

Indywidualny projekt programistyczny (Info, I rok) 16/17

Kokpit ► Moje kursy ► IPP.INFO.I.16/17 ► Temat 11 ► Wielomiany, część 3

Wielomiany, część 3

Zadanie wielomiany część 3

W trzeciej części zadania należy rozbudować program z poprzednich części i dodać do niego testy jednostkowe.

Biblioteka operacji na wielomianach

Do stworzonej w części 1 zadania biblioteki operacji na wielomianach (pliki `poly.h` i `poly.c`) należy dodać funkcję

```
Poly PolyCompose(const Poly *p, unsigned count, const Poly x[]);
```

Jeżeli największy indeks zmiennej występujący w jednomianach o wykładniku większym od zera danego wielomianu wynosi `k`, to liczba zmiennych tego wielomianu wynosi `k + 1` (bo zmienne indeksujemy od 0). Funkcja `PolyCompose` zwraca wielomian `p`, w którym pod zmienną x_i podstawiamy wielomian `x[i]`, czyli

```
p(x[0], x[1], ..., x[count - 1], 0, 0, 0, ...)
```

Jeśli `count < k + 1`, to $x_i = 0$ dla $i = \text{count}, \dots, k$, czyli brakujące wartości zmiennych wypełniamy zerami. Jeśli `count` jest równe zero, to funkcja zwraca po prostu wielomian stały będący wartością wielomianu `p` w „zerze”.

Kalkulator

Do stworzonego w części 2 programu kalkulatora należy dodać polecenie

```
COMPOSE count
```

Polecenie to zdejmuje z wierzchołka stosu najpierw wielomian `p`, a potem kolejno wielomiany `x[0]`, `x[1]`, ..., `x[count - 1]` i umieszcza na stosie wynik funkcji `PolyCompose`.

Jeśli w poleceniu `COMPOSE` nie podano parametru lub jest on niepoprawny, program powinien wypisać:

```
ERROR r WRONG COUNT\n
```

gdzie `r` jak poprzednio oznacza numer wiersza, a `\n` – znak przejścia do nowego wiersza.

Przykład 1

Dla danych wejściowych:

```
(1, 2)
(2, 0)+(1, 1)
COMPOSE 1
PRINT
(1, 3)
COMPOSE 1
PRINT
```

Program kalkulatora powinien wypisać:

```
(2, 0)+(1, 2)
(8, 0)+(12, 2)+(6, 4)+(1, 6)
```

Wyjaśnienie do przykładu:

- Pierwsze polecenie `COMPOSE` podstawia wielomian x_0^2 pod x_0 w wielomianie $(2+x_0)$, więc w jego wyniku otrzymujemy wielomian $(2+x_0^2)$.
- Drugie polecenie `COMPOSE` podstawia wielomian $(2+x_0^2)$ pod x_0 w wielomianie x_0^3 , więc w jego wyniku otrzymujemy wielomian $(8+12x_0^2+6x_0^4+x_0^6)$.

Przykład 2

Dla danych wejściowych:

```
((1, 0)+(1, 1), 1)
(1, 4)
(((1, 6), 5), 2)+((1, 0)+(1, 2), 3)+(5, 7)
COMPOSE 2
PRINT
```

Program kalkulatora powinien wypisać:

```
(1, 12)+((1, 0)+(2, 1)+(1, 2), 14)+(5, 28)
```

Wyjaśnienie do przykładu:

Polecenie `COMPOSE` podstawia do wielomianu $p = x_2^6 x_1^5 x_0^2 + (1+x_1^2)x_0^3 + 5x_0^7$:

- wielomian x_0^4 pod x_0 ,
- wielomian $(1+x_1)x_0$ pod x_1 ,
- 0 pod x_2 .

W rezultacie:

- wyraz $x_2^6 x_1^5 x_0^2$ przechodzi w 0,
- wyraz $(1+x_1^2)$ przechodzi w $(1+(1+2x_1+x_1^2)x_0^2)$,
- wyraz x_0^3 przechodzi w x_0^{12} ,
- wyraz $5x_0^7$ przechodzi w $5x_0^{28}$.

