Kalkulator RPN

Kacper Wiącek

22 Styczeń 2021

Spis treści

Listings

1 Program i jego działanie

Program kalkulator RPN działa w oparciu o odwrotną notację polską. Jest to kalkulator podstawowy, który potrafi wykonywać operacje matematyczne: dodawanie, odejmowanie, dzielenie, mnożenie. Dane są wczytywane ze standardowego wejścia, są to tylko liczby całkowite.Wprowadzane liczby zmiennoprzecinkowe są błędne i nie są dodawane do stosu liczb. Stos oparty jest na działaniu dynamicznej listy jednokierunkowej. Każda dodana liczba ląduje na szczyt stosu. Operacje matematyczne można wykonać jeżeli wymagane dwie liczby znajdują się na stosie.W przeciwnym wypadku program zwraca błąd, wyrzuca komunikat ostrzegawczy.Ponadto poza wykonywaniem operacji matematycznych, program operuje na wspomnianym wcześnie stosie. DOdawane są do niego liczby całkowite funkcją push, bądź usuwane ze szczytu funckją pop('P').Dodatkowo mamy możliwość zduplikowania liczby ze szczytu('d'), zamiany jej pozycji w liście z poprzednim elementem('r'), opcję wyczyszczenia zawartości stosu('c'),wydrukowanie jej elementów(funckja 'f') oraz wyświetlenie tylko liczby ze szczytu stosu ('p'). Program kończy działanie po wczytaniu znaku 'q'.

2 Kompilacja

Program składa sie z 3 plików: "main.c", funckja główna programu, "kalkulator rpn.c", który zawiera wszystkie deklaracje funckji,oraz "kalkulator rpn.h" z nagłówkami wszystkich funkcji oraz strukturą stosu.Kompilacja plików main.c oraz kalkulator rpn.c zachodzi następująco: gcc -c main.c oraz gcc -c kalkulator rpn.c.Uzyskujemy dwa pliki wykonawcze: kalkulator rpn.o oraz main.o.Ostatni etap ich kompilacji zachodzi w ten o to sposób: gcc -std=c99 -Wall -pedantic main.o kalkulator rpn.o -o prog. Na samym końcu wywoływujemy program i możemy zacząć działać -> ./prog.Oczywiście jest to męczące i czasochłonne więc dobrym rozwiązaniem jest posłużyenie się plikim makefile, który wykona za nas całą kompilacje.Wystarczy ,że pliki wcześniej wspomniane oraz plik Makefile znajdują się obok siebie(w jednym folderu) i wywołać komendę "make" w terminalu.PLiki zostaną automatycznie skompilowane.Musimy tylko wywołać program -> ./prog.DOdatkowo plik Makefile ma opcje czyszczenia plików wykonawczych.(make clean).

```
#Makefile
#Wywo anie komend  "make"
CFLAGS = -std=c99 -pedantic -Wall  #Inne flagi kompilatora

main: main.o kalkulator_rpn.o
  gcc $(CFLAGS) main.o kalkulator_rpn.o -o prog

main.o: main.c kalkulator_rpn.h
  gcc -c main.c

kalkulator_rpn.o: kalkulator_rpn.c kalkulator_rpn.h
  gcc -c kalkulator_rpn.c
# KOmenda "make clean" czy ci pliki z rozszerzeniem .o oraz
    utworzony plik prog
clean:
  rm *.o prog
```

3 Szczególne rozwiązania określonych problemów w kodzie

3.1 Struktura stosu,nagłówki funckji

Struktura stosu jest typu t stos. W tej strukturze mamy odpowiednio zmienną int, która przechowuje liczbę, oraz zmienną typu t stos* następny, jest to wskaznik na następny element w stosie.Jak wcześniej wspomniano stos jest dynamiczną listą jednokierunkową. Prawie każda funckja w momencie wywołania przez funckje "wykonaj program" wywołaną w funckji głównej main wymaga argumentu typu t stos**, który jest wskaźnikiem na wskaźnik, który wskazuje na stos.Funckje print czy full print wymagają tylko wskaźnika na stos.DOdatkowo funckja push potrzebuje wiedzieć jaką liczbę ma dodać do stosu, więc wywoływana jest również z argumentem int liczba. Wszystkie funckje są typu void -> nie ma potrzeby żeby nam coś zwracały, jedynie funckja pomocnicza size, zwraca nam ilość elementów w stosie. Definiowanie struktury oraz nagłówki funckji zostały umieszczone w jednym pliku "kalkulator rpn.h".

```
#ifndef KALKULATOR_RPN_H
#define KALKULATOR_RPN_H
/*Struktura stosu dla kalkulatora*/
typedef struct stos {
    int dane;  /*Przechowujemy tutaj liczb  ca kowit */
    struct stos *nastepny;  /*Wskaznik na nast pn  struktur
    stos*/
}t_stos; /*Zmienna typu t_stos*/
/*Nag  wek funkcji, kt ra jest g  wn  funkcj  w programie*/
/*Wczytuje dane i wywo uje wszystkie inne funkcje programu*/
```

```
10 void wykonaj_program(t_stos *);
11 /*Nag  wki funkcji do operacji na stosie*/
12 int size(t_stos *);
13 void init(t_stos **);
14 void push(t_stos **,int );
15 void pop(t_stos **);
16 /*Nag  wki funkcji do operacji na kalkulatorze(stosie)*/
17 void clear(t_stos **);
18 void reverse(t_stos **);
19 void duplication(t_stos **);
20 void full_print(t_stos *);
21 void print(t_stos *);
22 void addition(t_stos **);
23 void subtraction(t_stos **);
24 void multiplication(t_stos **);
25 void division(t_stos **);
26 #endif
```

