();
bool isPunct(int pos);
bool isPrintable();
bool isPrintable(int pos);
bool isHexDigit();
bool isHexDigit(int pos);
bool isCntrl();
```

```
bool isCntrl(int pos);
bool isGraph();
bool isGraph(int pos);
void clear();
int toInteger();
long parseLong();
double toDouble();
String token(char separator = ' '); // see also StringTokenizer, explode()
String crypt(char *original, char *salt);
String getline(FILE *infp = stdin); // see also putline()
//String getline(fstream *infp = stdin); // see also putline()
void putline(FILE *outfp = stdout); // see also getline()
//void putline(fstream *outfp = stdout); // see also getline()
void swap(String aa, String bb); // swap aa to bb
String *sort(String aa[]); // sorts array of strings
String sort(int startIndex = 0, int length = 0); // sorts characters inside a string
int freq(char ch); // returns the number of distinct, nonoverlapping matches
void Format(const char *fmt, ...);
String replace (int startIndex, int endIndex, String str);
void substring(int startIndex, int endIndex, bool dummy); // Directly changes object
void reverse(bool dummy); // Directly changes object. dummy to get different signature
String deleteCharAt(int loc, bool dummy); // Directly changes object
String deleteStr(int startIndex, int endIndex, bool dummy);
void trim(bool dummy); // Directly changes object. dummy to get different signature
String insert(int index, String str2);
String insert(int index, String str2, bool dummy); // Directly changes object
String insert(int index, char ch);
String insert(int index, char ch, bool dummy); // Directly changes object
String insert(char *newstr, int start = 0, int length = 0, char padchar = ' ');
String dump(); // Dump the string like 'od −c' (octal dump) does
// required by java's StringBuffer
```

```
void ensureCapacity(int capacity);

void setLength(int len);

void setCharAt(int where, char ch); // see also charAt(), getCharAt()

// required by java's Integer class, Long, Double classes

int parseInt(String ss) {return ss.toInteger();}

int parseInt(char *ss)

{String tmpstr(ss); return tmpstr.toInteger();}

long parseLong(String ss) {return ss.parseLong();}

long parseLong(char *ss)

{String tmpstr(ss); return tmpstr.parseLong();}

float floatValue() {return (float) toDouble(); }

double doubleValue() {return toDouble(); }

char * number2string(int bb); // see also String(int)

char * number2string(long bb); // see also String(long)

char * number2string(unsigned long bb); // see also String(long)

char * number2string(double bb); // see also String(double)

////////////////////////////////////////////

// List of duplicate function names

////////////////////////////////////////////

// char * c_str() // use val()

// bool find(); // Use regionMatches()

// bool search(); // Use regionMatches()

// bool matches(); // Use regionMatches()

// int rindex(String str2, int startIndex = 0); Use lastIndexOf()

// String blanks(int slength); // Use repeat()

// String append(String str2); // Use concat() or + operator

// String prepend(String str2); // Use + operator. See also append()

// String split(char separator = ' '); // Use token(), explode() or StringTokenizer bool contains(char *str2,
int startIndex = 0); // use indexOf()

// void empty(); Use is_empty()

// void vacuum(); Use clear()
```

```
// void erase(); Use clear()
// void zero(); Use clear()
// bool is_float(); Use is_numeric();
// bool is_decimal(); Use is_numeric();
// bool is_Digit(); Use is_numeric();
// float float_value(); Use toDouble();
// float tofloat(); Use toDouble();
// double double_value(); Use toDouble();
// double numeric_value(); Use toDouble();
// int int_value(); Use toInteger()
// int tonumber(); Use toInteger()
// String get(); Use substring() or val() but prefer java's substring
// String getFrom(); Use substring() or val() but prefer java's substring
// String head(int len); Use substring(0, len)
// String tail(int len); Use substring(length()−len, length())
// String cut(); Use deleteCharAt() or deleteStr()
// String cutFrom(); Use deleteCharAt() or deleteStr()
// String paste(); Use insert()
// String fill(); Use replace()
// char firstChar(); // Use substring(0, 1);
// char lastChar(); // Use substring(length()−1, length());
// String findNext(); Use token(), explode() or StringTokenizer class
// begin(); iterator. Use operator [ii]
// end(); iterator. Use operator [ii]
// copy(); Use assignment = operator, String aa = bb;
// clone(); Use assignment = operator, String aa = bb;
// void putCharAt(int where, char ch); Use setCharAt()
// void replaceCharAt(int where, char ch); Use setCharAt()
// char getCharAt(int where); Use CharAt()
// void parseArgs(int where, char ch); Use StringTokensizer class, token() or explode()
// void truncate(); Use trim(), rtrim(), chop() or chopall()
```

```
// convert number to string notostring(), int2str, long2str Use number2string()
// All Operators ...
String operator+ (const String & rhs);
friend String operator+ (const String & lhs, const String & rhs);
String& operator+= (const String & rhs); // using reference will be faster
String& operator= (const String & rhs); // using reference will be faster
bool operator== (const String & rhs); // using reference will be faster
bool operator== (const char *rhs);
bool operator!= (const String & rhs);
bool operator!= (const char *rhs);
char operator [] (unsigned long Index) const;
char& operator [] (unsigned long Index);
friend ostream & operator<< (ostream & Out, const String & str2);
friend istream & operator>> (istream & In, String & str2);
//do later: static list<String> explodeH; // list head
protected:
char *sval; // Not safe to make sval public
void verifyIndex(unsigned long index) const; // not "inline" because MS Win32 complains
void verifyIndex(unsigned long index, char *aa) const;// not "inline" − MS Win32
void _str_cat(char bb[]);
void _str_cat(int bb);
void _str_cat(unsigned long bb);
void _str_cat(float bb);
void _str_cpy(char bb[]);
void _str_cpy(int bb); // itoa
void _str_cpy(unsigned long bb);
void _str_cpy(float bb); // itof
private:
// Note: All the private variables and functions begin
// with _ (underscore)
//static String *_global_String; // for use in add operator
```

```
//inline void _free_glob(String **aa);
bool _equalto(const String & rhs, bool type = false);
bool _equalto(const char *rhs, bool type = false);
String *_pString; // temporary pointer for internal use..
char *_pNumber2String; // temporary pointer for internal use..
inline void _allocpString();
inline void _allocpNumber2String();
inline void Common2AllCstrs();
inline void _reverse();
inline void _deleteCharAt(int loc);
inline void _deleteStr(int startIndex, int endIndex);
inline void _trim();
inline void _ltrim();
inline void _rtrim();
inline void _substring(int startIndex, int endIndex);
void _roundno(double input_dbl, float input_flt, short precision, bool type);
};
// Global variables are defined in String.cpp
#endif // __STRING_H_ALDEV_
```

7.1 StringBuffer.h

```
//
// Author : Al Dev Email: alavoor@yahoo.com
//
#ifndef __STRINGBUFFER_H_ALDEV_
#define __STRINGBUFFER_H_ALDEV_
// Imitate Java's StringBuffer object
// This class is provided so that the Java code is
// portable to C++, requiring minimum code changes
// Note: While coding in C++ DO NOT use this class StringBuffer,
// this is provided only for compiling code written in Java
// which is cut/pasted inside C++ code.
```

```
class StringBuffer: public String
{
public:
StringBuffer();
~StringBuffer();
StringBuffer(char *aa);
StringBuffer(int size);
StringBuffer(String str);
int capacity();
StringBuffer append(String str2);
// See also operator +
//{ *this += str2; return *this;} // This is causing core dumps...
StringBuffer append(char *str2);
StringBuffer append(int bb);
StringBuffer append(unsigned long bb) ;
StringBuffer append(float bb) ;
StringBuffer append(double bb) ;
StringBuffer insert(int index, String str2);
StringBuffer insert(int index, char ch);
StringBuffer reverse();
// Java's "delete()". Cannot use name delete in C++
StringBuffer deleteStr(int startIndex, int endIndex);
StringBuffer deleteCharAt(int loc);
StringBuffer substring(int startIndex, int endIndex = 0);
void assign(char *str);
private:
StringBuffer *_pStringBuffer;
inline void allocpStringBuffer();
inline void Common2AllCstrs();
};
#endif // __STRINGBUFFER_H_ALDEV_
```

7.2 StringTokenizer.h

