Podstawy programowania

Laboratorium 8.

Cel:

- Opanowanie podstaw tworzenia i wykorzystania funkcji w C

[Uwaga: w trakcie ćwiczeń następuje tworzenie kodów o pewnej funkcjonalności, zawierających także testy pewnych formalnych aspektów programowania w języku C. Jako pomoc można wykorzystać wzorce w postaci plików kodu źródłowego znajdujące się na stronie przedmiotu, można także tworzyć od nowa własne kody, z własnymi komentarzami (zaczynając np. od oczyszczonej z komentarzy wersji kodu ze strony przedmiotu). Zaletą użycia dostarczonych kodów jest możliwość wykorzystania komentarzy w kodzie, które często zawierają kolejne numery odpowiadające punktom w scenariuszu (ważna jest kolejność działań, numery w kodzie niekoniecznie odpowiadają numerom punktów w scenariuszu). Możliwy sposób realizacji laboratorium polega na realizacji kolejnych działań dla kolejnych numerów w kodzie źródłowym, np. korzystając z wyszukiwania (po realizacji zadania pod numerem "2." można wyszukać w kodzie napis "3.", który będzie odpowiadał kolejnemu działaniu, a także kolejnym punktom scenariusza). Można także tworzyć własny kod, mając otwarte w dwóch oknach dwie wersje: dostarczoną z wypunktowanymi komentarzami (pomagającymi w nawigacji) i własną (zapewne napisaną własnym stylem z własnymi komentarzami).

Ostatecznie, w dowolny sposób, należy utworzyć kod źródłowy realizujący kolejne punkty poniższego scenariusza.]

Zajęcia:

1. Utworzenie katalogu roboczego *lab_8*
2. Skopiowanie ze strony przedmiotu pliku *prosta_funkcja.c* do nowego podkatalogu np. *simple*
3. Analiza kodu, kompilacja (uwaga na dołączenie biblioteki matematycznej *-lm*), uruchomienie
4. Modyfikacja lub stworzenie własnego kodu (należy wykonywać kolejne kroki, można wspierać się numeracją w kodzie pobranym ze strony w pliku *prosta_funkcja.c*):
    - zdefiniowanie zmiennej *test* w funkcji *main* i zainicjowanie wartością 5
    - wypisanie wartości zmiennej *test* w funkcji *main* przed wywołaniem *prosta_funkcja* i zaraz po nim
    - zdefiniowanie zmiennej *test* w funkcji *prosta_funkcja* i zainicjowanie wartością 10
    - wypisanie wartości zmiennej *test* wewnątrz funkcji *prosta_funkcja*
    - modyfikacja definicji funkcji *prosta_funkcja*, tak żeby przyjmowała argument (formalny) typu int
    - przesłanie wartości zmiennej *test* jako argumentu (aktualnego) wywołania funkcji *prosta_funkcja*
    - kompilacja – obserwacja komunikatów kompilatora
    - modyfikacja deklaracji funkcji *prosta_funkcja*
    - kompilacja, uruchomienie
    - zmiana wartości przesłanego argumentu wewnątrz funkcji *prosta_funkcja*
    - wypisanie wartości argumentu przed i po dokonaniu zmiany – obserwacja wartości argumentu (formalnego) wewnątrz funkcji *prosta_funkcja* i zmiennej *test* będącej argumentem aktualnym wywołania funkcji *prosta_funkcja* w funkcji *main*
    - (zadanie dodatkowe, nieobowiązkowe ale wartościowe: stworzenie wewnątrz funkcji *prosta_funkcja* bloków instrukcji i testowanie przesłaniania zmiennych o tych samych nazwach, poprzez inicjowanie innymi wartościami i wydruki wartości zmiennych - Wykład 7, slajd 12)

5. Skopiowanie za strony przedmiotu pliku *test_pierwiastka.c* do nowego podkatalogu np. *pierwiastek*
6. Analiza programu obliczania pierwiastka, uruchomienie
7. Modyfikacja lub stworzenie własnego programu (jak zwykle w kodzie ponumerowane kolejne kroki w komentarzach):
   - przeniesienie kodu obliczania pierwiastka do osobnej funkcji
     - utworzenie nagłówka, np. *double pierwiastek(double arg)*
     - umieszczenie prototypu funkcji jako deklaracji na początku pliku
     - umieszczenie definicji funkcji na końcu pliku
       - dokonanie zmian w nazwach zmiennych w kodzie funkcji
         - dopasowanie do nazwy argumentu (formalnego)
         - zdefiniowanie zmiennych lokalnych, ewentualnie z innymi nazwami
     - w dotychczasowym miejscu kodu obliczania pierwiastka w funkcji *main* umieszczenie poprawnego wywołania funkcji
   - uzupełnienie funkcji *main* o kilka innych wywołań funkcji *pierwiastek* (m.in. jako argumentu *printf*, jako dwóch argumentów dodawania, jako argumentu zewnętrznego wywołania (tzn. *pierwiastek(pierwiastek(liczba_double)); // jaki będzie wynik dla liczba_double=16?*)

---------- 3.0 ----------

8. Skopiowanie za strony przedmiotu pliku *test_potegowania.c* do nowego podkatalogu np. *wielomiany*
9. Analiza programu obliczania potęgi całkowitej liczby całkowitej, uruchomienie
10. Dodanie do programu funkcji *oblicz_wielomian_int* przyjmującej jako argument tablicę współczynników wielomianu (np. pięcioelementową – co oznacza wielomiany stopnia maksymalnie równego 4) o prototypie:
    1. *int oblicz_wielomian_int (int tab_wsp[], int argument);*
    2. napisanie definicji funkcji *oblicz_wielomian_int* wykorzystującej wywołania funkcji *potega_int*
11. Zdefiniowanie i zainicjowanie wartościami całkowitymi tablicy współczynników *tab_wsp* w funkcji *main*, poprawne wywołanie funkcji *oblicz_wielomian_int* wewnątrz funkcji *main*
12. Uruchomienie i przetestowanie poprawności działania programu
    - przykład testu: dla argumentu 1 wartość wielomianu powinna być sumą wyrazów tablicy *tab_wsp*