Listing 1: struktura stosu i nagłówki funckji

3.2 Asercje

W naszym programie mogą zdarzyć się błędy, które powinny zostać wykryte i zakomunikowane odpowiednio wyraźnie.Program sprawdza takie błędy jak:

1. Poprawność wczytywanych opcji.
2. Wcytywanie tylko liczb całkowitych,zmiennoprzecinkowe są odrzucane.
3. przydzielanie nowej pamięci, sprawdzamy czy pamięć jest pomyślnie zaalokowana.
4. Błędy operacyjne przy dodawaniu,odejmowaniu,dzieleniu,mnożeniu -> muszą w stosie znajdować się przynajmniej dwie liczby.
5. Błąd dzielenia przez zero równie istotny.
6. Błędy przy operacjach na stosie: funckja reverse wymaga dwóch liczb w stosie oraz funckje jak: duplication, pop, full print, print, wymagają aby stos nie był pusty aby funckje zadziałały poprawnie.

3.3 Funckja wykonaj program

Ta funckja jest istotna, wczytuje dane ze standardowego wejścia i odrazu po wczytanym znaku, wywołuje określone operacje zgodnie z funkcjonalnością programu.Działa do moemntu napotkania znaku 'q', Znak 'q' kończy działanie funckji wykonaj program jak i działanie całego programu. Zmienne: char znak,znak2 oraz int liczba są zmiennymi pomocniczymi do wczytywania znaków oraz liczb z stdin.Znaki spacji, tabulacji, oraz nowej linii są pomijane.Funkcja void nie ma potrzeby nic zwracać, gdy napotkamy błąd, dostaniemy tylko komunikat, aby poprawić swoje działanie, nie ma potrzeby kończyć program odrazu.Funckja korzysta ze switcha, który na podstawie wczytanego znaku, wie, którą funckję wywołać w programie.

```
1  void wykonaj_program(t_stos *stos){
2      char znak,znak2; /*Zmienna znak wczytuje znak ze stdin*/ /*
       znak2 pomocnicza zmienna do wczytywania znak w*/ int liczba;
       /*Zmienna liczba zapisuje liczb  z stdin, aby potem j  doda
        do stosu*/
3      init(&stos); /*Inicjacja pustego stosu*/
4      do{   /*Wykonuj si  dop ki znak!='q', kt ry ko czy program
       */
5          znak=getc(stdin);  /*Pobierz znak ze standardowego wej cia
       */
6          if((znak!=' ')&&(znak!='\t')&&(znak!='\n')){ /*spacja,
       tabulacja,enter musz  by  pomijane*/
7              switch(znak){  /*W zale no ci od wczytanego znaku
       wykonamy okre lone zadania*/
8                  case '+': {  addition(&stos); /*Mamy wykona
       dodawanie*/ break;}
9                  case '-': {  /*Wczytujemy kolejny znak,aby upewni
        si  czy wykona  odejmowanie b  d  */
10                     znak=getc(stdin); /*Czy mamy doda  ujemn
       liczb  do stosu*/
11                     if((znak<48)||(znak>57)) { /*Sprawdzamy czy
       obok minusa nie ma cyfry*/
12                         ungetc(znak,stdin); /*Je eli nie ma liczby
        zwr  znak z powrotem*/
13                         subtraction(&stos); /*Mamy wykona
       odejmowanie*/}
14                     else { /*Je eli jest cyfra zwr  znak*/
       ungetc(znak,stdin);
15                     scanf("%d",&liczba); /*Pobierz liczb  a  do
       napotkania innego znaku ni  cyfra*/
16                         znak=getc(stdin);  /*Pobieramy kolejny
       znak aby upewni  si  czy to liczba zmiennoprzecinkowa*/
17             switch(znak) {
18                 case '.': { /*Je eli obok liczby jest kropka
       sprawdzamy czy za kropk  jest znowu cyfra*/
19             znak2=getc(stdin);
20             if((znak2>=48)&&(znak2<=57)) {  /*Je eli tak, mamy
       liczb  z przecinkiem*/
```

```
21          fprintf(stderr,"B   d! Nale y wczyta  tylko liczby
    ca kowite!\n"); /*Zakomunikowanie  b   du*/ ungetc(znak2,stdin
    ); /*Zwr cenie znaku, kt ry jest liczb */
22          scanf("%d",&liczba); /*Pobranie wszytskich znak w za
    kropk , aby pozby  si  tej liczby*/}
23      else { /*Je eli za kropk  nie ma cyfry, mamy b   dn
    opcj  programu (znak '.' nie jest opcj )*/
24      ungetc(znak2,stdin); /*Zwracamy znak z cyfr */
25      ungetc(znak,stdin); /*Zwracamy kropk , aby potem mo
    zn w j  pobra  i wyrzuci  b   d nieprawid owej opcji*/
    push(&stos,(-1)*liczba); /*Dodanie ujemnej liczby do stosu*/ }
    break;}
26      default: { /*Gdy nie ma kropki dodajemy wcze niej
    wczytan  liczb  ujemn  do stosu oraz zwracamy znak*/ ungetc(
    znak,stdin); push(&stos,(-1)*liczba); break;}}}break;}
27              case '*': {multiplication(&stos); /*Mamy wykona
    mno enie*/ break;}
28              case '/': { division(&stos);  /*Mamy wykona
    dzielenie*/ break; }                    case 'P': { pop(&stos);
      /*Usuwamy liczb  ze szczytu stosu*/ break;}
29              case 'c': { clear(&stos); /*Czy cimy ca y stos*/
    break; }
30      case 'r': { reverse(&stos); /*Zamieniamy miejscami dwie
    pierwsze liczby w stosie*/ break;}
31              case 'd': { duplication(&stos);  /*Duplikacja
    liczby ze szczytu stosu*/ break;}
32              case 'p': { print(stos);   /*Wydrukowanie liczby ze
     szczytu stosu*/ break; }
33              case 'f': { full_print(stos); /*Wydrukowanie
    wszystkich liczb ze stosu*/ break;}
34              case 'q': { free(stos); /*Koniec programu,
    zwalnianie pami ci*/ stos=NULL; break;}
35              default: {  /*Inne opcje mog  by  cyframi b  d
    s  nieprawid owe*/
36                  if((znak>=48)&&(znak<=57)) { /*Musimy
    sprawdzi  czy mamy do czynienia z cyfr */
37                      ungetc(znak,stdin); /*Zwracamy cyfr */
38                      scanf("%d",&liczba); /*Wczytujemy wszystkie
     cyfry do napotkania innego znaku*/
39                      znak=getc(stdin); /*Wczytujemy kolejny znak
     -> czy jest kropka to wa ne*/
40              switch(znak){
41              case '.': { /*Je eli tak sprawdzamy czy za kropk
     jest kolejna cyfra*/
42              znak2=getc(stdin);
43              if((znak2>=48)&&(znak2<=57)) { /*Liczba
    zmiennoprzecinkowa -> zwr    b   d*/
44              fprintf(stderr,"B   d! Nale y wczyta  tylko
    liczby ca kowite!\n");
45              ungetc(znak2,stdin); /*Zwr    cyfr */
46              scanf("%d",&liczba); /*Wczytaj ca    liczb  za
    kropk */ }
47              else { /*Nie ma cyfry to mamy nieprawdi ow
    opcj  programu*/
48              ungetc(znak2,stdin); /*Zwro  cyfr */ ungetc(znak,
    stdin); /*Zwr    kropk */
```

```
49                  push(&stos,liczba); /*Dodaj dodatni  liczb  do
    stosu*/} break;}
50                default: { /*Nie ma kropki to zwracamy kropk  i
    dodajemy wcze niej wczytan  liczb  do stosu*/ ungetc(znak,
    stdin); push(&stos,liczba); break;}
51                } /*Je eli znak nie jest cyfr , musi by
    nieprawid owy -> zwr    b   d */
52                }else fprintf(stderr,"Niepoprawna opcja!!\n");
    break;}}}}while(znak!='q');}
```