```
//
// Author : Al Dev Email: alavoor@yahoo.com
//
#ifndef __STRINGTOKENIZER_H_ALDEV_
#define __STRINGTOKENIZER_H_ALDEV_
// Imitate java's StringTokenizer class
// provided to compile java code in C++ and vice−versa
class StringTokenizer: public String
{
public:
StringTokenizer(String str);
StringTokenizer(String str, String delimiters);
StringTokenizer(String str, String delimiters, bool delimAsToken);
~StringTokenizer();
int countTokens();
bool hasMoreElements();
bool hasMoreTokens();
String nextElement(); // in java returns type 'Object'
String nextToken();
String nextToken(String delimiters);
private:
int CurrentPosition; // current index on string
int TotalTokens;
int RemainingTokens;
char * ListOfDl; // list of delimiters
char * WorkStr; // temp work string
char * OrigStr; // original string passed
bool DlFlag; // delimiter flag
inline void vPrepWorkStr(char *delimiters = NULL);
};
```

#endif // __STRINGTOKENIZER_H_ALDEV_

8. Zmiana nazwy klasy String

8.1 Przypadek 1: Prosta zmiana nazwy

Jeśli nie podoba ci się nazwa klasy String, możesz użyć "typedef", aby zmienić nazwę klasy String. We wszystkich plikach, do których włączasz String.h, wstaw poniższe linie: // If you do not like the class name String, then you can rename using typedef

typedef String StringSomethingElseIwant;

// Your remaing code may be like this ....

int main()

{

StringSomethingElseIwant aa_renstr;

aa_renstr = "I renamed the String Class using typedef";

.....etc...

}

8.2 Przypadek 2: Rozwiąż konflikt

Jeśli występuje konflikt z inną nazwą klasy o tej samej nazwie i chcesz używać zarówno tej klasy, jak i klasy powodującej konflikt, użyj tej techniki - we wszystkich plikach, do których włączasz String.h, wstaw poniższe linie:

#define String String_somethingelse_which_I_want

#include "String.h"

#undef String

#include "ConflictingString.h" // This also has String class...

// All your code goes here...

main()

{

String_somethingelse_which_I_want aa;

String bb; // This string class from conflicting string class

aa = " some sample string";

bb = " another string abraka−dabraka";

.....

}

Preprocesor zastąpi wszystkie literały ciągu String na "String_somethingelse_which_I_want" i natychmiastowo niezdefiniuje String. Po undef dołączony jest plik nagłówkowy klasy konfliktu, który definiuje klasę "String".

9. Klasa plików

Można użyć klasy File do manipulowania plikami systemu operacyjnego. Ta klasa jest imitacją klasy plików Java i będzie bardzo przydatna w programowaniu w C ++. Korzystając z tej klasy plików w C ++, możesz zrobić, jeśli plik istnieje () ?, jeśli istnieje katalog ()?, Długość pliku () i inne funkcje.

* Klasa plików C ++ znajduje się w File.h http://www.angelfire.com/country/aldev0/cpphowto/File.h i File.cpp

http://www.angelfire.com/country/aldev0/cpphowto/File.cpp

* Definicja klasy Java java.io.File http://java.sun.com/j2se/1.3/docs/api/java/io/File.html

* Krótki przegląd klasy plików http://unicornsrest.org/reference/java/qref11/java.io.File.html

* Podsumowanie klasy plików http://www.idi.ntnu.no/~database/SIF8020/java-api/java.io.File.html

10. Funkcja C ++ Zap (Delete)

Operatory delete i new w C ++ są znacznie lepsze niż funkcje malloc i darmowe C. Rozważ użycie new i zap (delete function) zamiast malloc i free jak najwięcej. Aby operatory usuwania były jeszcze bardziej czyste, wykonaj wbudowaną funkcję Zap (). Zdefiniuj funkcję zap () w następujący sposób:

```
// Put an assert to check if x is NULL, this is to catch

// program "logic" errors early. Even though delete works

// fine with NULL by using assert you are actually catching

// "bad code" very early

// Defining Zap using templates

// Use zap instead of delete as this will be very clean

template <class T>

inline void zap(T & x)

{

{assert(x != NULL);}

delete x;

x = NULL;

}

// In C++ the reason there are 2 forms of the delete operator is − because

// there is no way for C++ to tell the difference between a pointer to

// an object and a pointer to an array of objects. The delete operator

// relies on the programmer using "[]" to tell the two apart.
```

```
// Hence, we need to define zaparr function below.

// To delete array of pointers

template <class T>

inline void zaparr(T & x)

{

{assert(x != NULL);}

delete [] x;

x = NULL;

}
```

Funkcja zap () usunie wskaźnik i ustawi NULL. Zapewni to, że nawet jeśli wiele zap () zostanie wywołanych na tym samym usuniętym wskaźniku, program się nie zawiesza. Zapoznaj się z funkcją zap_example () w example_String.cpp.

```
// See zap_example() in example_String.cpp

zap(pFirstname);

//zap(pFirstname); // no core dumps. Because pFirstname is NULL now

//zap(pFirstname); // no core dumps. Because pFirstname is NULL now

zap(pLastname);

zap(pJobDescription);

int *iiarray = new int[10];

zaparr(iiarray);
```

Nie ma w tym nic magicznego, to po prostu zapisuje powtarzalny kod, oszczędza czas pisania i sprawia, że programy są bardziej czytelne. Programiści C ++ często zapominają zresetować usunięty wskaźnik do NULL, co powoduje irytujące problemy powodujące zrzuty i awarie rdzenia. Funkcja zap () zajmuje się tym automatycznie. Nie przyklejaj typecast do funkcji zap () - jeśli coś wypływa na powyższą funkcję zap (), prawdopodobnie ma gdzieś inny błąd. Również my_malloc (), my_realloc () i my_free () powinny być używane zamiast malloc (), realloc () i free (), ponieważ są one znacznie czyściejsze i mają dodatkowe kontrole. Dla przykładu zobacz plik "String.h", który używa funkcji my_malloc () i my_free ().

OSTRZEŻENIE: Nie używaj funkcji free () do zwolnienia pamięci przydzielonej przez "new" lub "delete" do wolnej pamięci przydzielonej przez malloc. Jeśli to zrobisz, wyniki będą nieprzewidywalne .

11. Wskaźniki są problemami

Wskaźniki nie są wymagane do programowania ogólnego. W nowoczesnych językach takich jak Java nie ma wsparcia dla wskaźników (Java wewnętrznie używa wskaźników). Wskaźniki powodują, że programy są niechlujne, a programy używające wskaźników są bardzo trudne do odczytania. Unikaj używania wskaźników w jak największym stopniu i używaj referencji. Wskaźniki są naprawdę wielkim

bólem. Możliwe jest pisanie aplikacji bez użycia wskaźników. Powinieneś wskazywać tylko w tych przypadkach, w których referencje nie będą działać.

Odwołanie jest aliasem; kiedy tworzysz referencję, zainicjujesz ją nazwą innego obiektu, celu. Od tej chwili odnośnik działa jako alternatywna nazwa celu, a wszystko, co robisz do odniesienia, jest naprawdę zrobione do celu. Składnia odniesień: Zadeklaruj referencję, pisząc typ, a następnie operator referencyjny (&), a następnie nazwę referencyjną. Piśmiennictwo MUSI być zainicjalizowane w momencie tworzenia. Na przykład -

int weight;

int & rweight = weight;

DOG aa;

DOG & rDogRef = aa;

Referencje -

* Użyj referencji, aby utworzyć alias do obiektu

* Zrób inicjalizację wszystkich odniesień

* Używaj referencji dla wysokiej wydajności i wydajności programu.

* Używaj const, aby chronić odniesienia i wskaźniki, gdy tylko jest to możliwe.

Nie polecam -

· WAŻNE: nie używaj odwołań do obiektów NULL

Nie mylić adresu operatora z operatorem referencyjnym. Odniesienia są używane w sekcji deklaracji

* Nie próbuj ponownie przypisywać odniesienia

* Nie używaj wskaźników, jeśli odniesienia będą działać

* Nie zwracaj odwołania do lokalnego obiektu

* Nie należy przekazywać przez odniesienie, jeżeli przedmiot, o którym mowa, może wykraczać poza zakres

12. Używanie my_malloc i my_free

Staraj się unikać używania malloc i realloc jak najwięcej i użyj new i zap (delete). Ale czasami może być konieczne użycie alokacji pamięci w stylu "C" w "C ++". Użyj funkcji my_malloc (), my_realloc () i my_free (). Funkcje te wykonują odpowiednie alokacje i inicjalizacje i starają się zapobiegać problemom z pamięcią. Również te funkcje (w trybie DEBUG) mogą śledzić przydzieloną pamięć i drukować całkowite wykorzystanie pamięci przed i po uruchomieniu programu. To powie Ci, czy są jakieś wycieki pamięci. My_malloc i my_realloc są zdefiniowane poniżej. Przydziela nieco więcej pamięci (SAFE_MEM = 5) i inicjalizuje przestrzeń, a jeśli nie może jej przydzielić, kończy działanie programu. Funkcje "call_check (), remove_ptr ()" są aktywne tylko wtedy, gdy DEBUG_MEM jest zdefiniowany w pliku makefile i jest przypisany do ((void) 0), tj. NULL dla braku wersji debugowania. Umożliwiają śledzenie całkowitej pamięci używane.

void *local_my_malloc(size_t size, char fname[], int lineno)

```
{
size_t tmpii = size + SAFE_MEM;
void *aa = NULL;
aa = (void *) malloc(tmpii);
if (aa == NULL)
raise_error_exit(MALLOC, VOID_TYPE, fname, lineno);
memset(aa, 0, tmpii);
call_check(aa, tmpii, fname, lineno);
return aa;
}
char *local_my_realloc(char *aa, size_t size, char fname[], int lineno)
{
remove_ptr(aa, fname, lineno);
unsigned long tmpjj = 0;
if (aa) // aa != NULL
tmpjj = strlen(aa);
unsigned long tmpqq = size + SAFE_MEM;
size_t tmpii = sizeof (char) * (tmpqq);
aa = (char *) realloc(aa, tmpii);
if (aa == NULL)
raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);
// do not memset memset(aa, 0, tmpii);
aa[tmpqq−1] = 0;
unsigned long kk = tmpjj;
if (tmpjj > tmpqq)
kk = tmpqq;
for ( ; kk < tmpqq; kk++)
aa[kk] = 0;
call_check(aa, tmpii, fname, lineno);
return aa;
}
```

Zobacz my_malloc.cpp. oraz plik nagłówkowy my_malloc.h. dla pełnej implementacji programu my_malloc. Przykład użycia my_malloc i my_free jak poniżej:

```
char *aa;

int *bb;

float *cc;

aa = (char *) my_malloc(sizeof(char)* 214);

bb = (int *) my_malloc(sizeof(int) * 10);

cc = (float *) my_malloc(sizeof(int) * 20);

aa = my_realloc(aa, sizeof(char) * 34);

bb = my_realloc(bb, sizeof(int) * 14);

cc = my_realloc(cc, sizeof(float) * 10);
```

Zauważ, że w my_realloc nie musisz rzutować typu danych, gdy przekazywana jest sama zmienna i poprawiona jest my_realloc, która zwraca prawidłowy wskaźnik typu danych. My_realloc ma przeciążone funkcje char *, int * i float *.

12.1 Garbage Collector dla C ++

W C / C ++ Garbage Collection nie jest standardową funkcją, dlatego przydzielanie i zwalnianie pamięci jawnie jest trudne, skomplikowane i podatne na błędy. Garbage Collection (GC) nie jest częścią standardu C ++, ponieważ istnieje tylko tyle sposobów, w jaki można go wdrożyć; istnieje wiele technik GC, a podjęcie decyzji o użyciu konkretnego nie byłoby dobre w przypadku niektórych programów. Informatycy zaprojektowali wiele algorytmów GC, z których każdy jest przeznaczony dla konkretnej dziedziny problemowej. Nie ma jednego singla ogólny GC, który zajmie się wszystkimi problematycznymi domenami. W konsekwencji GC nie jest częścią standardu C ++, po prostu go opuścili. Mimo to zawsze masz do wyboru wiele swobodnie dostępnych bibliotek C ++, które wykonują zadanie za Ciebie.

13. Pliki debugowania

Do debugowania dowolnych programów w C ++ lub C należy plik debug.h, aw pliku Makefile zdefiniuj DEBUG_STR, DEBUG_PRT, DEBUG_MEM, aby włączyć śledzenie z funkcji debug.h. Po usunięciu "-DDEBUG_STR" itd., Wywołania funkcji debugowania są ustawione na ((void) 0), tj. NULL, więc nie ma to wpływu na końcową wersję projektu. Możesz hojnie korzystać z funkcji debugowania w swoich programach i nie zwiększy to rozmiaru pliku wykonywalnego produkcji.

Zobacz plik debug.cpp w celu implementacji procedur debugowania. Zobacz plik my_malloc.cpp dla przykładu, który używa funkcji debug.h i debugowania.

14. Java podobny do API

Odwiedź następujące strony dla Java, takie jak API dla C ++

* Java utils w C ++ http://www.pulsar.org/users/ej/archive/oop

* PhD Thesis book Java API w C ++ http://www.pulsar.org/archive/phd/ejphd

15. Narzędzia IDE dla C ++

Następujące narzędzia IDE (Integrated Development Environment) są dostępne dla C ++:

"Najlepiej oceniany" Dev-C ++ to w pełni funkcjonalne zintegrowane środowisko programistyczne (IDE) zarówno dla Win32 jak i Linuxa. Używa GCC, Mingw lub Cygwin jako zestawu kompilatora i bibliotek. Jest na http://www.bloodshed.net/devcpp.html i na mirror-site

* KDE Kdevelop

* Blatura site C++ Tools

* Amulet Amulet

* App Dev suite Angoss

* Make replacement Brass

* S/W product metrics CCC

* Project mgmt, edit, compile, debug C−Forge

*· Dev environment Code Crusader

* Graphic gdb Code Medic

* Code analysis CodeWizard

* Gen HTML, LaTex for C++ cod Doc C++

* GUI toolkit openGL Ftk

* C++ and Java IDE GLG IDE

* HP IDE HP Eloquence

* IDE C++, Java, Pascal RHIDE

* IDE for C++, Java SNiff

* IDE for C++, Java Wipeout

* X−based dev env XWPE

16. Standardy kodowania w C ++

Standard kodowania jest bardzo ważny dla czytelności i przydatności programów. A także znacznie zwiększa produktywność programisty. Kompilator GNU C ++ musi wymuszać dyscyplinę kodowania. Sugerowane jest następujące polecenie - definicja klasy wewnętrznej:

* Wszystkie publiczne zmienne muszą zaczynać się od m jak mFooVar. M oznacza członka.

* Wszystkie chronione zmienne muszą rozpoczynać się od mt, jak mtFooVar i metody z t, jak tFooNum (). t oznacza chronione.

* Wszystkie prywatne zmienne muszą rozpoczynać się od mv, jak mvFooVar i metod v, jak vFooLone (). V oznacza prywatne.

* Wszystkie publiczne, chronione i prywatne zmienne muszą zaczynać się wielkimi literami po m like F w mFooVar.

* Wszystkie zmienne wskaźnika muszą być poprzedzone przedrostkiem p, jak:

- Zmienne publiczne mpFooVar i metody takie jak FooNum ()

- Chronione zmienne mtpFooVar i metody z t lub tFooNum ()

- Zmienne prywatne mvpFooVar i metody z v jak vFooNum ()

Kompilator powinien generować błąd, jeśli kod nie jest zgodny ze standardem. Kompilator C ++ może dostarczyć opcję flagi, aby ominąć ścisły standard kodowania, aby skompilować stary kod źródłowy, a dla całego nowego kodu opracowane zgodnie z jednolitym światowym standardem kodowania. W przykładowym kodzie podanym poniżej t oznacza chroniony, v oznacza prywatny, m oznacza zmienną składową i p oznacza wskaźnik.

```
class SomeFunMuncho

{

public:

int mTempZimboniMacho; // Only temporary variables should be public as per OOP

float *mpTempArrayNumbers;

int HandleError();

float getBonyBox(); // Public accessor as per OOP design

float setBonyBox(); // Public accessor as per OOP design

protected:

float mtBonyBox;

int *mtpBonyHands;

char *tHandsFull();

int tGetNumbers();

private:

float mvJustDoIt;

char mvFirstName[30];

int *mvpTotalValue;

char *vSubmitBars();

int vGetNumbers();

