SPRAWOZDANIE		Data wykonania: 14.01.2019
Tytuł zadania:	**Wykonał:**	**Sprawdził:**
Maszyna Turinga	*Kamil Sztandur 307 354*	*dr inż. Konrad Markowski*

Spis treści

Cel projekt

Celem zadania było zaprogramowanie emulatora Maszyny Turinga liczącego różnicę właściwą dwóch liczb naturalnych podanych przez użytkownika.

Zadanie 16. *Napisać emulator maszyny Turinga obliczającą różnicę właściwą:*

$$m \dot{-} n = \begin{cases} m-n & dla \quad m \geq n \\ 0 & dla \quad m < n \end{cases}$$

dla parametrów zakodowanych unarnie.

Postać MT

$$M=(\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{0,1\}, \{0,1,B\}, \delta, q_0, B, 0)$$

dla

δ	0	1	B
q_0	(q_1,B,**P**)	(q_5,B,**P**)	-
q_1	(q_1,0,**P**)	(q_2,1,**P**)	-
q_2	(q_3,1,**L**)	(q_2,1,**P**)	(q_4,B,**L**)
q_3	(q_3,0,**L**)	(q_3,1,**L**)	(q_0,B,**P**)
q_4	(q_4,0,**L**)	(q_4,B,**L**)	(q_6,0,**P**)
q_5	(q_5,B,**P**)	(q_5,B,**P**)	(q_6,B,**P**)
q_6	-	-	-

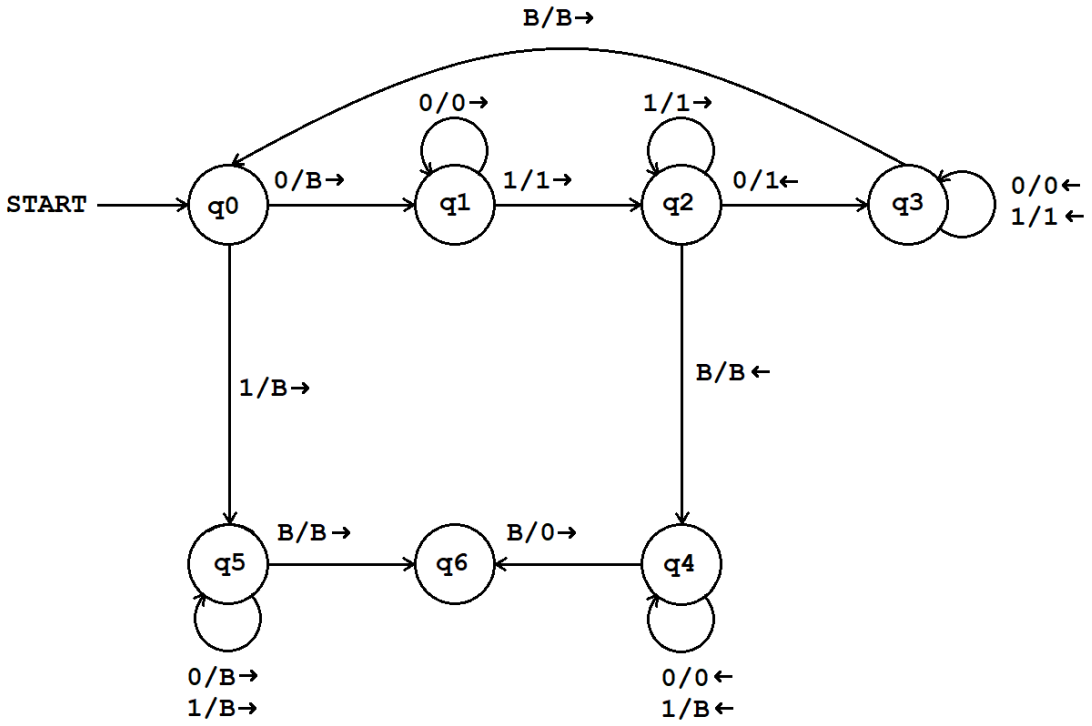
Program powinien:

- *Wyświetlić opis MT.*
- *Dla wczytanych dwóch liczb całkowitych generować taśmę wejściową zakodowaną unarnie.*
- *Wyświetlać ciąg opisów chwilowych MT dla zadanej taśmy wejściowej,*
- *Po zatrzymaniu automatu zinterpretować otrzymany wynik.*