Listing 2: Funckja wykonaj program

4 Funckje kalkulatora,stosu

Wszystkie funckje(oprócz size) nie muszą nic zwracać (typ void). Są wywoływane ze wskaźnikiem na stos, bądź na wskaźnik na wskaźnik, który pokazuje na stos. Funckje mogą zostać wywoływane jedna w drugiej, ich współpraca odpowiada za prawidłowe działanie całego programu.

4.1 Init

Inicjacja pustego stosu, Ustawianie wskaźnika na NULL.

```
1 void init(t_stos **wsk_stos){
2     *wsk_stos=NULL;  }
```

Listing 3: Init

4.2 Push

Dodawanie liczb do stosu, alokacja nowej pamięci na każdy nowy element, inicjowanie zmiennej pomocniczej pom, do poruszania się po stosie.

```
1 void push(t_stos **wsk_stos, int liczba){
2     if(*wsk_stos==NULL){    /*Gdy lista jest pusta nie potrzebujemy
       zmiennej pomocniczej*/
3         *wsk_stos= (t_stos *) malloc (sizeof(t_stos));
4         if(*wsk_stos==NULL){ /*Je eli malloc zwr   i NULL ->
      b    d przydzielenia pami ci*/
5             fprintf(stderr,"Blad przydzia u pami ci!!\n");
6             free(*wsk_stos);} else { /*DOdajemy liczb  do stosu na
       sam szczyt*/
7     (*wsk_stos)->dane=liczba;(*wsk_stos)->nastepny=NULL;  /*
      Ustawiamy wskaznik na NULL WA NE!!*/}   }else {  /*Gdy lista
      nie jest pusta inicujemy pomocnicz  zmienn  pom*/
8         t_stos *pom = (t_stos *) malloc (sizeof(t_stos)); /*
      Alokacja nowej pami ci*/
```

```
9       if(pom==NULL){  /*Zwr cenie  b   du  w przypadku niepowodzenia
        */fprintf(stderr,"B   d  przydzia u  pami ci\n"); free(pom); }
        else{ /*DOdawanie liczby*/
10          pom->dane=liczba;  /*Ustawianie wskaznika nast pny na
        element poprzedni*/
11          pom->nastepny=*wsk_stos;
12          *wsk_stos=pom;  /*LIsta(stos) teraz pokzauje na now
        liczb */}}}
```