};
```

Kiedy twój program rośnie o miliony linii kodu, bardzo docenisz konwencję nazewnictwa jak powyżej. Czytelność kodu poprawia się, ponieważ po prostu patrząc na nazwę zmiennej, taką jak mvFirstName, można stwierdzić, że jest ona członkiem klasy i jest zmienną prywatną.

17 Narzędzia Pamięci

Użyj następujących narzędzi do debugowania pamięci

* "MPatrol" jest bardzo potężnym narzędziem do debugowania pamięci. Jest na http://www.cbmamiga.demon.co.uk/mpatrol i na http://www.rpmfind.net przejdź tutaj i wyszukaj mpatrol. Jeśli używasz Linuksa, musisz pobrać plik mpatrol * .src.rpm z pliku rpmfind.net. Aby zaktualizować mpatrol * .src.rpm do najnowszej wersji, możesz użyć starego pliku mpatrol.spec i najnowszego pliku mpatrol * .tar.gz, aby odbudować nowy * .src.rpm.

* W linux contrib cdrom patrz pakiet mem_test * .rpm i na http://www.rpmfind.net przejdź tutaj i wyszukaj mem_test.

* Na CD ROM-ie zobacz pakiet ElectricFence * .rpm i http://www.rpmfind.net.

* Narzędzie Oczyszczanie z Rational Software Corp http://www.rational.com

* Narzędzie Insure ++ firmy Parasoft Corp http://www.parasoft.com

* Narzędzia Linux na http://www.xnet.com/~blatura/linapp6.html#tools

Wyszukaj w wyszukiwarkach internetowych takich jak Yahoo, Lycos, Excite, Mamma.com słowo kluczowe "Narzędzia do debugowania pamięci Linux".

18. Języki skryptów C ++

Główną wadą C ++ jest to, że musisz przekompilować i połączyć pliki obiektów, aby utworzyć plik wykonywalny za każdym razem, gdy wprowadzisz niewielką zmianę. Cykle kompilacji / łączenia / debugowania zabierają dużo czasu i są dość bezproduktywne. Ponieważ współczesne procesory i pamięć RAM stają się niezwykle szybkie i tanie, lepiej wydawać więcej pieniędzy na sprzęt i używać języków skryptowych do programowania.

19.1 PIKE (język skryptowy C / C ++)

Język skryptowy, taki jak PIKE, eliminuje łączenie i ponowne kompilowanie i naprawdę przyspieszy proces tworzenia. Ponieważ ceny pamięci (RAM) spadają, a prędkości procesora rosną, język skryptowy, taki jak PIKE, zyska na popularności. PIKE stanie się najczęściej używanym językiem skryptowym, ponieważ jest zorientowany obiektowo a jego składnia jest bardzo podobna do języka C ++. Wydajność programowania zwiększy się pięć razy dzięki zastosowaniu języka skryptowego Pike C ++. A Pike jest bardzo przydatny dla "proof of concept" i szybkiego opracowywania prototypów. Pike znajduje się na http://pike.roxen.com i http://www.roxen.com. Serwer sieciowy Roxen został całkowicie napisany w Pike, co pokazuje, jak potężny jest Pike. Szczupaki biegną znacznie szybszy niż Java dla niektórych operacji i jest dość wydajny w korzystaniu z zasobów pamięci.

19.2 SoftIntegration Ch (język skryptowy C / C ++)

Jeśli chcesz mieć komercyjny język skryptowy, uzyskaj produkt "Ch scripting" od SoftIntegration corporation pod adresem http://www.softintegration.com. Środowisko języków skryptowych o nazwie Ch jest nadzbiorem C z wysokopoziomowymi rozszerzeniami i istotnymi funkcjami z C ++ i innych języków, dzięki czemu użytkownicy mogą nauczyć się tego języka tylko raz i używać go w dowolnym miejscu dla prawie każdego programowania. Środowisko języka skryptowego zgodne z literą C jest również oprogramowaniem pośredniczącym, stanowiącym kluczową infrastrukturę oprogramowania do uruchamiania aplikacji przenośnych na heterogenicznych platformach. Przenośny kod Ch można bezpiecznie wdrożyć w Internecie lub w intranecie, aby można go było uruchomić w dowolnym miejscu superkomputery, stacje robocze, komputery osobiste, palm-piloty, urządzenia PDA, do nietradycyjnych urządzeń komputerowych, takich jak maszyny CNC, roboty, telewizory, lodówki i inne.

19.3 PHP (język skryptowy C ++)

PHP jest językiem skryptowym preprocesora hipertekstowego i bardzo szybko ewoluuje i uzyskuje cechy obiektowe. Ma słowo kluczowe "class", dzięki któremu próbuje zaimplementować skryptowanie obiektowe. Może być w niedalekiej przyszłości PHP szybko dojrzeje, aby stać się solidnym językiem skryptowym dla projektów zorientowanych obiektowo. W przyszłości zajmie się zarówno aplikacjami sieciowymi, jak i aplikacjami ogólnego przeznaczenia. Dlaczego mają różne języki skryptowe dla aplikacji internetowych i ogólnych, zamiast tego po prostu użyj PHP dla obu. PHP znajduje się pod adresem http://www.linuxdoc.org/HOWTO/PHP-HOWTO.html.

20 Przegląd STL

STL oferuje programistom szereg przydatnych struktur danych i algorytmów. Składa się z następujących składników.

* Pojemniki. Istnieją dwa rodzaje:

-Sekwencyjny. Ta grupa zawiera typy wektorów, listy i usunięć.

-Posortowane skojarzenie. Ta grupa obejmuje typy set, map, multiset i multimap.

* Iteratory. Są to wskaźniki podobne do obiektów, które pozwalają użytkownikowi przejść przez zawartość kontenera. Ogólne algorytmy. STL zapewnia szeroki zakres wydajnie wdrażanych standardowych algorytmów (na przykład znajdź, sortuj i scalaj), które działają z typami kontenerów. (Niektóre z kontenerów mają specjalne zastosowania tych algorytmów jako funkcji składowych).

* Obiekty funkcyjne. Obiekt funkcji jest instancją klasy, która zapewnia definicję operatora (). Oznacza to, że możesz użyć takiego obiektu, jak funkcja.

* Adaptery. STL zapewnia

- Adaptery kontenera, które pozwalają użytkownikowi na używanie, na przykład, wektora jako podstawy stosu.

- Funkcje interfejsu, które umożliwiają użytkownikowi tworzenie nowych obiektów funkcji z istniejących obiektów funkcji.

* Alokatory. Każda klasa kontenera STL używa klasy Alokatora do przechowywania informacji o modelu pamięci używanym przez program. Całkowicie zignoruję ten aspekt STL.

Zastanowię się nad wykorzystaniem kontenerów wektorowych, list, zestawów i map. Aby korzystać z tych kontenerów, musisz mieć możliwość korzystania z iteratorów, więc będę miał coś do powiedzenia na temat iteratorów STL. Używanie zestawu i mapowanie pojemników może oznaczać konieczność dostarczenia prostego obiektu funkcji do instancji, więc będę miał również coś do powiedzenia na temat obiektów funkcyjnych. Będę krótko wspomnieć o algorytmach dostarczonych przez STL. W ogóle nie wspomnę o adapterach. Zyskałem swobodę z niektórymi typami argumentów funkcji - na przykład większość argumentów liczb całkowitych, o których mowa poniżej, ma typ typ size_type, który jest typowany do odpowiedniego typu podstawowego, w zależności od zastosowanego modelu alokacji. Jeśli chcesz zobaczyć prawdziwe podpisy różnych omówionych funkcji, spójrz na dokument roboczy lub pliki nagłówkowe. Istnieje szereg klas użytkowych dostarczanych z STL. Jedynym ważnym dla nas jest klasa pary. Ma to następującą definicję:

```
template<class T1, class T2>

class pair {
```

public:

T1 first;

T2 second;

pair(const T1& a, const T2& b) : first(a), second(b) {}

};

i jest wygodna funkcja make_pair z podpisem:

para <T1, T2> make_pair (const T1 i f, const T2 &, s)

20.1 Pliki nagłówkowe

Aby użyć różnych bitów STL, musisz #otworzyć odpowiednie pliki nagłówkowe. Mogą one nieznacznie różnić się od kompilatora do kompilatora, ale dla g ++ interesujące są:

* Vector.h dla typu wektorowego.

* List.h dla typu listy.

* Set.h dla ustawionego typu.

* Map.h dla typu mapy.

* Algo.h, aby uzyskać dostęp do ogólnych algorytmów.

Jeśli nie chcesz pamiętać, który z nich mógłbyś po prostu użyć stl.h, który zawiera wszystkie powyższe (i wszystkie inne pliki nagłówkowe STL).

20.2 Interfejs klas kontenera

Klasy kontenerów mają wiele funkcji składowych, które mają te same nazwy. Te funkcje zapewniają te same (lub bardzo podobne) usługi dla każdej z klas (chociaż oczywiście implementacje będą różne). Poniższa tabela zawiera listę funkcji, które należy rozważyć bardziej szczegółowo. Gwiazda naprzeciw nazwy funkcji wskazuje, że typ kontenera w kolumnie udostępnia funkcję składową o tej nazwie. i jest wygodna funkcja make_pair z podpisem:

pair <T1, T2> make_pair (const T1 i f, const T2 &, s)

a także implementacje operatora == i operatora <. Nie ma nic skomplikowanego w tej klasie szablonów i powinieneś być w stanie go używać bez dalszych wskazówek. Aby go użyć # dołącz plik nagłówkowy pair.h. Występuje w wielu miejscach, ale w szczególności przy korzystaniu z klas zestawu i mapy.

20.3 Interfejs klas kontenera

Klasy kontenerów mają wiele funkcji składowych, które mają te same nazwy. Te funkcje zapewniają te same (lub bardzo podobne) usługi dla każdej z klas (chociaż oczywiście implementacje będą różne). Poniższa tabela zawiera listę funkcji, które należy rozważyć bardziej szczegółowo. Gwiazda naprzeciw nazwy funkcji wskazuje, że typ kontenera w kolumnie udostępnia funkcję składową o tej nazwie.i jest wygodna funkcja make_pair z podpisem:

pair <T1, T2> make_pair (const T1 i f, const T2 &, s)

a także implementacje operatora == i operatora <. Nie ma nic skomplikowanego w tej klasie szablonów i powinieneś być w stanie go używać bez dalszych wskazówek. Aby go użyć # dołącz plik nagłówkowy

pair.h. Występuje w wielu miejscach, ale w szczególności przy korzystaniu z klas zestawu i mapy.

Operation	Purpose	Vector	List	Set	Map
==	comparison	*	*	*	*
<	comparison	*	*	*	*
begin	iterator	*	*	*	*
end	iterator	*	*	*	*
size	no. of elements	*	*	*	*
empty	is container empty	*	*	*	*
front	first element	*	*		
back	last element	*	*		
[ ]	element access/modification	*			*
insert	insert element(s)	*	*	*	*
push_back	insert new last element	*	*		
push_front	insert new first element		*		
erase	remove element(s)	*	*	*	*
pop_back	remove last element	*	*		
pop_front	remove last element			*	
Container Class Interface					

Jeśli poniższa dyskusja pozostawia niejasną (i tak będzie), zawsze możesz napisać mały program testowy, aby sprawdzić, jak zachowuje się jakaś funkcja lub funkcja.

20.4 Wektory

Wektor jest tablicą podobną do kontenera, która poprawia typy tablic C ++. W szczególności nie jest konieczne wiedzieć, jak duży ma być wektor, kiedy go zadeklarujesz, możesz dodać nowe elementy na końcu wektora przy użyciu funkcji push_back. (W rzeczywistości funkcja wstawiania umożliwia wstawianie nowych elementów w dowolnej pozycji wektora, ale jest to bardzo nieefektywna operacja - jeśli musisz to zrobić, często rozważ użycie listy).

Konstruowanie wektorów

wektor jest szablonem klasy, więc deklarując obiekt wektorowy musisz podać typ obiektów, które ma zawierać wektor. Na przykład następujący fragment kodu

```
vector<int> v1;

vector<string> v2;

vector<FiniteAutomaton> v3
```

deklaruje, że v1 jest wektorem zawierającym liczby całkowite, v2 to wektor przechowujący ciągi, a v3 zawiera obiekty typu FiniteAutomaton (przypuszczalnie typ klasy zdefiniowany przez użytkownika). Te deklaracje nie mówią nic o tym, jak duże mają być wektory (implementacje będą używały domyślnego rozmiaru początkowego), a Ty możesz je rozwijać tak, jak potrzebujesz. Możesz podać początkowy rozmiar wektorowi za pomocą deklaracji typu

```
vector<char> v4(26);
```

co oznacza, że v4 ma być wektorem znaków, które początkowo mają miejsce na 26 znaków. Istnieje również sposób inicjalizacji elementów wektorowych. Deklaracja

```
vector<float> v5(100,1.0);
```

mówi, że v5 jest wektorem 100 liczb zmiennoprzecinkowych, z których każdy został zainicjowany na 1.0.

Sprawdzanie swojego wektora

Po utworzeniu wektora możesz sprawdzić aktualną liczbę zawartych w nim elementów za pomocą funkcji rozmiaru. Ta funkcja nie przyjmuje argumentów i zwraca liczbę całkowitą (ściśle określoną jako typ size_type, ale jest to przekształcane na liczbę całkowitą), która mówi, ile elementów znajduje się w wektorze. Co będzie wyświetlone przez następujący mały program?