Teoria

Opracowany przez Alana Turinga model matematyczny został prototypem dzisiejszych komputerów. Ze względu na abstrakcyjność swojego działania bezpieczne zakodowanie MT w celu uzyskania konkretnej instrukcji jest znacznie bardziej skomplikowane, co potwierdza geniusz jej autora. Poprzez odpowiednią instrukcję dla ruchu głowicy MT i zawartość taśmy wejściowej musimy zarządzać znakami na taśmie, aby po dotarciu przez głowicę do stanu końcowego – tutaj q6 – stan taśmy zawierał odpowiedź na nasze zadanie.

Diagram przejść dla naszej MT prezentuje się następująco:



W związku ze złożonością problemu postanowiłem podzielić program na podmoduły, które uporządkują kod ze względu na funkcje oraz które pozwolą jednoczesną kulturalną pracę wielu potencjalnych programistów, którzy mogą w przyszłości nad nim pracować.

Te moduły to:

- **main.c** odpowiadający za koordynację pracy programu (wywoływanie odpowiednich modułów), jasną komunikację z użytkownikiem (schludny interfejs tekstowy) oraz ochroną przed niewłaściwymi działaniami użytkownika, mogące zaburzyć pracę programu lub „nieświadomie" wygenerować nieprawidłowe wyniki. Np. poprzez podanie niekompletnej pary liczb lub czegokolwiek innego niż liczby naturalne.
- **help.c** wypisujący instrukcję korzystania z programu dla użytkownika, gdy tego potrzebuje - sam poprosi o to lub nieprawidłowo skorzysta z programu.
- **inputGen.c** konwertujący podane przez użytkownika liczby dziesiętne na zapis unarny stanowiący taśmę, na której będzie pracować MT.
- **solver.c** stanowiący serce obliczeniowe programu, zawierający instrukcje potrzebne do emulujacji działania MT na podstawie przyjętej taśmy z modułu inputGen.c. Solver jest także odpowiedzialna za wypisywanie bieżących opisów chwilowych. Po zatrzymaniu się w stanie końcowym – q6 - wraca taśmę wyjściową, czekającą na interpretację.
- **interpreter.c** służący interpretacji taśmy wyjściowej, zamieniający jej zapis unarny na system dziesiątkowy zawierający jasną odpowiedź na pytanie użytkownika.

Kompatybilności wszystkich modułów „pilnuje" plik nagłówkowy **MT.h** zawierający prototypy funkcji znajdujących w programie. Program instalujemy poprzez plik Makefile, zawierający potrzebne instrukcje.

Dzięki takiej budowie korzystanie z programu jest bezpieczne oraz jasne nawet dla niedoświadczonych użytkowników, a intuicyjny i czysty podział programu na moduły z pilnującym kompatybilności plikiem nagłówkowym przyszli potencjalni programiści będą mogli łatwo i bezpiecznie pracować nad moim kodem.

Szczegóły implementacyjne

Program natychmiast po uruchomieniu musi być gotowy na interakcję z użytkownikiem i odpowiednio zareagować zależnie od wprowadzonych przez niego danych. Mój program sprawdza, czy użytkownik podał dwa potrzebne argumenty oraz czy są one liczbami całkowitymi dodatnimi. Jeżeli wszystko jest w porządku, to kontynuuje swoją pracę. W przeciwnym wypadku wyświetla komunikat o błędzie lub proponuje użytkownikowi wyświetlenie instrukcji help.

```
if( argc < 2 ) {
    printf("        Dodaj parametr -h / -help / -? aby wyswietlic Pomoc.\n");
    printf("\n");
}
printf("*------------------------------------------------------------------*\n\n");
if( argc < 2 ) return 0;

if( strcmp(argv[1], "-help") == 0 || strcmp(argv[1], "-h") == 0 || strcmp(argv[1], "-?") == 0 ) help();
else if( atoi( argv[1] ) == 0 || argv[2] == NULL || atoi( argv[2] ) <= 0 || atoi( argv[1] ) <= 0 ) {
        printf("Nieprawidlowe uzycie programu. Oba parametry powinny byc liczbami naturalnymi.\n" );
        printf("Dodaj parametr -h / -help / -? aby wyswietlic Pomoc.\n\n");
        return 1;
}
```