Listing 4: Push

4.3 Pop

Usuwanie liczby ze szczytu stosu, Zwalnianie pamięci nadmiarowej, inicjowanie zmiennej pomocniczej pom, aby solidnie dokonać zmian na stosie.Sprawdzanie czy stos pusty. Jeżeli tak usuwanie liczby nie ma sensu, zwracamy komunikat z błędem.

```
1  void pop(t_stos **wsk_stos){
2       if(*wsk_stos!=NULL){  /*SPrawdzamy czy stos nie jest pusty*/
3           t_stos *pom = (*wsk_stos)->nastepny; /*Je eli nie ->
        usuwamy liczb  ze szczytu*/
4           free(*wsk_stos); /*Zwalnianie pami ci "usuni tego"
        elementu*/
5           *wsk_stos=pom;   /*Lista pokazuje na poprzedni element*/ /*
        pom zmienna pomocnicza*/
6       }else{  /*Stos pusty? -> nie ma czego usuwa */
7           fprintf(stderr,"Stos jest pusty!\n ");}}
```

Listing 5: Pop

4.4 Print

Drukowanie liczby ze szczytu stosu.Sprawdzenie czy stos pusty. Jeżeli nie jest pusty drukujemy liczbę ze szczytu listy(stosu).

```
1  void print(t_stos *wsk_stos){ /*SPrawdzamy czy lista nie jest pusta
       */
2       if(wsk_stos==NULL) fprintf(stderr,"Stos jest pusty!!\n");
3       else printf("%d\n", wsk_stos->dane); /*Wypisujemy licizb , na
        kt  r   poakzuje wsk_stos */}
```

Listing 6: Print

4.5 Full print

Wydrukowanie wszystkich elementów ze stosu, SPrawdzenie czy stos jest pusty, inicjowanie zmiennej pomocniczej pom, Dopóki nie napotkamy na NUll(koniec listy) wypisujemy elementy stosu.

```
void full_print(t_stos *wsk_stos){ /*Sprawdzenie czy stos nie jest
    pusty*/
    if(wsk_stos==NULL) fprintf(stderr,"Stos jest pusty!!\n");
    else {  /*Zmienna pomocnicza */ t_stos *pom = wsk_stos;
        do{ /*Dopoki nie napotkamy NULL -> koniec stosu to
    drukujemy liczby, kt re s  w stosie*/
            printf("%d", pom->dane); printf("\n");
            pom=pom->nastepny; /*Przechodzimy mi dzy kolenymi
    elementami stosu*/}while(pom != NULL); }}
```

Listing 7: Full print

4.6 Clear

Wyczyszczenie zawartości stosu, Sprawdzenie czy stos jest pusty, zmienna pomocnicza liczba elementów zostaje nadpisana przez funckje size, która zlicza ilość liczb w stosie. Dopóki jest ich więcej niż 0 wywołyjemy funckje pop(usuń element). Po wszystkim stos jest pusty, wskaźnik na stos pokazuje na NULL.

```
void clear(t_stos **wsk_stos){
    if(*wsk_stos==NULL) fprintf(stderr,"Stos jest pusty!\n"); /*
    SPrawdzenie czy stos pusty*/else {
    int liczba_elementow = size(*wsk_stos); /*Wywo anie funcji
    size, kt ra zlicza wszystkie liczby na stosie*/while(
    liczba_elementow>0){   /*Tyle ile liczb -> tyle wywo aj
    funkcje pop -> usuwanie ka dego elementu*/ pop(wsk_stos);
    liczba_elementow--;}   /*PO zako czeniu stos jest pusty,
    wskaznik ma warto    NULL*/}}
```

Listing 8: Clear

4.7 Reverse

Zamiana miejsc liczby ze szczytu stosu z poprzednim elementem. SPrawdzamy czy stos pusty, wymagane są conajmniej dwie liczby w stosie. Inicjowanie zmiennej pomocniczej na potrzeby operacji, Nadpisywane zmiennych liczba1 oraz liczba2, przechowują elementy ze stosu chwilowo, usuwanie dwóch pierwszych elementów funckją pop, następnie dodawanie tych samych elementów w odwrotnej kolejności.Tak zachodzi zamiana miejsc dwóch liczb.

```
void reverse(t_stos **wsk_stos){
    if(*wsk_stos==NULL) fprintf(stderr,"Stos jest pusty\n"); /*Czy
    stos pusty?*/
    else if(size(*wsk_stos)<2) fprintf(stderr,"Na stosie jest mniej
     ni   dwie liczby!\n");
    else {  /*Musz  by  conajmniej dwie liczby w stosie*/
        t_stos *pom= (*wsk_stos)->nastepny; /*Zmienna pomocnicza*/
```

```
6         int liczba1 = (*wsk_stos)->dane; /*Liczba1 zapisuje
    pierwsz  liczb  ze stosu*/
7         int liczba2= pom->dane;          /*Liczba2 zapisuje drug
    liczb  ze stosu*/
8         pop(wsk_stos);                  /*Usuwanie chwilowe tych
    dw ch element w*/
9         pop(wsk_stos);
10        push(wsk_stos,liczba1);       /*Dodanie najpierw liczba1
    potem liczba2*/
11        push(wsk_stos,liczba2);}    /*Tak nast pi a "zamiana"
    miejscami*/}
```

