Sprawozdanie Kalkulator RPN

Autor: Mikołaj Zapotoczny

22 stycznia 2020

Spis treści

Listings

1 Zawartość paczki

- `main.c` - tutaj znajduje się funkcja 'main' oraz obsługa wyboru opcji.
- `stos.c` - tutaj znajdują się funkcje realizujące dane operacje na stosie.
- `stos.h` - plik nagłówkowy dla stos.c, tutaj znajdują się warunki PRE i POST dla funkcji obsługujących stos.
- `opcje.c` - tutaj znajdują się funkcje realizujące zadania kalkulatora z wykorzystaniem funkcji obsługujących stos.
- `opcje.h` - plik nagłówkowy dla opcje.c, tutaj znajdują się warunki PRE i POST dla funkcji obsługujących opcje kalkulatora.
- `struktura.h` - tutaj jest struktura przechowująca informacje o stosie
- `Makefile`.
- `instrukcja.sh` - skrypt zawierający instrukcję obsługi programu (uruchamiany przez Makefile).
- `testn.sh` - gdzie w miejsce litery 'n' należy wstawić numer (0-10) testu. Są to skrypty umożliwiające automatyczne testy programu. Liczba skryptów: 11.

2 Kompilacja i uruchomienie programu

Po uruchomieniu pliku Makefile (na diablo) program się kompiluje (bez ostrzeżeń i błędów). Ponadto uruchamia się skrypt Instrukcja.sh, który wypisuje opcje programu.

2.1 Makefile i Instrukcje.sh

```
all:
  clear
  /bin/bash Instrukcja.sh
  gcc -Wall -pedantic main.c stos.c stos.h struktura.h opcje.c opcje.h
```

Listing 1: Makefile

```
echo "";
echo "Kalkulator RPN";
echo "Operacje wykonywane tak samo jak w kalkulatorze Unixowym 'dc' (odwrotnosc
    w stosunku do dzielenia i odejmowania).";
echo "";
echo "Opcje: ";
echo "+ ... dodawanie";
echo "- ... odejmowanie";
echo "x ... mnozenie";
echo "/ ... dzielenie calkowitoliczbowe";
echo "";
echo "P ... usuniecie szczytu stosu";
echo "c ... usuniecie calego stosu";
echo "r ... zamaina dwoch ostatio wprowadzonych danych";
echo "d ... duplikowanie dwoch ostatnio wprowadzonych danych";
echo "p ... wypisanie szczytu stosu";
```

```
16 echo "f ... wypisanie calego stosu";
17 echo "q ... zakonczenie dzialania";
18 echo "";
19 echo "UWAGA!";
20 echo "Gdy wprowadzisz bledna opcje, np: 'y' to zostanie ona zamieniona na
       liczbe '0' i dodana jako element do stosu.";
21 echo "Program nie dziala dla liczb ujemnych (fragment programu napisany na
       konsultacjach wykluczal opcje odejmowania).";
22 echo "";
23 echo "AUTOMATYCZNE TESTY";
24 echo "Testy przygotowane przeze mnie znajduja sie w skryptach testn.sh, gdzie
       litera 'n' to numer testu ze sprawozdania.";
25 echo "Przyklad wywolania testu: /bin/bash test1.sh";
26 echo "";
27 echo "Opcja uruchamiania programu z argumentami zostala dodana tylko ze wzgledu
        na testy.";
28 echo "Z tego wzgledu obsluguje ona tylko liczby jednocyfrowe.";
29 echo "Program mozna uruchomic z argumentami wejsciowymi lub bez nich, bez
       roznicy w dzialaniu.";
30 echo "";
31 echo "";
32 echo "";
33 echo "";
```

Listing 2: Instrukcja.sh

Po skompilowaniu się programu, można go uruchomić na dwa sposoby. Można poprostu wpisać: './a.out', co uruchomi kalkulator i wprowadzi oczekiwanie na wprowadzanie dalszych danych. Można również uruchomić program wraz z argumentami, np './a.out 2 3 4 + r f', co spowoduje wykonanie najpierw operacji na wejsciu, a następnie innych operacji wpisywanych w trakcie pisania programu.

Funkcje obsługujące oba sposoby pracy z programem są identyczne. Występuje tutaj tylko jedna różnica wynikająca z przystosowania wywołania z argumentami tylko do testowania za pomocą skryptów. Otóż ta opcja obsługuje tylko liczby jednocyfrowe.

W całym programie są obsługiwane tylko liczby nieujemne! Poniższy fragment kodu napisany na konsultacjach wykluczał opcję odejmowania, ale wtedy rzeczywiście opcja liczb ujemnych działała poprawnie.

