Sprawozdanie z przebiegu projektu z Architektury Komputerów 2		
Prowadzący	dr inż. Dominik Żelazny	
Temat projektu	Arytmetyka modularna	
Termin zajęć	Wtorek, 13:35-15:05 TN	
Skład grupy projektowej	Daniel Leśniewicz	250996
	Patryk Fidrych	248828

Poniżej zostały opisane poszczególne etapy projektu wraz z krótkim opisem dodanych funkcjonalności.

Pierwszy raport z postępu prac

Tak, jak zostało wcześniej ustalone, nasz projekt został napisany w języku C++. Poszczególne funkcje implementujące algorytmy zostały napisane w języku asembler i wstawione do programu.

Program, który do tej pory udało nam się stworzyć składa się między innymi z pliku *main.cpp*, w którym znajduje się wywołanie funkcji wyświetlania menu głównego programu oraz logika odpowiedzialna za wybieranie określonego algorytmu. Zostało to zaimplementowano z używając switch`a.

Pierwszą implementacją w programie jest wyznaczanie NWD za pomocą algorytmu Euklidesa. Aby zobrazować działanie algorytmu, załóżmy że należy wyznaczyć NWD z liczb *a* oraz *b*. Na początku wykonywane jest dzielenie z resztą liczby *a* przez liczbę *b*. Jest to realizowane za pomocą instrukcji *div* w języku assembler. Gdy reszta z dzielenia, która jest umieszczona w rejestrze %edx wynosi 0 to największym wspólnym dzielnikiem jest liczba *b*. W przypadku gdy reszta jest różna od zera to następuje przypisanie liczbie *a* wartości liczby *b*. Następnie liczbie *b* jest przypisywana wartość reszty. Ponownie jest realizowane dzielenie liczby *a* przez *b*, aż reszta nie będzie równa zero. Cała operacja jest realizowana w pętli while, poprzez użycie instrukcji *cmp*. Implementacja algorytmu znajduje się w pliku *Euklides.cpp*. Dodatkowo znajduje się tam również funkcja wyświetlająca menu oraz pobierająca dane od użytkownika.

Kolejnym algorytmem, który został zaimplementowany jest naiwny algorytm wyznaczania pierwszości liczby. Algorytm polega na próbnym dzieleniu sprawdzanej liczby *a* przez liczby z zakresu od *2* do *sqrt(a)*. Przy każdej takiej operacji badana jest reszta z dzielenia. Jeśli reszta podczas którejś operacji dzielenia będzie wynosiła 0, to liczba *a* nie będzie liczbą pierwszą. Interesujące może się wydawać, dlaczego wystarczy sprawdzić liczby z podanego wyżej zakresu. Dzieje się tak, ponieważ jeśli liczba posiada czynnik większy od *sqrt(a)*, to drugi jego czynnik musi

być mniejszy od pierwiastka z *a*, aby ich iloczyn musiał być równy *a*. Zatem wystarczy podzielić liczbę *a* przez liczby z danego przedziału, aby wykluczyć liczby złożone. Tak jak zostało wcześniej założone, algorytm został napisany w języku asembler. Z powodu braku dobrej znajomości tego języka, pierwiastkowanie liczby *a* zostało wykonane w języku *C++.* Wynik tej operacji został jednak wykorzystany bezpośrednio we wstawionym fragmencie z kodem algorytmu. Algorytm zwraca podmienioną liczbę *a*, w zależności od tego czy jest pierwsza, czy nie. Jeśli liczba *a* była pierwsza, to zostanie zwrócone *1*, a jeśli nie była liczbą pierwszą to zwróci *0*. Zostało to wykorzystane w wywołaniu algorytmu. W pliku *NaivePrime.cpp*, w którym znajduje się implementacja algorytmu, znajduje się również funkcja odpowiedzialna za wczytywanie liczby podanej obserwacji. Wykorzystując zaimplementowany algorytm w funkcji *checkPrime()* pokazuje ona, czy dana liczba jest liczbą pierwszą wyświetlając stosowną informację.

Drugi raport z postępu prac

Podczas drugiego etapu dodane zostały nowe algorytmy napisane w języku asembler (Chiński Test Pierwszości, Małe Twierdzenie Fermata).