Listing 9: Reverse

4.8 Duplication

Duplikowanie liczby ze szczytu stosu, Sprawdzenie czy stos jest pusty, wymagana conajmniej jedna liczba na stosie, zmienna pomocnicza duplikat zawiera wartość tej liczby ze szczytu.Na koniec dodawanie tej liczby(kopii) na sam szczyt.

```
1 void duplication(t_stos **wsk_stos){ /*Czy stos pusty?*/
2 if(*wsk_stos==NULL) fprintf(stderr,"Stos jest pusty!\n");
3     else {  /*Duplikat -> tworzenie i dodanie do stosu*/int
      duplikat = (*wsk_stos)->dane;
4     push(wsk_stos,duplikat);}}
```

Listing 10: Duplication

4.9 Addition

DOdawanie dwóch liczb z góry stosu,Wymagane są conajmniej dwie liczby na stosie,Inicjowanie zmiennej pomocniczej pom, Zmienne liczba1 oraz liczba2 zostają nadpisane o wartość tych dwoch liczb z góry stosu,Usuwanie dwóch liczb, w zamian do stosu dodajemy ich wynik sumy.

```
1 void addition(t_stos **wsk_stos){ /*Brak liczb brak wyniku*/
2     if(*wsk_stos==NULL) fprintf(stderr,"Nie wprowadzono  adnych
      liczb ca kowitych!\n");
3     else if(size(*wsk_stos)<2) fprintf(stderr,"Na stosie jest mniej
       ni  dwie liczby!\n");
4     else {                                  /*Musz  by
      conajmniej dwie liczby w stosie*/
5         t_stos *pom = (*wsk_stos)->nastepny;  /*Zmienna pomocnicza
      */
6         int liczba1 = (*wsk_stos)->dane;   /*liczba1 zapisuje
      element ze szczytu*/
7         int liczba2= pom->dane;          /*Liczba2 zapisuje drugi
      element*/
```

```
8           pop(wsk_stos);   /*Usuwanie liczb ze stosu*/
9           pop(wsk_stos);
10          push(wsk_stos,liczba1+liczba2);   /*DOdanie wyniku sumy
        dw ch liczb na sam szczyt*/}}
```

Listing 11: Addition

4.10 Subtraction

Odejmowanie dwóch liczb z góry stosu,Wymagane są conajmniej dwie liczby na stosie,Inicjowanie zmiennej pomocniczej pom, Zmienne liczba1 oraz liczba2 zostają nadpisane o wartość tych dwoch liczb z góry stosu,Usuwanie dwóch liczb, w zamian do stosu dodajemy ich wynik różnicy.

```
1  void subtraction(t_stos **wsk_stos){  /*Brak liczb brak wyniku*/
2      if(*wsk_stos==NULL) fprintf(stderr,"Nie wprowadzono  adnych
       liczb ca kowitych!\n");
3      else if(size(*wsk_stos)<2) fprintf(stderr,"Na stosie jest mniej
        ni  dwie liczby!\n");
4      else {                                      /*Musz  by
       conajmniej dwie liczby w stosie*/
5          t_stos *pom = (*wsk_stos)->nastepny;  /*Zmienna pomocnicza
       */
6          int liczba1 = (*wsk_stos)->dane;  /*liczba1 zapisuje
       element ze szczytu*/
7          int liczba2= pom->dane;  /*Liczba2 zapisuje drugi element*/
8          pop(wsk_stos); /*Usuwanie liczb ze stosu*/
9          pop(wsk_stos);
10         push(wsk_stos,liczba2-liczba1); /*Dodanie wyniku r  nicy
       dw ch liczb na sam szczyt*/}}
```

Listing 12: Subtraction

4.11 Multiplication

Mnożenie dwóch liczb z góry stosu,Wymagane są conajmniej dwie liczby na stosie,Inicjowanie zmiennej pomocniczej pom, Zmienne liczba1 oraz liczba2 zostają nadpisane o wartość tych dwoch liczb z góry stosu,Usuwanie dwóch liczb, w zamian do stosu dodajemy ich wynik iloczynu.

```
1  void multiplication(t_stos **wsk_stos){  /*Brak liczb brak wyniku*/
2      if(*wsk_stos==NULL) fprintf(stderr,"Nie wprowadzono  adnych
       liczb ca kowitych!\n");
3      else if(size(*wsk_stos)<2) fprintf(stderr,"Na stosie jest mniej
        ni  dwie liczby!\n");
4      else {                                      /*Musz  by
       conajmniej dwie liczby w stosie*/
5          t_stos *pom = (*wsk_stos)->nastepny; /*Zmienna pomocnicza*/
```

```
6           int liczba1 = (*wsk_stos)->dane; /*liczba1 zapisuje element
     ze szczytu*/
7           int liczba2= pom->dane; /*Liczba2 zapisuje drugi element*/
8           pop(wsk_stos); /*Usuwanie liczb ze stosu*/
9           pop(wsk_stos);
10          push(wsk_stos,liczba1*liczba2); /*Dodanie wyniku iloczynu
    dw ch liczb na sam szczyt*/ }}
```