```
<vector−size.cc>=

#include <iostream.h>

#include <vector.h>

void main()

{

vector<int> v1;

vector<int> v2(10);

vector<int> v3(10,7);

cout << "v1.size() returns " << v1.size() << endl;

cout << "v2.size() returns " << v2.size() << endl;

cout << "v3.size() returns " << v3.size() << endl;

}
```

Aby sprawdzić, czy twój wektor jest pusty, możesz użyć pustej funkcji. Nie przyjmuje żadnych argumentów i zwraca wartość logiczną, prawda, jeśli wektor jest pusty, a fałsz, jeśli nie jest pusty. Co

zostanie wydrukowane przez następujący mały program (prawdziwe odbitki jako 1 i fałszywe odbitki jako 0)?

<vector−empty.cc>=

```
#include <iostream.h>

#include <vector.h>

void main()

{

vector<int> v1;

vector<int> v2(10);

vector<int> v3(10,7);

cout << "v1.empty() has value " << v1.empty() << endl;

cout << "v2.empty() has value " << v2.empty() << endl;

cout << "v3.empty() has value " << v3.empty() << endl;

}
```

Dostęp do elementów wektora

Możesz uzyskać dostęp do elementów wektora za pomocą operatora []. Tak więc, jeśli chcesz wydrukować wszystkie elementy w wektorze, możesz użyć takiego kodu

```
vector<int> v;

// ...

for (int i=0; i<v.size(); i++)

cout << v[i];
```

(który jest bardzo podobny do tego, co możesz napisać dla wbudowanej tablicy).Możesz również użyć operatora [], aby ustawić wartości elementów wektora.

```
vector<int> v;

// ...

for (int i=0; i<v.size(); i++)

v[i] = 2*i;
```

Front funkcji daje dostęp do pierwszego elementu wektora.

```
vector<char> v(10,'a');

// ...

char ch = v.front();
```

Możesz także zmienić pierwszy element za pomocą frontu.

```
vector<char> v(10,'a');
// ...
v.front() = 'b';
```

Funkcja wstecz działa tak samo jak front, ale dla ostatniego elementu wektora.

```
vector<char> v(10,'z');
// ...
char last = v.back();
v.back() = 'a';
```

Oto prosty przykład użycia [].

```
<vector−access.cc>=
#include <vector.h>
#include <iostream.h>
void main()
{
vector<int> v1(5);
int x;
cout << "Enter 5 integers (seperated by spaces):" << endl;
for (int i=0; i<5; i++)
cin >> v1[i];
cout << "You entered:" << endl;
for (int i=0; i<5; i++)
cout << v1[i] << ' ';
cout << endl;
}
```

Wstawianie i usuwanie elementów wektorowych

Wraz z operatorem [], jak opisano powyżej, istnieje wiele innych sposobów zmiany lub dostępu do elementów w wektorze.

* push_back doda nowy element na końcu wektora.

* pop_back usunie ostatni element wektora.

* insert wstawi do wektora jeden lub więcej nowych elementów w wyznaczonym miejscu.

* erase usunie jeden lub więcej elementów z wektora pomiędzy wyznaczonymi pozycjami.

Zauważ, że wstawianie i kasowanie są drogimi operacjami na wektorach. Jeśli używasz ich dużo, powinieneś rozważyć użycie struktury danych listy, dla której są bardziej wydajne

<vector−mod.cc>=

```
#include <iostream.h>
#include <vector.h>
void main()
{
vector<int> v;
for (int i=0; i<10; i++) v.push_back(i);
cout << "Vector initialised to:" << endl;
for (int i=0; i<10; i++) cout << v[i] << ' ' ;
cout << endl;
for (int i=0; i<3; i++) v.pop_back();
cout << "Vector length now: " << v.size() << endl;
cout << "It contains:" << endl;
for (int i=0; i<v.size(); i++) cout << v[i] << ' ';
cout << endl;
int a1[5];
for (int i=0; i<5; i++) a1[i] = 100;
v.insert(& v[3], & a1[0],& a1[3]);
cout << "Vector now contains:" << endl;
for (int i=0; i<v.size(); i++) cout << v[i] << ' ';
cout << endl;
v.erase(& v[4],& v[7]);
cout << "Vector now contains:" << endl;
for (int i=0; i<v.size(); i++) cout << v[i] << ' ';
cout << endl;
}
```

W powyższym zadeklarowany wektor v został zainicjowany za pomocą funkcji push_back. Następnie niektóre elementy zostały przycięte z jego końca za pomocą pop_back. Następnie utworzono zwykłą tablicę całkowitą, a następnie niektóre jej elementy wstawiono do v za pomocą wstawki. Na koniec usunięto elementy z v. Funkcje użyte powyżej przyjmują argumenty w następujący sposób.

* Funkcja push_back przyjmuje pojedynczy argument typu elementów przechowywanych w wektorze.

* pop_back nie przyjmuje argumentów. Błędem jest użycie pop_back na pustym wektorze.

insert ma trzy formy:

- insert (pos, T & x), który wstawi pojedynczy element xw pozycję pos w wektorze.

-insert (pos, start, end), które wstawia sekwencję elementów z innego kontenera

-położenie pos w wektorze. Sekwencja elementów jest identyfikowana jako rozpoczynająca się od elementu początkowego i kontynuująca, ale nie w tym elemencie końcowym

-insert (pos, int, T & x) wstawia powtórzenia kopii x w pozycji pos w wektorze.

Jak wskazano w powyższym kodzie, pozycja pos powinna być adresem elementu do wstawienia w, podczas gdy argumenty początkowy i końcowy również są adresami. (Prawdziwą historią jest to, że są to iteratory - patrz następny podrozdział i następna sekcja).

* Kasowanie ma dwie formy (pos, początek i koniec mają te same typy, co dla funkcji wstawiania):

- erase(pos), które usunie element w pozycji pos w wektorze.

-insert (start, end), które usunie elementy zaczynające się od pozycji start up do, ale nie włączając elementu na końcu pozycji.

Iteratory wektorowe

Prosty sposób przechodzenia przez elementy wektora v jest taki, jak to zrobiliśmy powyżej:

for (int i = 0; i <v.size (); i ++) {... v [i] ...}

Innym sposobem jest użycie iteratorów. Iterator można uważać za wskaźnik w pojemniku, a inkrementowanie iteratora pozwala przejść przez pojemnik. W przypadku typów kontenerów innych niż wektory, iteratory są jedynym sposobem na przejście przez kontener. Dla wektora zawierającego elementy typu T:

wektor <T> v;

Iterator jest zadeklarowany w następujący sposób:

wektor <T> :: iterator i;

Takie iteratory są konstruowane i zwracane przez funkcje begin () i end (). Możesz porównać dwa iteratory (tego samego typu) używając == i! =, Inkrementując używając ++ i dereferencji używając *. [W rzeczywistości wektorowe iteratory pozwalają na więcej operacji na nich - zobacz następną sekcję po więcej informacji]. Oto ilustracja użycia iteratorów z wektorami.

<vector−iterator.cc>=

```
#include <iostream.h>

#include <vector.h>

void main()

{

vector<int> v(10);
```

```
first is ``less'' than the second

int j = 1;

vector<int>::iterator i;

// Fill the vector v with integers 1 to 10.

i = v.begin();

while (i != v.end())

{

*i = j;

j++;

i++;

}

// Square each element of v.

for (i=v.begin(); i!=v.end(); i++) *i = (*i) * (*i);

// Print out the vector v.

cout << "The vector v contains: ";

for (i=v.begin(); i!=v.end(); i++) cout << *i << ' ';

cout << endl;

}
```

Zwróć uwagę, jak * i można użyć po lewej stronie instrukcji przypisania, aby zaktualizować element wskazany przez i, a po prawej stronie, aby uzyskać dostęp do bieżącej wartości.

Porównywanie wektorów

Możesz porównać dwa wektory używając == i <. == zwróci true tylko wtedy, gdy oba wektory mają tę samą liczbę elementów i wszystkie elementy są równe. Funkcje <wykonuje leksykograficzne porównanie dwóch wektorów. Działa to poprzez porównywanie elementów wektorów po elemencie. Załóżmy, że porównujemy v1 i v2 (czyli v1 <v2?). Ustaw i = 0. Jeśli v1 [i] <v2 [i] następnie zwraca wartość true, jeśli v1 [i]> v2 [i] następnie zwraca wartość false, w przeciwnym razie inkrementuje i (czyli przechodzi do następnego elementu). Jeśli koniec v1 zostanie osiągnięty, zanim v2 zwróci true, w przeciwnym razie zwróci false. Kolejność leksykograficzna znana jest również jako kolejność słowników. Kilka przykładów:

(1,2,3,4) < (5,6,7,8,9,10) is false.

(1,2,3) < (1,2,3,4) is true

(1,2,3,4) < (1,2,3) is false

(0,1,2,3) < (1,2,3) is true

Poniższy kod ilustruje trzeci przykład powyżej.

