Opis 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

Zaimplementowane algorytmy

W naszym projekcie udało się zaimplementować następujące algorytmy:

- Algorytm Euklidesa
- Naiwny Algorytm Sprawdzania Pierwszości Liczb
- Chiński Test Pierwszości
- Male Twierdzenie Fermata
- Test Millera-Rabina

Algorytm Euklidesa służy do wyznaczania NWD liczb. Pozostałe z wymienionych algorytmów zajmują się badaniem pierwszości liczb.

Zostały one opisane poniżej.

Plik main.cpp

Pliku main.cpp znajduje się wywołanie funkcji wyświetlania menu głównego programu oraz logika odpowiedzialna za wybieranie określonego algorytmu. Zostało to zaimplementowano z używając switch`a. W zależności od wybranej opcji, zostaje wywołany właściwy algorytm.

Fragment pliku *main.cpp*:

```
1.          switch(option){
2.          case 1:
3.              cout << "\n------ALGORYTM EUKLIDESA------" << endl;
4.              Euklides::enterValues();
5.              break;
6.          case 2:
7.              cout << "\n------NAIWNY ALGORYTM SPRAWDZANIA PIERWSZOSCI LICZB------" << endl;
8.              NaivePrime::enterValue();
9.              break;
10.         case 3:
11.             cout << "\n---------CHINSKI TEST PIERWSZOSCI---------" << endl;
12.             ChineseTestPrime::enterValue();
13.             break;
14.         case 4:
15.             cout << "\n---------MALE TWIERDZENIE FERMATA---------" << endl;
```

```
            LittleFermat::enterValue();
            break;
        case 5:
            cout << "\n---------TEST MILLERA-RABINA---------"<< endl;
            MillerRabin::enterValue();
            break;
        case 0:
            exit(0);
            break;
        default :
            cout << "Bledny numer wybranego algorytmu!"<< endl;
            break;
        }
        cout << "\n\nNaciśnij dowolny przycisk, aby przejść dalej!" << endl;
        getch();
        system("CLS");

    }while (option != 0);
```

Plik Euklides.cpp

W tym pliku został zaimplementowany kod algorytmu Euklidesa. Sprawdza on jaki jest największy wspólny dzielnik dwóch liczb. …

Kod funkcji:

```
int Euklides::NWD(int a, int b){
        asm("\
        # 8(%ebp) - zmienna a                                          \n\
        # 12(%ebp) - zmienna b                                         \n\
        # %ecx - rejestr pomocniczy                                    \n\
                                                                       \n\
        start_loop:                                                    \n\
            cmpl $0, 12(%ebp)         # jesli b=0 to koniec            \n\
            je end_loop                                                \n\
            movl 12(%ebp), %ecx       # przechowanie wartosci b        \n\
                                                                       \n\
            # OPERACJA MODULO                                          \n\
            # Wykonanie dzielenia: podwojne slowo w %edx:eax           \n\
            # przez argument instr. div                                \n\
            # W tym przypadku a/b. Resza dzielenia w %edx              \n\
            movl $0, %edx                                              \n\
            movl 8(%ebp), %eax                                         \n\
            movl 12(%ebp), %ebx                                        \n\
            divl %ebx                                                  \n\
                                                                       \n\
            # Przypisanie reszty z dzielenia do b                      \n\
            movl %edx, 12(%ebp)                                        \n\
            # Stara wartosc b jest nowa dzielna                        \n\
            movl %ecx, 8(%ebp)                                         \n\
            jmp start_loop                                             \n\
        end_loop:                                                      \n\
    ");

    return a;
```

Plik NaivePrime.cpp

Kolejnym algorytmem, który został zaimplementowany jest naiwny algorytm sprawdzania pierwszości liczb. Funkcja przedstawiona poniżej zwraca *1* jeśli dana liczba jest liczbą pierwszą lub *0* jeśli nie jest liczbą pierwszą.

Kod funkcji:

```
1.  int NaivePrime::checkPrime(int a){
2.
3.      int root=sqrt(a);
4.
5.      asm("\
6.          # 8(%ebp) - zmienna a                                          \n\
7.          # -12(%ebp) - sqrt(a)                                          \n\
8.                                                                         \n\
9.          movl 8(%ebp), %ecx       # zapis do %ecx sprawdzanej liczby    \n\
10.         movl $2, %ebx            # licznik pierwszosci do symulowania zakresu   \n\
11.                                  # a do sqrt(a)                        \n\
12.                                                                        \n\
13.         cmpl $1, %ecx            # jesli liczba jest 1, to nie jest pierwsza    \n\
14.         je niePierwsza                                                 \n\
15.                                                                        \n\
16.         start_loop:                                                    \n\
17.         cmpl %ebx, -12(%ebp)     # jesli róznica: sqrt(a)-licznik pierw. jest   \n\
18.         jl loop_exit             # mniejsze od 0 to koniec petli       \n\
19.         movl %ecx, %eax          # zapis do %eax sprawdzanej liczby    \n\
20.                                                                        \n\
21.         movl $0, %edx            # dzielenie %edx:%eax przez arg instr. divl    \n\
22.         divl %ebx                # czyli wartosc licznika pierwszosci  \n\
23.                                                                        \n\
24.                                  # reszta z dzielenia jest w %edx      \n\
25.         cmpl $0, %edx            # jesli reszta rowna 0 to liczba      \n\
26.         je niePierwsza           # nie jest pierwsza - koniec petli    \n\
27.                                                                        \n\
28.                                                                        \n\
29.         incl %ebx                #zwieksz licznik                     \n\
30.         jmp start_loop                                                 \n\
31.                                                                        \n\
32.         loop_exit:                                                     \n\
33.         movl $1, 8(%ebp)         # jesli jest pierwsza to a=1          \n\
34.         jmp koniec                                                     \n\
35.                                                                        \n\
36.         niePierwsza:             # jesli nie jest pierwsza to a=0      \n\
37.         movl $0, 8(%ebp)                                               \n\
38.                                                                        \n\
39.         koniec:                                                        \n\
40.     ");
41.
42.     return a;
43. }
```

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. Wystarczy sprawdzić liczby jedynie z zakresu *2* do *sqrt(a)*. 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.

Plik ChineseTestPrime.cpp

…

Plik Little Fermat.cpp

Pierre de Fermat jest autorem Małego Twierdzenia Fermata, które bazuje na chińskim twierdzeniu o pierwszości. Twierdzenie Fermata zostało ono odkryte w 1640 roku. Można je przedstawić następująco:

Dla liczby pierwszej num i dowolnej liczby naturalnej a, jeśli *NWD (num, a) = 1, to* $a^{num\ -1}$ *mod num = 1.* Twierdzenie to jest bardziej ogólne od twierdzenia chińskiego.

W zaimplementowanym algorytmie 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.

Aby sprawdzić, czy dana liczba jest pierwsza należy wywołać *controller*, zarządza on powyższą logiką algorytmu. Wywołuje on funkcję *multiplicationModuloF()* oraz *powerModuloF()*. Dodatkowo w algorytmie został wykorzystany kod algorytmu Euklidesa, aby wyznaczyć NWD liczb.

Ze względu na obszerność kodu, algorytm nie został on wklejony w tym dokumencie.

Plik MillerRabin.cpp

……