Listing 13: Multiplication

4.12 Division

Dzielenie dwóch liczb z góry stosu,Wymagane są conajmniej dwie liczby na stosie,Inicjowanie zmiennej pomocniczej pom, Zmienne liczba1 oraz liczba2 zostają nadpisane o wartość tych dwoch liczb z góry stosu,Usuwanie dwóch liczb, w zamian do stosu dodajemy ich wynik ilorazu.Funckja sprawdza czy liczba na szczycie stosu jest równa 0(nie dzielimy przez zero).

```
1  void division(t_stos **wsk_stos){  /*Brak liczb brak wyniku*/
2      if(*wsk_stos==NULL) fprintf(stderr,"Nie wprowadzono  adnych
       liczb ca kowitych!\n");
3      else if(size(*wsk_stos)<2) fprintf(stderr,"Na stosie jest mniej
        ni  dwie liczby!\n");
4      else {                                    /*Musz  by
       conajmniej dwie liczby w stosie*/
5          t_stos *pom = (*wsk_stos)->nastepny; /*Zmienna pomocnicza*/
6          int liczba1 = (*wsk_stos)->dane; /*liczba1 zapisuje element
        ze szczytu*/
7          int liczba2= pom->dane; /*Liczba2 zapisuje drugi element*/
8          if(liczba1==0) fprintf(stderr,"Nie dzielimy przez zero!\n")
       ;/*MIanownik nie mo e si  zerowa */
9          else {
10         pop(wsk_stos); /*Usuwanie liczb ze stosu*/
11         pop(wsk_stos);
12         push(wsk_stos,liczba2/liczba1);} /*Dodanie wyniku ilorazu
       dw ch liczb na sam szczyt*/}}
```

Listing 14: Division

4.13 Size

Funckja pomocnicza do zliczania liczb w stosie. Zwraca ich ilość (return licznik), inicjiowanie zmiennej pomocniczej pom, oraz zmiennej licznik=0.Dopóki nie napotkamy na koniec listy, inkrementujemy zmienną licznik.Wskaznik pokazuje na następny element aż do napotkania Nulla.

```
1  int size(t_stos *wsk_stos){
2      int licznik=0; /*Licznik zlicza liczby na stosie*/
```

```
    t_stos *pom = wsk_stos; /*Zmienna pomocnicza*/
    while(pom!=NULL){     /*Dopoki nie napotkamy koniec listy*/
        licznik++;       /*ZLiczamy liczb  element w stosu*/
        pom=pom->nastepny;  /*Poruszay si  po ka dym elemencie*/}
        return licznik;  /*Zwr    ilo    liczb na stosie*/}
```

Listing 15: Size

5 Testy

Formę automatycznych testów dokonano z pliku testowego przygotowanego wcześniej: "plik testowy.txt". W pliku znajduje się 8 strumieni danych, które starają się zbadać poprawność funkcji jak i inne szczególne przypadki.Pliki main.c kalkulator rpn.h oraz kalkulator rpn.c zostały lekko zmodyfikowane aby, dane były wczytywane z pliku, nie ma potrzeby wpisywać dane z klawiatury. Oczywiście testy zostały dokonane własnoręcznie i na ich podstawie wrzucone do jednego pliku.

```
#TEST 1 Dane wej ciowe: 4 7 14 167 f + f + f P f P f 47 12 45 23 f - f - f q
#Dodawanie, odejmowanie, drukowanie zawarto ci stosu, usuwanie liczby ze szczytu stosu.
4 7 14 167 f + f + f P f P f 47 12 45 23 f - f - f q
#TEST 2 Dane wej ciowe: -45 -56 -23 -67 f * f * f  c f 123 -45 34 17 f / f / f q
#Pr ba wprowadzania liczb ujemnych, mno enie,dzielenie, czyszczenie zawarto ci stosu.
-45 -56 -23 -67 f * f * f r f c f 123 -45 34 -17 f / f / f q
#TEST 3 Dane wej ciowe: 24 -57 f r f d f d f c f 234 67 45 f p q
#Dzia anie funkcji reverse, drukowanie liczby ze szczytu oraz pr ba jej duplikacji
24 -57 f r f d f d f c f 234 67 45 f p q
#TEST 4 Dane wej ciowe: 2 0 f / r f / f P d 5 r q
#Dzielenie przez zero,odwr cenie liczb(reverse),ponowna pr ba dzielenia,duplikacja,reverse.
2 0 f / r f / f P d 5 r q
#TEST 5 Dane Wej ciowe: --45    - 45 1.57    2345.2    0.0 -0.0 - 0.0
#Odr  nienie liczb ujemnych od odejmowania, odst py mi dzy znakami, liczby zmiennoprzecinkowe
-        -45         - 45 1.57      2345.2    0.0 -0.0 -     0.0
#TEST 6 Dane wej ciowe: \ [ ' ' ] e kjs 2 f + - * P f + - / q
#Nieprawdi owe opcjie,pr ba operacji na jednej liczby oraz na pustym stosie.
\ [ ' ' ] e kjs 2 + - / P f + - * q
#TEST 7 Dane wej ciowe: 12 567 32 54 24 f + f - f * f / f + f 12 4 -567drpPfq
#Wykonywanie dzia a  do skutku, wczytywanie znak w obok siebie.
12 567 38337 125646 24 f + f - f * f / f + f 12 4 -567drpPfq
#TEST 8 Dane wej ciowe: 2147483647 1 f + f c -2147483647 -1 f + f c 9999999999 0000000000 f / f q
```

```
#Wczytywanie liczb ca kowitych poza ich zakres(od -2147483647 do
    2147483367).
2147483647 1 f + f c -2147483647 -1 f + f c 9999999999 0000000000 f
    / q
```