```
<vector−comp.cc>=

#include <vector.h>

#include <iostream.h>

void main()

{

vector<int> v1;

vector<int> v2;

for (int i=0; i<4; i++) v1.push_back(i+1);

for (int i=0; i<3; i++) v2.push_back(i+1);

cout << "v1: ";

for (int i=0; i<v1.size(); i++) cout << v1[i] << ' ';

cout << endl;

cout << "v2: ";

for (int i=0; i<v2.size(); i++) cout << v2[i] << ' ';

cout << endl;

cout << "v1 < v2 is: " << (v1<v2 ? "true" : "false") << endl;

}
```

21. Zbiory

Ustawiony typ kontenera umożliwia użytkownikowi przechowywanie i pobieranie elementów bezpośrednio, a nie poprzez indeks do kontenera. Ustawiony kontener działa jak zbiór matematyczny, ponieważ zawiera tylko odrębne elementy. Jednak w przeciwieństwie do zestawu matematycznego, elementy w ustawionym pojemniku są przechowywane w zamówieniu (dostarczanym przez użytkownika). W praktyce jest to tylko drobne ograniczenie w traktowaniu zestawu kontenerów jako implementacji abstrakcyjnego typu zbioru danych matematycznych i pozwala na znacznie bardziej wydajną implementację niż podejście nieuporządkowane.

Budowanie zbiorów

Do skonstruowania zbioru kontenerów potrzebne są dwa argumenty szablonu - typ obiektów, które zestaw ma zawierać, oraz obiekt funkcji, który może porównywać dwa elementy danego typu, czyli:

set<T, Compare> s;

(Zbiór deklaracji <T> powinien być również możliwy - użyłby domyślnego argumentu szablonu mniej <T> jako drugiego argumentu, ale wiele kompilatorów C ++ (w tym g ++) nie może jeszcze poradzić sobie z domyślnymi argumentami szablonu.) Dla prostych typy T możemy użyć obiektu function less <T> (bez martwienia się o co obiekt funkcji '' to), na przykład wszystkie poniższe są ustawionymi deklaracjami prawnymi.

set<int, less<int> > s1;

set<double, less<double> > s2;

set<char, less<char> > s3;

set<string, less<string> > s4;

Zwróć uwagę, że wymagana jest przestrzeń między dwoma ostatnimi> w szablonie - w przeciwnym razie kompilator zinterpretuje >> jako operator prawego przesunięcia.) W każdym z tych przypadków obiekt funkcji wykorzystuje operatora <jak zdefiniowano dla typ podstawowy (tj. int, double, char i string). Poniższy kod deklaruje zbiór liczb całkowitych, a następnie dodaje niektóre liczby całkowite do zestawu za pomocą metody wstawiania a następnie wypisuje wybrane elementy, wykonując iterację przez zestaw. Zauważysz, że zawartość zestawu jest drukowana w porządku rosnącym, mimo że zostały dodane w dowolnej kolejności.

<set−construct1.cc>=

#include <iostream.h>

#include <set.h>

void main()

{

set<int, less<int> > s;

set<int, less<int> >::iterator i;

s.insert(4);

s.insert(0);

s.insert(−9);

s.insert(7);

s.insert(−2);

s.insert(4);

s.insert(2);

cout << The set contains the elements: ;

for (i=s.begin(); i!=s.end(); i++) cout << *i << ' ';

cout << endl;

}

Zwróć uwagę, że 4 jest dodane dwa razy, ale pojawia się tylko raz na liście elementów - czyli tego, czego oczekuje się od zestawu.

Czym są obiekty funkcyjne?

Jedną ze znakomitych funkcji języka C ++ jest możliwość przeciążania operatorów, dzięki czemu można mieć + znaczące to, co się lubi dla nowo zaprojektowanej klasy. Jednym z operatorów C ++, który pozwala przeciążać, jest operator wywołania funkcji (), który pozwala tworzyć klasy, których instancje mogą zachowywać się jak funkcje na wiele sposobów. Są to obiekty funkcyjne. Oto prosty przykład.

```
<function−object.cc>=

#include <iostream.h>

template<class T>

class square {

public:

T operator()(T x) { return x*x; }

};

// This can be used with any T for which * is defined.

void main()

{

// Create some function objects.

square<double> f1;

square<int> f2;

// Use them.

cout << 5.1^2 = << f1(5.1) << endl;

cout << 100^2 = << f2(100) << endl;

// The following would result in a compile time error.

// cout << 100.1^2 = << f2(100.1) << endl;

}
```

Obiekty funkcyjne są używane w wielu miejscach w STL. W szczególności są używane podczas deklarowania zestawów i map. Obiekt funkcji wymagany do tych celów, załóżmy, że nazywa się to comp, musi spełniać następujące wymagania.

1. Jeśli comp (x, y) i comp (y, z) są prawdziwe dla obiektów x, y i z, to comp (x, z) również jest prawdziwe.

2. comp (x, x) jest fałszywe dla każdego obiektu x.

Jeśli dla jakiegoś konkretnego obiektu x i y, oba comp (x, y) i comp (y, x) są fałszywe, to x i y są uważane za równe. W rzeczywistości jest to zachowanie ściśle niższej relacji (tj. <) Na liczbach. Obiekt funkcji less <T> użyty powyżej jest zdefiniowany w terminach operatora <dla typu T. Jego definicję można uważać za następującą.

```
template<class T>

struct less {

bool operator()(T x, T y) { return x<y; }

}
```

(Rzeczywista definicja używa odniesień, ma odpowiednie adnotacje const i dziedziczy po klasie szablonu binary_function.)

Oznacza to, że jeśli typ T ma operator <zdefiniowany dla niego to możesz użyć mniej <T> jako komparatora podczas deklarowania zestawów T. Możesz nadal chcieć użyć komparatora specjalnego celu, jeśli dostarczony operator <nie jest odpowiedni dla twojego cele. Oto inny przykład. Definiuje to prostą klasę z definicją operatora <i obiektu funkcji, który wykonuje inne porównanie. Należy zauważyć, że przeciążonym operatorom <i () należy nadać adnotacje w postaci stałej, aby funkcje te działały poprawnie ze STL.

<set−construct2.cc>=

```
#include <iostream.h>

#include <set.h>

// This class has two data members. The overloaded operator< compares

// such classes on the basis of the member f1.

class myClass {

private:

int f1;

char f2;

public:

myClass(int a, char b) : f1(a), f2(b) {}

int field1() const { return f1; }

char field2() const { return f2; }

bool operator<(myClass y) const

{ return (f1<y.field1()); }

};

// This function object compares objects of type myClass on the basis

// of the data member f2.

class comp_myClass {

public:

bool operator()(myClass c1, myClass c2) const

{ return (c1.field2() < c2.field2()); }

};

void main()

{
```

```
set<myClass, less<myClass> > s1;

set<myClass, less<myClass> >::iterator i;

set<myClass, comp_myClass> s2;

set<myClass, comp_myClass>::iterator j;

s1.insert(myClass(1,'a'));

s2.insert(myClass(1,'a'));

s1.insert(myClass(1,'b'));

s2.insert(myClass(1,'b'));

s1.insert(myClass(2,'a'));

s2.insert(myClass(2,'a'));

cout << Set s1 contains: ;

for (i=s1.begin(); i!=s1.end(); i++)

{ cout << ( << (*i).field1() << ,

<< (*i).field2() << ) << ' ';

}

cout << endl;

cout << Set s2 contains: ;

for (j=s2.begin(); j!=s2.end(); j++)

{ cout << ( << (*j).field1() << ,

<< (*j).field2() << ) << ' ';

}

cout << endl;

}
```

Zbiór s1 zawiera (1, a) i (2, a), ponieważ porównanie jest na członie danych f1, tak że (1, a) i (1, b) są uważane za ten sam element. Zbiór s2 zawiera (1, a) i (1, b), ponieważ porównanie jest na członie danych f2, tak że (1, a) i (2, a) są uważane za ten sam element.

Narzędzie wyświetlania

Sposób, w jaki wyświetlaliśmy zestawy w poprzednich przykładach, jest trochę niezręczny, dlatego został zapisany następujący plik nagłówkowy zawierający prostą przeciążoną wersję operatora <<. Działa dobrze dla małych zestawów z prostymi typami elementów.

```
<printset.h>=

#ifndef _PRINTSET_H
```

```
#define _PRINTSET_H

#include <iostream.h>

#include <set.h>

template<class T, class Comp>

ostream& operator<<(ostream& os, const set<T,Comp>& s)

{

set<T,Comp>::iterator iter = s.begin();

int sz = s.size();

int cnt = 0;

os << {;

while (cnt < sz−1)

{

os << *iter << ,;

iter++;

cnt++;

}

if (sz != 0) os << *iter;

os << };

return os;

}

#endif
```

Użycie tutaj << jako procedury wyjściowej dla zestawu zakłada, że << zostało zdefiniowane dla zestawów elementów i używa tego do drukowania rozdzielonej przecinkami listy ustawionych elementów owiniętych w nawiasy klamrowe. Zostanie użyty bez komentarza w poniższych przykładach.

Ile elementów?

Możesz określić, czy zestaw jest pusty, czy nie, używając metody empty (). Możesz dowiedzieć się, ile elementów znajduje się w zestawie, używając metody size (). Te metody nie pobierają żadnych argumentów, empty () zwraca true lub false, a size () zwraca liczbę całkowitą.