Po zaakceptowaniu danych wejściowych program zwraca się do funkcji generatora taśmy wejściowej i podaje jej podane przez użytkownika liczby. Generator na ich podstawie zwraca napis będący taśmą wejściową (zakodowaną unarnie), po której będzie poruszać się nasza MT.

```
int inputLength = m + n + 1;

char *input = malloc( sizeof(char) * inputLength );
for( int i = 0; i < inputLength; i++ ) {
    if( i == m ) input[i] = '1';
    else input[i] = '0';
}

return input;
```

Następnie program wypisuje tę taśmę na ekran i przekazuje ją modułowi **solver.c** odpowiadającemu za serce emulacji naszej MT. Solver zmienia taśmę wejściową, doklejając na jej początku pozycję głowicy (literka H) oraz symbole puste B na jej granicach dla bezpieczeństwa pracy programu. Tutaj rozpoczyna się proces iteracyjny aż do momentu uzyskania wyniku. Po każdym przesunięciu się MT program wyświetla jej opis chwilowy.

```
int iterationCount = 1; //Zmienna pomocnicza do wyswietlania opisow chwilowych
while( ON_OFF_switch ) {
    for( int j = 0; j < strlen( tape ); j++ ) {
        if( tape[j] == 'H' ) printf("[q%d]", currentState );
        else if( tape[j] == 'B' && ( j == 0 || j == ( strlen( tape ) -1 ) ) ) continue;
            else printf("%c", tape[j] );
    }
    if( iterationCount%7 == 0 ) printf("\n");
    if( currentState != 6 ) printf("|-> ");
```

Iteracja odbywa się dopóki funkcja ustalająca następny ruch nie zwróci informacji o dotarciu do stanu końcowego lub komunikatu błędu. Do tego czasu na początku każdej iteracji wypisywany jest opis chwilowy MT. Wypisuje on napis pełniący funkcję taśmy wejściowej, zamieniając **H** (oznaczające pozycję głowicy na taśmie) na obecny stan Q głowicy oraz pomijając obydwa symbole puste B na krańcach taśmy. Po wyświetleniu całego opisu chwilowego wypisuje symbol przejścia/przypisania oraz łamie linijkę co 7 opis chwilowy dla aspektów kosmetycznych.

```
char currentInput = tape[position + 1];

if( moveHead( currentInput ) == 0 ) break;

if( direction == 'P' ) {
    tape[position] = charToWrite;
    tape[position + 1] = 'H';
    position++;
} else {
    tape[position + 1] = charToWrite;
    tape[position] = tape[position - 1];
    tape[position - 1] = 'H';
    position--;
}
iterationCount++;
```

Po wypisaniu OC maszyna planuje wykonanie następnego ruchu. Pobiera znak znajdujący się na prawo od głowicy, a następnie podaje go funkcji moveHead() określającej następny ruch (którą opiszę później). Na podstawie zmodyfikowanych przez nią zmiennych globalnych MT wykonuje ruch w ustalonym kierunku, nadpisując znak po prawej stronie na ustalony przez wspomnianą funkcję.

Po skończonej liczbie takich iteracji i MT powinna dotrzeć do stanu końcowego i zwrócić prawidłowy wynik zakodowany unarnie na zmodyfikowanej taśmie wejściowej.

Stan głowicy opisują cztery zmienne globalne:

- **currentState** (nr. aktualnego stanu q)
- **charToWrite** (znak, który ma zostać zapisany na taśmie po tym ruchu)
- **direction** (L lub P)
- **position** (nr. pozycji głowicy na taśmie wejściowej)

Jak wspomniałem przed każdym ruchem MT czyta znak znajdujący się na prawo od głowicy i przekazuje go funkcji **moveHead**(), która na jego podstawie ustala wartości tych czterech zmiennych globalnych, aby MT prawidłowo wykonała ruch. **Oto przykładowa instrukcja dla stanu q3:**

```
case 3:
    if( input == '0' ) {
                        currentState = 3;
                        charToWrite = '0';
                        direction = 'L';
                        return 1;
                    } else if( input == '1' ) {
                                currentState = 3;
                                charToWrite = '1';
                                direction = 'L';
                                return 1;
                            } else if( input == 'B' ) {
                                    currentState = 0;
                                    charToWrite = 'B';
                                    direction = 'P';
                                    return 1;
                                    }
    break;
```