Zatem cały wielomian p przechodzi w wielomian:

$$0 + (1+(1+2x_1+x_1^2)x_0^2)x_0^{12} + 5x_0^{28} = x_0^{12} + (1+2x_1+x_1^2)x_0^{14} + 5x_0^{28}.$$

Przykład 3

Dla danych wejściowych:

```
((1,0)+(1,1),1)
(1,4)
COMPOSE -1
```

Program powinien wypisać na standardowe wyjście diagnostyczne:

```
ERROR 3 WRONG COUNT
```

Dokumentacja

Dodany kod należy udokumentować w formacie Doxygen.

Testy jednostkowe

Należy zaimplementować dwie grupy testów jednostkowych z wykorzystaniem biblioteki `cmocka`.

Pierwsza grupa powinna zawierać testy funkcji `PolyCompose`. Ma zawierać 7 testów przypadków granicznych:

- `p` wielomian zerowy, `count` równe 0,
- `p` wielomian zerowy, `count` równe 1, `x[0]` wielomian stały,
- `p` wielomian stały, `count` równe 0,
- `p` wielomian stały, `count` równe 1, `x[0]` wielomian stały różny od `p`,
- `p` wielomian x_0 , `count` równe 0,
- `p` wielomian x_0 , `count` równe 1, `x[0]` wielomian stały,
- `p` wielomian x_0 , `count` równe 1, `x[0]` wielomian x_0 .

Druga grupa powinna zawierać testy parsowania polecenia `COMPOSE` wykorzystujące atrapy funkcji `main`, `exit`, `printf`, `fprintf`, `scanf` itp. Ma zawierać 8 testów pokrywających różne przypadki użycia parametru `count`:

- brak parametru,
- minimalna wartość, czyli 0,
- maksymalna wartość reprezentowana w typie `unsigned`,
- wartość o jeden mniejsza od minimalnej, czyli -1 ,
- wartość o jeden większa od maksymalnej reprezentowanej w typie `unsigned`,
- duża dodatnia wartość, znacznie przekraczająca zakres typu `unsigned`,
- kombinacja liter,
- kombinacja cyfr i liter, rozpoczynająca się cyfrą.

Z uwagi na automatyczne testowanie rozwiązań, testy powinny być wywoływane dokładnie w wyżej podanej kolejności.

Uwaga: poprawnym zakończeniem testu z maksymalną wartością jest wypisanie komunikatu o błędzie

```
ERROR r STACK UNDERFLOW\n
```

gdyż nie jesteśmy w stanie umieścić na stosie dostatecznej liczby wielomianów.

W obu grupach testów należy testować wycieki pamięci za pomocą atrap funkcji `calloc`, `malloc`, `realloc` i `free`.

Dostarczamy

Rozwiązanie części 3 zadania powinno korzystać z własnego rozwiązania poprzednich jego części. Istnieje możliwość zaimplementowania części 3 zadania, bazując na otrzymanej od nas implementacji. Trzeba jednak o to poprosić (mail do Jakuba Pawlewicza) i w takim przypadku ocena z części 1 i 2 zostanie zmniejszona o połowę. Do części 3 zadania nie dostarczymy testów.

Wymagamy

Jako rozwiązanie części 3 zadania wymagamy:

- zachowania struktury plików z poprzednich części,
- zmodyfikowania plików biblioteki operacji na wielomianach `poly.h` i `poly.c`,
- zmodyfikowania plików kalkulatora,
- umieszczenia testów jednostkowych w pliku `unit_tests_poly.c`,
- uzupełnienia pliku konfiguracyjnego dla programu `cmake`,
- uzupełnienia dokumentacji dla programu `doxygen`.

Gotowe rozwiązanie powinno się kompilować za pomocą sekwencji poleceń:

```
mkdir release
cd release
cmake ..
make
make doc
make test
```

W wyniku wykonania polecenia `make` powinny powstać plik wykonywalny kalkulatora `calc_poly` oraz plik wykonywalny `unit_tests_poly` uruchamiający testy jednostkowe. W wyniku wykonania polecenia `make doc` powinna powstać dokumentacja. Polecenie `make test` uruchamia testy jednostkowe.