```
<set−size.cc>=

#include <iostream.h>

#include <set.h>

#include printset.h
```

```
void main()

{

set<int, less<int> > s;

cout << The set s is

<< (s.empty() ? empty. : non−empty.) << endl;

cout << It has << s.size() << elements. << endl;

cout << Now adding some elements... << endl;

s.insert(1);

s.insert(6);

s.insert(7);

s.insert(−7);

s.insert(5);

s.insert(2);

s.insert(1);

s.insert(6);

cout << The set s is now

<< (s.empty() ? empty. : non−empty.) << endl;

cout << It has << s.size() << elements. << endl;

cout << s = << s << endl;

}
```

Sprawdzanie równości zbiorów

Dwa zestawy mogą być sprawdzane pod kątem równości za pomocą ==. Ten test równości działa poprzez testowanie w kolejności odpowiadających elementów każdego zestawu dla równości przy użyciu T :: operator ==.

```
<set−equality.cc>=

#include <iostream.h>

#include <set.h>

#include printset.h

void main()

{

set<int, less<int> > s1, s2 ,s3;

for (int i=0; i<10; i++)
```

{

s1.insert(i);

s2.insert(2*i);

s3.insert(i);

}

cout << s1 = << s1 << endl;

cout << s2 = << s2 << endl;

cout << s3 = << s3 << endl;

cout << s1==s2 is: << (s1==s2 ? true. : false.) << endl;

cout << s1==s3 is: << (s1==s3 ? true. : false.) << endl;

}

Możliwe jest również porównanie dwóch zestawów przy użyciu <. Porównanie s1 <s2 jest prawdziwe, jeśli zbiór s1 jest leksykograficznie mniejszy od zbioru s2, w przeciwnym razie jest on fałszywy.

Dodawanie i usuwanie elementów

Sposób dodawania elementów do zestawu polega na użyciu metody wstawiania (tak jak to zrobiliśmy powyżej). Sposobem na usunięcie elementów ze zbioru jest użycie metody kasowania. W przypadku zestawu elementów trzymających typu T metody te występują w następujących postaciach:

* pair <iterator, bool> insert (T & x). Jest to standardowa funkcja wstawiania. Wartość zwracana może zostać zignorowana lub użyta do sprawdzenia, czy wstawienie zakończyło się powodzeniem (to znaczy, że element nie był jeszcze w zestawie). Jeśli wstawienie zakończy się pomyślnie, komponent logiczny będzie prawdziwy, a iterator wskaże właśnie wstawiony element. Jeśli element jest już obecny, komponent boolowski będzie miał wartość false, a iterator wskaże element x, który już istnieje.

* insert iteracyjny (pozycja iteratora, T & x). Ta wersja funkcji wstawiania pobiera, oprócz elementu do wstawienia, iterator z informacją, gdzie funkcja wstawiania powinna rozpocząć wyszukiwanie. Iterator wskazuje na nowo wstawiony element (lub już obecny element).

· * Int erase (T & x). Ta wersja metody kasowania pobiera element i usuwa 1, jeśli element był obecny (i usuwa) lub 0, jeśli element nie był obecny.

· * Void erase (pozycja iteratora). Ta wersja przyjmuje iterator wskazujący na jakiś element zestawu i usuwa ten element.

* void erase (pierwszy iterator, ostatni iterator). Ta wersja pobiera dwa iteratory wskazujące na zestaw i usuwa wszystkie elementy z zakresu [pierwszy, ostatni].

Poniższy przykład ilustruje te różne formy.

<set−add−delete.cc>=

#include <iostream.h>

```
#include <set.h>
#include printset.h
void main()
{
set<int, less<int> > s1;
// Insert elements in the standard fashion.
s1.insert(1);
s1.insert(2);
s1.insert(−2);
// Insert elements at particular positions.
s1.insert(s1.end(), 3);
s1.insert(s1.begin(), −3);
s1.insert((s1.begin()++)++, 0);
cout << s1 = << s1 << endl;
// Check to see if an insertion has been successful.
pair<set<int, less<int> >::iterator,bool> x = s1.insert(4);
cout << Insertion of 4 << (x.second ? worked. : failed.)
<< endl;
x = s1.insert(0);
cout << Insertion of 0 << (x.second ? worked. : failed.)
<< endl;
// The iterator returned by insert can be used as the position
// component of the second form of insert.
cout << Inserting 10, 8 and 7. << endl;
s1.insert(10);
x=s1.insert(7);
s1.insert(x.first, 8);
cout << s1 = << s1 << endl;
// Attempt to remove some elements.
cout << Removal of 0 << (s1.erase(0) ? worked. : failed.)
<< endl;
```

```
cout << Removal of 5 << (s1.erase(5) ? worked. : failed.)

<< endl;

// Locate an element, then remove it. (See below for find.)

cout << Searching for 7. << endl;

set<int,less<int> >::iterator e = s1.find(7);

cout << Removing 7. << endl;

s1.erase(e);

cout << s1 = << s1 << endl;

// Finally erase everything from the set.

cout << Removing all elements from s1. << endl;

s1.erase(s1.begin(), s1.end());

cout << s1 = << s1 << endl;

cout << s1 is now << (s1.empty() ? empty. : non−empty.)

<< endl;

}
```

Znajdowanie elementów

Wspominamy o dwóch funkcjach składowych, które można wykorzystać do sprawdzenia, czy element występuje w zbiorze, czy nie.

* iterator find (T & x). Wyszukuje element x w zestawie. Jeśli zostanie znalezione x, zwraca iterator wskazujący na x w przeciwnym razie zwraca end ().

· * Int count (T & x). Zwraca 1, jeśli znajdzie x w zbiorze i 0 w przeciwnym razie. (Funkcja count dla multisets zwraca liczbę kopii elementu w zbiorze, która może być większa niż 1. Stąd, jak sądzę, nazwa funkcji.)

· Wykorzystanie find zostało zilustrowane powyżej. Moglibyśmy użyć count, aby napisać prostą funkcję przynależenia do zestawu opartego na szablonie. (Powinno to również zapewnić wersję, która odwołuje się do argumentu x.)

Ustaw operacje teoretyczne

STL dostarcza jako ogólne algorytmy, w których ustawione operacje obejmują, połączenie, przecięcie, różnicę i symetryczną różnicę. Aby uzyskać dostęp do tych funkcji, musisz włączyć algo.h. (W dalszej części iter oznacza odpowiedni iterator).

* bool zawiera (iter f1, iter l1, iter f2, iter l2).

To sprawdzi, czy zestaw reprezentowany przez zakres [f2, l2] jest zawarty w zbiorze [f1, l1]. Zwraca true, jeśli jest, i false w przeciwnym razie. Aby sprawdzić, czy jeden zestaw jest zawarty w innym, użyłbyś (s1.begin (), s1.end (), s2.begin (), s2.end ()) Funkcja include sprawdza prawdę 3 # 3 (czyli 4 # 4). Ta funkcja zakłada, że zbiory są uporządkowane za pomocą operatora porównania <. Jeśli użyto

jakiegoś innego operatora porównania, musi on zostać przekazany jako dodatkowy argument (obiekt funkcji) po innych argumentach.

* iter set_union (iter f1, iter l1, iter f2, iter l2, wynik itera).

Tworzy to połączenie zbiorów reprezentowanych przez zakresy [f1, l1] i [f2, l2]. Wynik argumentu jest iteratorem wyjścia, który wskazuje na początku zestawu, który będzie utrzymywał związek. Zwracaną wartością funkcji jest iterator wyjścia, który wskazuje na końcu nowego zestawu.

Fakt, że argument wynikowy jest iteratorem wyjścia, oznacza, że nie można używać set_union w następujący, naturalny sposób:

set<int, less<int> > s1, s2, s3;

// Add some elements to s1 and s2 ...

// Then form their union. (This does not work!)

set_union(s1.begin(), s1.end(),

s2.begin(), s2.end(),

s3.begin());

Powodem jest, że begin () (również end ()), gdy używane z zestawami (lub mapami) zwraca (stały) iterator wejściowy. Ten rodzaj iteratora umożliwia dostęp do elementów zestawu do czytania, ale nie do zapisu. (I to jest dobra rzecz, ponieważ jeśli mógłbyś przypisać do iteratora dereferencyjnego (jak w (* i) = ...), to możesz zniszczyć podstawową kolejność zestawu.) Rozwiązaniem jest użycie iteratora insertowego opartego na ustawionym typie. To, w zasadzie, konwertuje przypisanie (* i) = wartość (która jest nielegalna) do (legalnego) wstawienia s.insert (i, wartość) (gdzie s jest ustawionym obiektem, na który wskazuje iterator i). Jest używany w następujący sposób:

// Typedef for convenience.

typedef set<int, less<int> > intSet;

intSet s1, s2, s3;

// Add some elements to s1 and s2 ...

// Then form their union.

set_union(s1.begin(), s1.end(),

s2.begin(), s2.end(),

insert_iterator<intSet>(s3,s3.begin()))

Oto przykład ilustrujący wszystkie te operacje.

<set−theory.cc>=

#include <iostream.h>

#include <set.h>

#include <algo.h>

#include <iterator.h>