Funkcja ta zwraca 1, jeżeli MT jest gotowa do następnego ruchu lub 0, jeżeli nie potrafi znaleźć żadnej instrukcji bądź dotarła do stanu końcowego. Taki rezultat przerywa emulację. Jeżeli MT zatrzymała się w stanie końcowym (q6), to program przekazuje taśmę do modułu głównego, który następnie przekazuje ją interpreterowi, tłumaczącemu ten wynik unarny na liczbę decymalną. Jeżeli nie, to program wypisuje na ekran komunikat o błędzie i zapewnia użytkownika, że poniższy wynik powstał z powodu błędu i jest definitywnie nieprawidłowy.

```
if( currentState != 6 ) {
    printf("Maszyna Turinga zatrzymala sie na niedozwolonym stanie.\n");
    printf("Ponizszy wynik jest definitywnie nieprawidlowy.\n");
    return tape;
}
```

Następnie moduł główny przekazuje tę taśmę interpreterowi, którego zadaniem jest przetłumaczenie jej wyniku z unarnego na system decymalny. Po dokonaniu obliczeń interpreter zwraca modułowi głównemu wynik, który następnie jest przez niego wyświetlany. Praca programu dobiega końca.

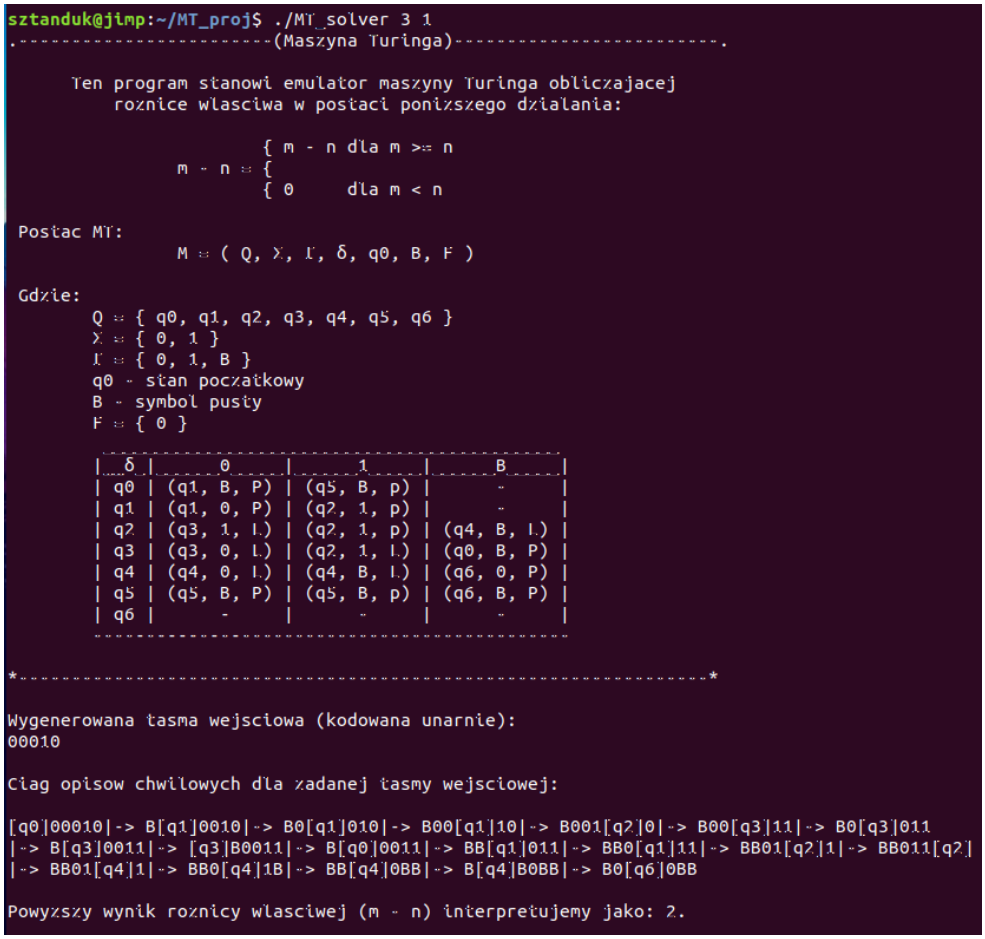
```
int interpreter( char *output ){
    int result = 0;
    for( int i = 0; i < strlen( output ); i++)
        if( output[i] == '0' ) result ++;

    return result;
}
```

Sposób wywołania programu

Program można wywołać na cztery sposoby – czysty, błędny lub poprawny. Oddzielone spacją argumenty użytkownik podaje w linii poleceń wraz z wywołaniem programu:

- Wywołanie poprawne np. **'./MT_solver 3 1'** wykona różnicę właściwą (3 – 1).