5.1 TEST 1

Dane wejściowe: 4 7 14 167 f + f + f P f P f 47 12 45 23 f - f - f q Test sprawdza działanie operacji dodawania oraz odejmowanie tylko liczb dodatnich całkowitych,Ponadto Sprawdzamy funckje pop, czy elementy zostają prawidłowo usunięte, Wszystkie operacje i ich wyniki są drukowanę funckją full print. Na sam koniec kończyymy test znakiem 'q'.Na wyjściu dostaliśy ciąg znaków każdy odzielonny nową linią:
167
14
7
4
181
7
4
188
4
4
Stos jest pusty!!
23
45
12
47
22
12
47
-10
47 Jak widać rezultaty są poprawne, wczytaliśmy 4 liczby potem zostały wydrukowane poprawnie(1 wczytana jest na dole, ostatnia na górze)potem po operacji dodawania jednej i drugiej stos się zmniejszył o dwe liczby, gdyż dodaje dwie do siebie i zwraca wynik sumy. Po usunięciu dwukrotnie liczb ze stosu i po wyświetleniu zawartości widzimy, że jest stos pusty -> pop działap poprawnie.Operacje odejmowania działają na tej samej zasadzie.Na koniec znak 'q' kończy

test z pozytywnym wynikiem.

5.2 TEST 2

Dane wejściowe:-45 -56 -23 -67 f * f * f r f c f 123 -45 34 -17 f / f / f q Test 2 wczytuje zarówno liczby dodatnie jak ujemne, Wykonujemy mnożenie,dzielenie, oraz czyścimy zawartość stosu.Oto dane wyjściowe:
-67
-23
-56
-45
1541
-56
-45
-86296
-45
-45
-86296
Stos jest pusty!!
-17
34
-45
123
-2
-45
123
22
123 Jak widać ujemne znaki wczytują się prawdiłowo(znak minu musi znajdować się obok dowolnej cyfry), Mnożenie liczb również wykonuje działanie na dwóch liczbach, robi z nich wynik iloczynu i zwraca do stosu na szczyt.Dzielenie działa na podobnej zasadzie. Przed dzieleniem wyczyściliśmy stos znakiem 'c' pomyślnie.PO wydrukowaniu widzimy, że stos jest pusty!, zanak 'q' kończy test 2. NIc nadzwyczajnego się nie wydarzyło, test był wykonany z wynikiem poprawnym.

5.3 TEST 3

Dane wejściowe: 24 -57 f r f d f d f c f 234 67 45 f p q Test 2 Wykonujemy operacje reverse i duplikatu liczby ze szczytu stosu, oraz print szczytu stosu.Wczytywane są liczby dwie, następnie zamieniane miejscami, potem duplikujemy dwukrotnie liczbę z góry stosu i wrzucamy na sam szczyt. Na koniec czyścimy stos i od początku wczytujemy kolejne 3 liczby, sprawdzamy działanie funckji print. 'q' kończy test 3. Oto dane wyjściowe:
-57
24
24
-57
24
24
-57
24
24
24
-57
Stos jest pusty!!
45
67
234
45
Wynik testu pokazał, że nasze trzy kolejne funckje print,reverse oraz duplication działają bez zarzutu.

5.4 TEST 4

Dane wejściowe: 2 0 f / r f / f P d 5 r q SPrawdzenie dzielenia przez zero, próba naprawy przez funckje reverse(zero teraz będzie w liczniku), ponowna operacja. Usuwanie wyniku , duplikowanie pustego stosu, dodanie tylko jednej liczby, zamiana miejscami jednej liczby -> sprawdziamy czy program wyrzuci błędy. 'q' KOńczy program. Oto dane wyjściowe:
0
2
Nie dzielimy przez zero!
2

0
0
Stos jest pusty!
Na stosie jest mniej niż dwie liczby! Wyskoczyły nam 3 błędy, nie możemy dzielić przez zero, duplikować pustego stosu, bądź odwracać jedną liczbę z pustką, Błędy poprawnie sie pojawiły. Po zamianie miejsc zera oraz liczby 2 dzielenie było możliwe więc wszytsko jest jak najbardziej poprawne.

5.5 TEST 5

Dane wejściowe: –45 - 45 1.57 2345.2 0.0 -0.0 - 0.0 Test 5 sprawdza jak program odróżnia znak '-', czy ma wykonać odejmowanie, bądź dodać ujemną liczbę do stosu. Test również stara się wczytywać dane z odstępem między sobą więcej niż jedna spacja.DOdatkowo co ważne wczytujemy liczby zmiennoprzecinkowe, a nie powinny one mieć miejscca w programie.Oto wyniki testu:
Nie wprowadzono żadnych liczb całkowitych!
Na stosie jest mniej niż dwie liczby!
45
-45
Błąd! Należy wczytać tylko liczby całkowite!
Błąd! Należy wczytać tylko liczby całkowite!
Błąd! Należy wczytać tylko liczby całkowite!
Błąd! Należy wczytać tylko liczby całkowite!
Błąd! Należy wczytać tylko liczby całkowite!
-90
DUżo błędów na wyjściu ale czy to dobrze? Jeżeli się przyjrzymy to jest to poprawna rekacja programy na dane wejściowe, Każda liczba zmienno przecinkowa nie została dodana do stosu, a pojawił się komunikat o błędzie, słusznie.Ponadto minus jest prawidłowo rozróżniany w programie(Gdy minus jest obok liczby to dodajemy liczbę ujemną, jeżeli nie wykonujemy odejmowanie), odstępy między znakami nie robią probelmu programowi aby je prawidłowo odczytać. Wynik testu jest jak najbardziej poprawny.