    [Uwaga: przedstawiony powyżej sposób obliczania wielomianu jest niewłaściwy z punktu widzenia wydajności obliczeń (liczby operacji potrzebnych do uzyskania wyniku). Powinno stosować się wzór Hornera, *obowiązujący dla wielomianów dowolnego stopnia, który w przypadku wielomianów stopnia 4 wykorzystuje równość:*
    $a[0] + a[1]*x + a[2]*x^2 + a[3]*x^3 + a[4]*x^4 = a[0] + x*(a[1] + x*(a[2] + x*(a[3] + x*a[4])))$
    (ilu operacji wymaga implementacja funkcji oblicz_wielomian_int *korzystająca z funkcji* potega_int *, a ilu korzystająca ze schematu Hornera? - spróbuj wyrazić te liczby jako funkcje stopnia wielomianu)]*

13. Implementacja funkcji *oblicz_wielomian_int_Horner* stosującej schemat Hornera
    - uruchomienie, sprawdzenie poprawności działania (np. przez porównanie z wynikami funkcji *blicz_wielomian_int* - można stworzyć pętlę wywoływania obliczeń wartości wielomianów obiema funkcjami, dla tego samego wielomianu i różnych argumentów)

---------- 4.0 ----------

14. Modyfikacja programu, tak aby funkcja *oblicz_wielomian_int_Horner* przyjmowała jako argument tablicę dowolnego rozmiaru i stopień wielomianu (stopień powinien być o co

najmniej 1 mniejszy od rozmiaru tablicy - podany powyżej wzór należy zamienić na obliczenia w odpowiedniej pętli)

15. Modyfikacja programu o sprawdzenie, czy obliczane wartości nie przekraczają liczby INT_MAX
16. Obliczenie wartości wielomianu milion razy funkcją *oblicz_wielomian_int* i milion razy funkcją *oblicz_wielomian_int_Horner* – porównanie czasu działania w jednym i drugim przypadku (np. za pomocą komendy *time: $ time test_potegowania)*

---------- 4.5 ----------

17. Dalsze modyfikacje funkcji *pierwiastek* :
    - zamiast stałego parametru ACCURACY użycie w treści obu funkcji: *main i pierwiastek* zmiennej globalnej
      - np. *double dokladnosc = ACCURACY*
    - ponieważ używanie zmiennych globalnych jest częstym źródłem błędów:
      - przeniesienie zmiennej *dokładnosc* do wnętrza funkcji *main*
      - kompilacja – obserwacja informacji zwracanych przez kompilator
      - przesłanie zmiennej *dokladnosc* jako argumentu do funkcji *pierwiastek*
        - modyfikacja nagłówka – w definicji i prototypie (deklaracji)
        - można użyć tej samej nazwy na argument aktualny i formalny – nie zwiększa to czytelności kodu, ale jest częstą praktyką
      - uruchomienie, przetestowanie
    - dla zwiększenia niezawodności obliczeń, przyjęcie wewnątrz funkcji *pierwiastek*, wartości dziesięciokrotnie mniejszej niż przesłany argument: *dokladnosc *= 0.1;*
      - przetestowanie, obserwacja błędu obliczeń (wydruk w funkcji *main*)
      - sprawdzenie, czy zmiana wartości zmiennej *dokladnosc* wewnątrz funkcji *pierwiastek* ma jakikolwiek wpływ na wartość zmiennej *dokladnosc* w funkcji *main*
      - narysowanie szkicu schematu pamięci w trakcie wykonania funkcji *pierwiastek*, na którym widoczne będą dwa obszary pamięci oznaczone nazwą zmiennej *dokladnosc*:
        - jeden widoczny tylko w funkcji *main*
        - i drugi widoczny tylko w funkcji *pierwiastek*
    - modyfikacja polegająca na obsłudze błędnego argumentu mniejszego od zera
      - wprowadzenie do funkcji *main* sprawdzenia, czy funkcja *pierwiastek* zwraca kod błędu
    - wykorzystanie wydruków kontrolnych zapisanych w komentarzach na końcu pliku *test_pierwiastka.c* do obserwacji zbieżności wyniku
      - który błąd nadaje się lepiej do śledzenia dokładności obliczeń i przerwania obliczeń: bezwzględny czy względny?
      - porównaj zbieżność wyników dla bardzo dużych (np. 10^7) i bardzo małych liczb (np. 10^{-7})
      - czy da się powiązać błędy z liczbą cyfr znaczących wyniku?
    - usunięcie zadanego argumentu *dokladnosc*
      - zamiast tego przeprowadzanie obliczeń do uzyskania względnego błędu kolejnych iteracji mniejszego od 10^{-12}
      - jaką daje to ostateczną dokładność względną wyniku? (wydruk w funkcji *main*)

---------- 5.0 ----------

Warunki zaliczenia:

1. Obecność na zajęciach i wykonanie co najmniej kroków 1-7
2. Oddanie sprawozdania o treści i formie zgodnej z regulaminem ćwiczeń laboratoryjnych, zawierającego m.in.: opis wykonanych zadań, kod źródłowy podstawowych funkcji i konstrukcji sterujących, zrzuty ekranu dla przypadków testowania kodu, wnioski