W poleceniu `cmake` powinno być również możliwe jawne określenie wariantu budowania pliku wynikowego:

```
cmake -D CMAKE_BUILD_TYPE=Release ..
cmake -D CMAKE_BUILD_TYPE=Debug ..
```

Pliki z rozwiązania poprzednich części zadania i pliki dostarczone przez nas można modyfikować, o ile nie zmienia to działania programu i zachowuje wymagania podane w treści zadania, przy czym nie należy zmieniać opcji kompilacji.

Wymagania dotyczące implementacji

Wymagania dotyczące implementacji są takie same jak w poprzednich częściach zadania.

Oddawanie rozwiązania

Rozwiązanie należy oddawać, podobnie jak poprzednio, przez repozytorium git. W repozytorium mają się znaleźć wszystkie pliki niezbędne do zbudowania plików wykonywalnych `calc_poly` i `unit_tests_poly` oraz dokumentacji. *W repozytorium nie wolno umieszczać plików binarnych ani tymczasowych.* W Moodle jako rozwiązanie należy umieścić tekst zawierający identyfikator commita z finalną wersją rozwiązania, na przykład:

Finalna wersja mojego rozwiązania części 3 zadania wielomiany znajduje się w repozytorium w commicie 518507a7e9ea50e099b33cb6ca3d3141bc1d6638.

Rozwiązanie należy zatwierdzić (git commit) i wysłać do repozytorium (git push) najpóźniej do godz. 23.55 dnia

- 1.06 – termin 0,
- 7.06 – termin 1,
- 21.06 – termin 2.

Każde rozwiązanie oddane w terminie *i* będzie oceniane. Ostateczna ocena za to zadanie będzie wynosiła tyle ile ocena za ostatnie sprawdzanie. Termin 2 jest ostateczny i nie można wysłać rozwiązania po tym terminie.

Punktacja

Za w pełni poprawne rozwiązanie zadania implementujące wszystkie wymagane funkcjonalności można zdobyć maksymalnie 20 punktów. Od tej oceny będą odejmowane punkty karne za poniższe uchybienia:

- za każdy test, którego program nie przejdzie, traci się 1 punkt;
- za wycieki pamięci można stracić do 6 punktów;
- za błędy w testach jednostkowych można stracić do 10 punktów;
- za niezgodną ze specyfikacją strukturę katalogów i plików w rozwiązaniu lub umieszczenie w repozytorium niepotrzebnych albo tymczasowych plików można stracić do 2 punktów;
- za błędy w stylu kodowania można stracić do 3 punktów;
- za braki w dokumentacji można stracić do 2 punktów.

Rozwiązania należy implementować *samodzielnie* pod rygorem niezaliczenia przedmiotu.

Status przesłanego zadania

Numer próby	To jest próba nr 1.
Status przesłanego zadania	Nie próbowano
Stan oceniania	Nie ocenione
Termin oddania	środa, 7 czerwiec 2017, 23:55
Pozostały czas	13 dni 1 godz.
Ostatnio modyfikowane	-
Komentarz do przesłanego zadania	► Komentarze (0)

Dodaj swoją pracę

Dodaj lub edytuj swoje zadanie

NAWIGACJA



Kokpit

- Strona główna

Strony

Moje kursy

IPP.INFO.I.16/17

Uczestnicy

 Odznaki

 Kompetencje

 Oceny

Główne składowe

Temat 1

Temat 2

Temat 3

Temat 4

Temat 5

Temat 6

Temat 7

Temat 8

Temat 9

Temat 10

Temat 11

 **Wielomiany, część 3**

 Testy jednostkowe

 cmocka_example

WPI.INFO.I.16/17

WPI.LAB.INFO.I.16/17

PO.INFO.I.16/17

MD.INFO.I.16/17

ADMINISTRACJA



Administracja kursem

Jesteś zalogowany(a) jako Piotr Szuberski (Wyloguj)

IPP.INFO.I.16/17

Moodle, wersja 3.2.2+ | moodle@mimuw.edu.pl