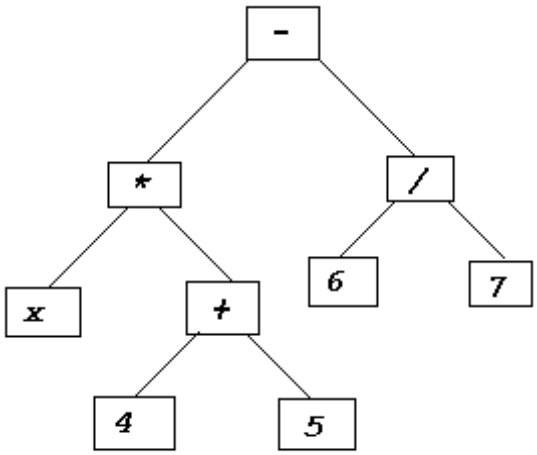
Projekt **fEval**

1.Wstęp

Głównym założeniem projektu jest stworzenie klas, umożliwiających obliczanie wartości wyrażeń matematycznych (np. `x^2-3*sin(y/2+2.5)`), dla zadanych wartości zmiennych. Klasy mają posiadać postać szablonową, co umożliwi wykonywanie obliczeń dla różnych typów danych (np. int, double, complex). Wyrażenia w postaci ciągu znaków będą przetwarzane tylko raz (dla danej formuły) na drzewo, a następnie szybko obliczane rekurencyjnym algorytmem. Identyfikatory funkcji i zmiennych będą przechowywane w strukturze słownikowej.

2. Sposób działania

Załóżmy, że chcemy obliczyć wartość formuły `x*(4+5)-6/7`.W tym celu należy zbudować drzewo wyrażenia o następującej postaci:



Każdy prostokąt (węzeł drzewa) będziemy nazywać **unitem**. Jego zadaniem jest zwracanie wartości określonego typu. Klasą bazową dla każdego unitu jest wirtualna klasa **`CBaseUnit`**. Można wyszczególnić następujące klasy potomne:

- `CValUnit` zwraca stałą wartość (w przykładzie bloki z wartościami 4, 5, 6, 7)
- `CVarUnit` zwraca wartość zmiennej (w przykładzie blok z wartością x)
- `C1OpUnit` zwraca wartość funkcji jednoargumentowej
- `C2OpUnit` zwraca wartość funkcji dwuargumentowej (w przykładzie funkcje odpowiadające operatorom -, *, +, /)

Klasy `C1OpUnit` oraz `C2OpUnit` posiadają odpowiednio jeden i dwa wskaźniki na klasę `CBaseUnit`. Ponadto posiadają adres do wywoływanej funkcji. Klasy `CValUnit` i `CVarUnit` posiadają jedno pole, przechowujące wartość lub adres do zmiennej.
Obliczenie wartości wyrażenia odbywa się w prosty, rekurencyjny sposób. Jeżeli unit jest bezargumentowy, to zwracana jest ustalona wartość. Jeżeli unit posiada argumenty, to najpierw obliczane są wartości kolejnych argumentów, a następnie zwracana jest wartość wywoływanej funkcji dla obliczonych wcześniej wartości argumentów. Obliczenie wartości wrażenia odbywa się przez wywołanie `NazwaKlasy()`.

Za tworzenie drzewa wyrażenia oraz jego wywoływanie odpowiada klasa **`CEval`**. Posiada ona metodę `Set`, która tworzy drzewo wyrażenia na podstawie danej formuły. Ponadto możemy przekazywać w niej listę używanych zmiennych lokalnych.

Przykład:

```
CEval<float> eval;
vector<char*> varl;

varl.push_back("x");
varl.push_back("y");
eval.Set("sin(x-y)+cos(x+y)+2.345", varl);

vector<float> vall;

vall.push_back(3);
vall.push_back(5*5-2.f);
std::cout << eval(vall) << std::endl; //2.07897
```

Powstaje pytanie: Jak interpretowane są wartości liczbowe w wyrażeniach? Dokonywana jest najpierw konwersja ciągu znaków na typ `double`, a następnie dokonywane jest przypisanie tej liczby, do pola `val` klasy `CValUnit`. Może się oczywiście zdarzyć, że taka konwersja będzie niemożliwa.

Przekazywane w przykładzie zmienne x i y są tworzone tymczasowo, przechowywane aż do kolejnego wywołania metody `Set`. Możemy też w ogóle nie przekazywać żadnej listy, używając tylko zmiennych i funkcji globalnych, zdefiniowanych w bibliotece standardowej programu (`stdlib`). Możemy też dodawać nowe zmienne globalne do biblioteki rozszerzającej bibliotekę standardową (`extlib`) używając metody `AddVar`. Adres do zmiennej, określanej przez ciąg znaków, można uzyskać poprzez metodę `GetVarPtr`. Możemy potem usunąć zmienna używając `DelVar`.

Przykład:

```
obliczaj.AddVar("n",0);
int *n=obliczaj.GetVarPtr("n");

obliczaj.Set("n*n+n+41");

std::cout << "5 liczb pierwszych\n";
for (*n=0; *n<5; (*n)++)
{
     std::cout<<"n=" << *n << " f(n)=" << obliczaj() << std::endl;
}

obliczaj.DelVar("n");
```

Pozwala to na szybsze wykonywanie obliczeń gdy chcemy w każdym przebiegu zmienić wartość zmiennej (dokonujemy to bezpośrednio).

Klasa **`CFLib`** jest wykorzystywana przez klasę `CEval` do identyfikacji nazw w parsowanym wyrażeniu i przechowuje wszystkie, jakie możemy w nim wykorzystywać. `stdlib` (statyczna) przechowuje funkcje standardowe: ADD, SUB, MUL, DIV, POW (odpowiadające operatorom +, -, *, /, ^), ABS, CEIL, COS, CTAN, FLOOR, RND, SIN, SGN, TAN oraz stałe PI i E. Biblioteka `extlib` zawiera zmienne zdefiniowane przez użytkownika. Wielkość liter w nazwach nie ma znaczenia, mogą zawierać (nie na początku) cyfry.

3. Dalsze rozszerzenia

Możliwy jest następujący rozwój programu:

- Stworzenie wygodnego interfejsu GUI, z możliwością rysowania funkcji
- Stworzenie klasy `CThread`, która umożliwi wykorzystanie wielu wątków do wykonywania obliczeń
- Funkcje o różnych typach argumentów

Autor projektu: Kamil Korolkiewicz (grupa U1)
E-mail: korolkiewiczk@student.mini.pw.edu.pl
Strona projektu: http://kk.plenty.vbiz.pl/pw/feval.php