```
#include printset.h
void main()
{
typedef set<int, less<int> > intSet;
intSet s1, s2, s3, s4;
for (int i=0; i<10; i++)
{ s1.insert(i);
s2.insert(i+4);
}
for (int i=0; i<5; i++) s3.insert(i);
cout << s1 = << s1 << endl;
cout << s2 = << s2 << endl;
cout << s3 = << s3 << endl;
// Is s1 a subset of s2?
bool test = includes(s2.begin(),s2.end(),s1.begin(),s1.end());
cout << s1 subset of s2 is << (test ? true. : false.) << endl;
// Is s3 a subset of s1?
test = includes(s1.begin(),s1.end(),s3.begin(),s3.end());
cout << s3 subset of s1 is << (test ? true. : false.) << endl;
// Form the union of s1 and s2.
set_union(s1.begin(), s1.end(), s2.begin(), s2.end(),
insert_iterator<intSet>(s4,s4.begin()) );
cout << s1 union s2 = << s4 << endl;
// Erase s4 and form intersection of s1 and s2. (If we don't erase
// s4 then we will get the previous contents of s4 as well).
s4.erase(s4.begin(),s4.end());
set_intersection(s1.begin(), s1.end(), s2.begin(), s2.end(),
insert_iterator<intSet>(s4,s4.begin()) );
cout << s1 intersection s2 = << s4 << endl;
// Now set difference.
s4.erase(s4.begin(),s4.end());
```

```
set_difference(s1.begin(), s1.end(), s2.begin(), s2.end(),

insert_iterator<intSet>(s4,s4.begin()) );

cout << s1 minus s2 = << s4 << endl;

// Set difference is not symmetric.

s4.erase(s4.begin(),s4.end());

set_difference(s2.begin(), s2.end(), s1.begin(), s1.end(),

insert_iterator<intSet>(s4,s4.begin()) );

cout << s2 minus s1 = << s4 << endl;

// Finally symmetric difference.

s4.erase(s4.begin(),s4.end());

set_symmetric_difference(s1.begin(), s1.end(), s2.begin(), s2.end(),

insert_iterator<intSet>(s4,s4.begin()) );

cout << s1 symmetric_difference s2 = << s4 << endl;

// Which is symmetric!

s4.erase(s4.begin(),s4.end());

set_symmetric_difference(s2.begin(), s2.end(), s1.begin(), s1.end(),

insert_iterator<intSet>(s4,s4.begin()) );

cout << s2 symmetric_difference s1 = << s4 << endl;

}
```

22 Projektowanie klasy wątków w C ++

Programowanie wielowątkowe staje się coraz bardziej popularne. Ta sekcja przedstawia projekt klasy C ++, która będzie enkapsulować mechanizm wątków. Niektóre aspekty programowania nici, takie jak muteksy i semafory, nie są tutaj omawiane. Ponadto wywołania systemu operacyjnego do manipulowania wątkami są pokazane w ogólnym formacie.

Krótkie wprowadzenie do wątków

Aby zrozumieć wątki, trzeba pomyśleć o kilku programach działających jednocześnie. Wyobraź sobie ponadto, że wszystkie te programy mają dostęp do tego samego zestawu zmiennych globalnych i wywołań funkcji. Każdy z tych programów reprezentowałby wątek wykonania i dlatego jest nazywany wątkiem. Ważnym rozróżnieniem jest to, że każdy wątek nie musi czekać na kontynuację żadnego innego wątku. Wszystkie wątki przebiegają jednocześnie. Aby użyć metafory, są jak biegacze w wyścigu, żaden biegacz nie czeka na kolejnego biegacza. Wszystkie postępują według własnego kursu. Po co używać wątków, o które możesz zapytać. Nici często mogą poprawić wydajność aplikacji i nie powodują znacznego nakładu pracy związanej z implementacją. Skutecznie dają dobry wybuch za złotówki. Wyobraź sobie program serwera obrazów, który musi obsługiwać żądania dotyczące obrazów. Program otrzymuje żądanie dla obrazu z innego programu. Następnie musi pobrać obraz z bazy danych i wysłać go do programu, który o to poprosił. Jeśli serwer był zaimplementowany w

pojedynczym trybie gwintowania, tylko jeden program mógł zażądać jednocześnie. Gdy zajęło pobieranie obrazu i wysłanie go do żądającego, nie mogło obsługiwać innych żądań. Oczywiście nadal można wdrożyć taki system bez użycia wątków. Byłoby jednak wyzwaniem. Za pomocą wątków można bardzo naturalnie zaprojektować system do obsługi wielu żądań. Prostym podejściem byłoby utworzenie wątku dla każdego otrzymanego żądania. Główny wątek utworzyłby ten wątek po otrzymaniu żądania. Wątek byłby wtedy odpowiedzialny za konwersację z programem klienta. Po pobraniu obrazu wątek zakończyłby się. Zapewniłoby to sprawny system, który nadal obsługiwałby żądania, mimo że był zajęty obsługą innych żądań w tym samym czasie.

Podejście podstawowe

Utwórz wątek, musisz określić funkcję, która stanie się punktem wejścia dla wątku. Na poziomie systemu operacyjnego jest to normalna funkcja. Musimy wykonać kilka sztuczek, aby owinąć wokół niego klasę C ++, ponieważ funkcja wejścia nie może być normalną funkcją członka klasy. Może to jednak być statyczna funkcja członkowska klasy. To właśnie wykorzystamy jako punkt wejścia. Tutaj jest jednak chęć. Statyczne funkcje członkowskie nie mają dostępu do tego wskaźnika obiektu C ++. Mogą uzyskać dostęp tylko do danych statycznych. Na szczęście istnieje sposób, aby to zrobić. Funkcje punktu wejścia wątku przyjmują wartość void * jako parametr, dzięki czemu osoba wywołująca może typować dane i przekazywać je do wątku. Wykorzystamy to, aby przekazać to do funkcji statycznej. Statyczne funkcja następnie wypisze typ pustki * i użyje jej do wywołania niestatycznej funkcji składowej.

Implementacja

Należy wspomnieć, że omówimy klasę wątków o ograniczonej funkcjonalności. Można zrobić więcej za pomocą wątków, niż pozwoli na to klasa

```
class Thread

{

public:

Thread();

int Start(void * arg);

protected:

int Run(void * arg);

static void * EntryPoint(void*);

virtual void Setup();

virtual void Execute(void*);

void * Arg() const {return Arg_;}

void Arg(void* a){Arg_ = a;}

private:

THREADID ThreadId_;

void * Arg_;
```

```
};
Thread::Thread() {}
int Thread::Start(void * arg)
{
Arg(arg); // store user data
int code = thread_create(Thread::EntryPoint, this, & ThreadId_);
return code;
}
int Thread::Run(void * arg)
{
Setup();
Execute( arg );
}
/*static */
void * Thread::EntryPoint(void * pthis)
{
Thread * pt = (Thread*)pthis;
pthis−>Run( Arg() );
}
virtual void Thread::Setup()
{
// Do any setup here
}
virtual void Thread::Execute(void* arg)
{
// Your code goes here
}
```

Ważne jest, aby zrozumieć, że owijamy obiekt C ++ wokół wątku. Każdy obiekt zapewni interfejs do pojedynczego wątku. Wątek i obiekt nie są takie same. Obiekt może istnieć bez wątku. W tej implementacji wątek nie istnieje, dopóki nie zostanie wywołana funkcja Start. Zwróć uwagę, że przechowujemy argument użytkownika w klasie. Jest to konieczne, ponieważ potrzebujemy miejsca do tymczasowego przechowywania go, dopóki wątek nie zostanie uruchomiony. Wywołanie wątku systemu operacyjnego pozwala nam przekazać argument, ale użyliśmy go do przekazania tego

wskaźnika. Przechowujemy więc rzeczywisty argument użytkownika w samej klasie, a po wywołaniu funkcji execute może uzyskać dostęp do argumentu.

Thread(); To jest konstruktor.

int Start (void * arg); Ta funkcja umożliwia utworzenie wątku i uruchomienie go. Argument arg zapewnia sposób przekazywania danych użytkownika do wątku. Start () tworzy wątek, wywołując funkcję tworzenia wątku systemu operacyjnego.

int Run (void * arg); Jest to funkcja chroniona, która nigdy nie powinna być naruszona.

static void * EntryPoint (void * pthis); Ta funkcja służy jako punkt wejścia do wątku. Po prostu rzuca pthis na wątek * i

virtual void Setup (); Ta funkcja jest wywoływana po utworzeniu wątku, ale przed wywołaniem Execute (). Jeśli zastąpisz tę funkcję, pamiętaj, aby wywołać klasę rodzica Execute (). wirtualna pustka Execute (void *); Musisz zastąpić tę funkcję, aby zapewnić własną funkcjonalność.

Używanie klasy wątków

Aby użyć klasy wątku, możesz uzyskać nową klasę. zastępujesz funkcję Execute (), w której podasz swoją własną funkcjonalność. Możesz zastąpić funkcję Setup (), aby wykonać wszelkie obowiązki uruchamiania przed wywołaniem Execute. Jeśli nadpiszesz Setup (), pamiętaj, aby wywołać klasę Parent Setup ().

Wniosek

W tej sekcji przedstawiono implementację klasy wątków napisanej w C ++. Oczywiście jest to proste podejście, ale zapewnia solidną podstawę, na której można zbudować solidniejszą konstrukcję.