Pierwszym algorytmem, który został zaimplementowany jest Chiński test pierwszości. Główna zasada działania algorytmu jest następująca: jeśli liczba *a*, jest liczbą pierwszą to wyrażenie $2^a - 2$ jest podzielne przez liczbę *a*. Na tej podstawie można wyciągnąć wniosek: jeśli $2^a mod\ a\ \neq 2$ to liczba a nie jest liczbą pierwszą. Zważywszy na fakt, że reszta z dzielenia 2^a przez *a*, jest z przedziału <0; a-1>, w algorytmie można w prosty sposób obliczyć wyrażenie $2^a mod\ a$. Kolejne potęgi są rozbijane na operacje mnożenia modulo a. Dzięki temu istnieje możliwość wyznaczania dużych potęg 2^a poprzez kolejne wymnażanie reszt. W programie do realizacji algorytmu, zostały stworzone dwie metody: *multiplicationModulo(int num1, int num2, int modNum)* oraz *powerModulo(int num1, int numMod)*. Pierwsza z nich zwraca wynik następującej operacji: $(num1 \cdot num2)\ mod\ modNum$. W swoim działaniu wykorzystuje dodawanie wielokrotności mnożnej modulo *modNum*. Druga z funkcji zwraca wartość wyrażenia $2^{num1}\ mod\ modNum$ i w swoim działaniu wykorzystuje funkcję *multiplicationModulo*. Dla ułatwienia implementacji algorytmu oraz możliwości jakie daje język asembler w zakresie możliwości przesuwania bitów zastosowano w jej działaniu maskę bitową. Maska bitowa znajduje zastosowanie, gdyż w końcowym mnożeniu biorą udział tylko te potęgi liczby 2, które odpowiadają wagom bitów ustawionych na 1 w binarnej reprezentacji wykładnika, czyli parametru num1 przekazanego do funkcji *powerModulo*. Gdy funkcja skończy zwraca wynik, który jest porównywany z liczbą 2 i na tej podstawie jest ustalana pierwszość liczby.

Drugim algorytmem, który został zaimplementowany jest algorytm bazujący na Małym Twierdzeniu Fermata. Na początku sprawdzana jest podzielność wprowadzonej liczby przez liczby pierwsze z przedziału <2;1000>. Umożliwia to wstępną eliminację liczb złożonych oraz liczb pseudopierwszych Carmichaela. Jeśli liczba będzie złożona, to zwracana jest od razu informacja, że nie spełnia ona zadanego warunku. Następnie w pętli sprawdzany jest warunek Fermata. Zostaje wylosowana podstawa *a*, która zawiera się w przedziale od 2 do testowanej liczby

pomniejszonej o jeden. Następnie sprawdzane jest czy *a* jest względnie pierwsza z wprowadzoną liczbą *num*. Polega to inaczej mówiąc na sprawdzeniu, czy *NWD (num, a) = 1*. Jeśli tak, to testowany jest warunek: $a^{num - 1} \ mod \ num = 1$. Jeśli zostanie spełniony ten warunek, to liczba może być liczbą pierwszą. Jednak nie jest to na tym etapie pewne. Aby to potwierdzić należy sprawdzić to wykonując test Fermata np. dziesięciokrotnie. Liczba, która została uznana liczbą pierwsza może być również liczbą pseudopierwsza Carmichaela. Liczby te są jednak bardzo odległe, a stosując podzielność przez liczby pierwsze z przedziału <2;1000> można być pewnym, że algorytm daje poprawne wyniki.

Trzeci raport z postępu prac

Podczas ostatniego etapu został zaimplementowany test Millera-Rabina, który sprawdza pierwszość liczby nieparzystej. Funkcje wywoływane w algorytmie są napisane w języku asembler. Ich wywołania zostały tym razem napisane w języku C. Algorytm ten sprawdza, czy liczba jest liczbą pierwszą. Podana liczba do sprawdzenia, ze względu na zaimplementowany algorytm, powinna być nieparzysta. Dodatkowym argumentem, który jest podawany jest ilość powtórzeń testu Millera-Rabina. Nim więcej powtórzeń, tym większa szansa, że test zwróci poprawny wynik. Algorytm zwraca informację o pierwszości liczby (lub silnej pseudo-pierwszości) z prawdopodobieństwem $¼^n$. Stała n oznacza ilość powtórzeń testu.

Dodatkowo, w trzecim - ostatnim etapie naszym celem było opracowanie sprawozdania z przebiegu projektu. Zostały w nim opisane poszczególne etapy. Oprócz przedstawienia przebiegu projektu sporządzony został opis projektu, który znajduje się w osobnym pliku. Zawiera on informacje o zaimplementowanych algorytmach.