5.6 TEST 6

Dane wejściowe: [' '] e kjs 2 f + - * P f + - / q Wczytywanie nieprawdiłowych opcji, próby operacji na jednej liczbie tylko, bądź na pustym stosie. Oto dane wyjściowe:
Niepoprawna opcja!!
Niepoprawna opcja!!
Niepoprawna opcja!!
Niepoprawna opcja!!
Niepoprawna opcja!!
Niepoprawna opcja!!
Niepoprawna opcja!!
Niepoprawna opcja!!
Na stosie jest mniej niż dwie liczby!
Na stosie jest mniej niż dwie liczby!
Na stosie jest mniej niż dwie liczby!
Stos jest pusty!!
Nie wprowadzono żadnych liczb całkowitych!
Nie wprowadzono żadnych liczb całkowitych!
Nie wprowadzono żadnych liczb całkowitych!
DUżo błedów na wyjściu, nieprawdiłowe opcje nie są brane pod uwagę, wyrzucane są błędy, Gdy dodamy tylko jedną liczbę do stosu, to nie da rady wykonać żadnej operacji matematycznej(potrzebne są dwie), program wyrzuca błąd. Gdy stos jest pusty tym bardziej nic nie zrobimy, potrzebujemy conjamniej dwóch liczb do operacji matematycznych.Test jest poprawny.

5.7 TEST 7

Dane wejściowe: 12 567 32 54 24 f + f - f * f / f + f 12 4 -567drpPfq Wczytujemy 5 cyfr aby wykonać odpowiednio każdą operację na stosie, aż otrzymamy tylko jeden wynik w stosie. Dodatkowo wczytujemy ciąg znaków jeden za drugim, sprawdzimy czy program będzie wstanie je odróżnić od siebie. Oto dane wyjściowe:
24
125646
38337
567
12
125670

38337
567
12
-87333
567
12
-49517811
12
0
Na stosie jest mniej niż dwie liczby!
0
-567
-567
4
12
0
Jak widać z 5 cyfr została nam jedna.POtem nie możmy wykonać więcej operacji. Dodatkowo program dobrze odróznia znaki, osobno wykonuje funckje duplikacji, reverse, wyświetlanie szczytu stosu,usuwanie , drukowanie zawartości stosu oraz kończy program. Z odstępami czy blisko siebie program wyznacza granicę wczytywania, wie kiedy ma pobrać liczbę, kiedy znak,kiedy pobierać do skuku itd.Test pokazuje, że kalkulator działa w mairę przyzwoicie.

5.8 TEST 8

Dane wejściowe: 2147483647 1 f + f c -2147483647 -1 f + f c 9999999999 0000000000 f / f q SPrawdzenie jak zachowa się program, gdy wyjdziemy poza zakres inta.Do stosu zostaną podane odpowiednio większe liczby, wykonamy na nich operacje i sprawdzimy co się stanie. Oto wyniki:
1
2147483647
-2147483648
-1
-2147483647
-2147483648
0
1410065407

Nie dzielimy przez zero!
Gdy dodamy do liczby ze skraju jedynkę program zaczyna szwankować i robi się liczba ujemna, gdyż wyszliśmy poza zakres inta. Wczytanie liczby 99999999999 zamieniło się na liczbę 1410065407 co nie jest zgodne. Gdy podzielimy liczbę tą przez zero nie możemy bo przez zero się nie dzieli, Należy uważać na zakres wczytywanych liczb, lub zwiększyć pojemnik na liczbę (zamiast int posłużyć się long int bądź long long int itp.)Program nie zawsze działa dobrze i to daje nam ostrzegawczy sygnał, że należy brać każdy przypadek pod uwagę.

6 Wnioski

Nasz program kalkulator działa inaczej niż taki jaki znamy. Operacje na stosie jak i na liczbach znajdujących się na nim wykonują się w miarę poprawnie.Ważne jest wczytywanie tylko liczb całkowitych dodatnich bądź ujmenych. Należy wczytywać poprawne opcje programu, zwracać uwagę na zera w mianowniku, oraz na ilość liczb na stosie. Na każdy nowy element na stosie alokowana jest pamięć dynamicznie, gdy kończymy program opcją 'q', pamięć zostaje zwalniana. Na początku wywołania funkcji wykonaj program automatycznie inicjowany jest pusty stos(lista).Wczytywanie znaków może być dowolne, od ogromnych odstępów, aż po wczytywanie znaków jeden za drugim.Jeżeli pojawi się nieklarowna sytuacja, program zwraca komunikaty z błędem.Możliwe jest odróżnianie znaku '-' w programie. Aby wczytać ujemną liczbę należy poprzedzić ją znakiem '-' bez żadnych spacji.Testy programu pokazały, że funckje jak i operacje na stosie, kalkulatorze mogą działać poprawnie o ile zakres wczytywanych liczb mieści się w zakresie int.Rezultaty są zadowalające, a cały program na miarę swoich możliwości działa tak jak powinien.