- Wywołanie czyste (bez żadnych argumentów) wyświetla powyższy opis programu wraz z postacią symulowanej MT oraz **dodatkowo proponuje wyświetlenie instrukcji help** poprzez dodanie parametru **-h**, **-help** lub **-?** jako parametr przy wywoływaniu programu.
- Wywołanie niepoprawne poprzez np. podanie niepełnej pary argumentów albo czegoś innego niż parę liczb całkowitych skutkuje wyświetleniem informacji o nieprawidłowym skorzystaniu z programu, odesłaniem do wspomnianej instrukcji help i zakończeniu pracy programu.

Program instalujemy poprzez plik **Makefile** i komendę 'make' w linii poleceń. Struktura tego pliku wygląda następująco:

```
CC=gcc
NAME=-o MT_solver
CFLAGS=-Wall

all: main
main: main.o help.o inputGen.o solver.o interpreter.o
	$(CC) $(CFLAGS) $(NAME) $(^)
main.o: main.c MT.h
	$(CC) $(CFLAGS) -c main.c
help.o: help.c
	$(CC) $(CFLAGS) -c help.c
inputGen.o: inputGen.c
	$(CC) $(CFLAGS) -c inputGen.c
solver.o: solver.c
	$(CC) $(CFLAGS) -c solver.c
interpreter.o: interpreter.c
	$(CC) $(CFLAGS) -c interpreter.c
clean:
	rm inputGen.o
	rm main.o
	rm help.o
	rm interpreter.o
	rm solver.o
```

Wnioski i spostrzeżenia

Proszę przedstawić wnioski z realizowanego programu. Należy skupić się na tym co było problemem oraz z czego jesteście Państwo szczególnie zadowoleni, czym chcielibyście się pochwalić.

Rzeczą, która sprawiła mi największy problem w czasie pracy na programem, była sama Maszyna Turinga, a mianowicie prawidłowa implementacja jej działania. Odwzorowanie Maszyny Turinga w C jest zbliżone do wcześniejszego z moich projektów, automatu skończonego, lecz tutaj problem jest znacznie bardziej złożony. Dla naszej MT musiałem użyć znacznie więcej zmiennych determinujących następny ruch MT oraz jej aktualny stan, a więc wymagało to ode mnie napisania bardziej złożonych instrukcji, biorących pod uwagą aktualną wartość każdej z tych zmiennych. Moje pierwsze rozwiązanie zawierało znacznie więcej zmiennych oraz niepotrzebnych instrukcji czy powtórzeń. Drugie tyle czasu spędziłem na „odchudzaniu" kodu co nad pisaniem pierwowzoru aż do momentu, gdy uznałem mój kod za zarówno optymalny jak i wciąż czytelny pod względem logiki oraz terminologii dla programistów.

Ciężko mi wyróżnić ten jeden szczegół, z którego jestem najbardziej zadowolony po ukończeniu projektu tego projektu, ponieważ każdy taki szczegół wynika z mojej ciężkiej pracy. Jestem szczególnie dumny z tego, jak w czasie tego semestru udało mi się podnieść moje umiejętności programowania w języku C do tego stopnia, aby móc zwinnie programować bardziej zaawansowane programy tego rodzaju, swobodnie przelewając swoje myśli i dziesiątki pomysłów na kod, a potem świadomie i skutecznie go zoptymalizować tak, aby wciąż pozostał czytelny i przyjazny. W efekcie praca nad tym projektem była dla mnie szczerą przyjemnością i zaangażowała mnie jak dobra książka, a każdą przeszkodę potraktowałem jako wyzwanie oraz szansę na rozwinięcie się. Podsumowując, mogę śmiało stwierdzić, że jestem dumny z całokształtu tego projektu, mojej postawy w czasie pracy nad nim oraz tego jaką drogę przeszedłem w czasie tego semestru.