2.2 Realizacja użycia liczb ujemnych w programie

```
1  case '-':                        /* realizacja odejmowania */
2          if(tab[0]-'0')
3          {
4              push((int)tab[0]-'0',&stos); /* ten for sprawia, ze program
       poprawnie wczytuje liczby ujemne wykorzystujac kod ASCI (konsultacje) */
5              break;
6          }
7
8
9          if(empty(stos)>1)              /* sprawdzenie czy na stosie jest wiecej
       niz jedna dana, aby moc wykonac operacje */
10         {
11             dana1=stos->dana;        ...........
```

Listing 3: Liczby ujemne

2.3 Listing funkcji main

```
1
2       /*************************************************************/
3       /* Funkcja main, wprowadzanie danych                         */
4       /* Argumenty funkcji:                                        */
5       /*   int argc - liczba wpisanych argumentow podczas wywolania*/
6       /*   char **argv - tablica przechowujaca dane z wywolania    */
7       /* PRE:                                                      */
8       /*      wprowadzane dane (liczby lub funkcje kalkulatora)    */
9       /*      ze standardowego wejscia w momencie dzialanie funkcji*/
10      /*      lub jako argumenty wywolania                         */
11      /* POST:                                                     */
12      /*      wywolanie funkcji 'opcje' i zwracanie '0' gdy koniec */
13      /*      dzialania programu                                   */
14      /*************************************************************/
15  int main(int argc, char **argv)
16  {
17      elem *stos;         /* zmienna typu 'elem' odwolujaca sie do struktury */
18      inicjuj(&stos);     /* inicjowanie stosu */
19
20      char tab[100];
21      /* tablica, do ktorej wczytujemy informacje ze standardowego wejscia */
22
23
24  /* Ponizszy fragment kodu jest tutaj ze wzgledu na automatyczne testy -
       przeznaczony do obslugi argumentow wejscia */
25  /* Ze wzgledu na przeznaczenie do testowania, pozwala on tylko na operacje na
       liczbach jednocyfrowych, dalsza czesc kodu nie ma juz takich ograniczen */
26
27  ////////////////////////////////////////////////////////////////////////////
28
29      int i;
30      for(i=1; i<argc; i++)   /* petla wskazujaca kolejne argumenty wywolania */
31      {
32          tab[0]=argv[i][0];
33          /* przypisywanie kolejnych argumentow jako tab[0], aby potem
       wykorzystac to w okraslonych funkcjach */
34
35          if(opcje(tab)==0)   /* sprawdzanie, czy funkcja opcje nie zwrocila 0 */
36          {
37              return 0;       /* jesli tak, to koniec dzialania programu */
38          }
39      }
40
41  ////////////////////////////////////////////////////////////////////////////
42
43
44  /* Ponizszy fragment kodu jest przeznaczony do dalszej obslugi programu po
       wykonaniu operacji na argumentach wejsciowych */
45
46  /*************************************************************************/
47
48      while(1)
49      {
50          scanf("%s",tab);    /* wprowadzanie danych ze standardowego wejscia */
```

```
51
52          if(opcje(tab)==0)    /* sprawdzanie, czy funkcja opcje nie zwrocila 0 */
53          {
54              return 0;        /* jesli tak, to koniec dzialania programu */
55          }
56      }
57  }
```

Listing 4: Funkcja 'main'

3 Realizacja stosu

3.1 Funkcja stos.c

```
1
2   // Zainicjowanie stosu
3   void inicjuj(elem **szczyt)
4   {
5   *szczyt=NULL;
6   /* na koncu listy musi byc wartosc NULL, wiec na poczatku taki musi byc szczyt
        stosu */
7   }
8
9
10
11  // Odlozenie na gorze stosu
12  void push(int dana, elem **szczyt)
13  {
14  elem *nowy=malloc(sizeof(elem));
15  /* stworzenie miejsca na poczatku listy na nowy element, co jest rozumiane jako
        szczyt */
16  nowy->dana=dana;             /* nadanie wartosci nowemu elementowi */
17  nowy->nastepny=*szczyt;
18  /* stworzenie wskaznika na poprzedni element (poprzedni szczyt) */
19
20  *szczyt=nowy;                /* zapisanie nowego elementu jako (nowy) szczyt */
21  }
22
23
24
25  // Zdjecie ze stosu
26  void pop(elem **szczyt)
27  {
28  elem *nowy=*szczyt;
29  /* zmienna pomocnicza, aby moc przekazac wskaznik i nie stracic dostepu do
        reszty listy */
30  *szczyt=(*szczyt)->nastepny;  /* przekazanie wskaznika */
31
32  free(nowy);                   /* zwolnienie pamieci */
33  }
34
35
36
37
38  // Sprawdzenie czy stos jest pusty
```

```
int empty(elem *szczyt)
{
    int i=0;      /* zmianna pomocnicza do zliczania elementow na stosie */
    if(szczyt)    /* warunek czy cokolwiek jest na stosie */
    {
        while(szczyt)
        {
        i++;
        /* gdy stos nie jest pusty do liczba elementow zwieksza sie o 1 */
        szczyt=szczyt->nastepny;
        /* gdy zmienna szczyt != NULL to bedzie sie wykonywac, gdy szczyt =
    NULL to oznacza koniec listy i koniec petli */
        }
        return i;    /* zwracana liczba elementow na stosie */
    }
    else
    {
        printf("Stos jest pusty! \n");  /* komunikat gdy stos jest pusty */
        return 0;
    }
}



// Wypisanie stosu
void print(elem *szczyt)
{
    if(szczyt)    /* warunek czy cokolwiek jest na stosie */
    {
        while(szczyt)
        {
        printf("%d ",szczyt->dana);    /* wypisanie wartosci */
        szczyt=szczyt->nastepny;
        /* 'przeskok' na kolejna wartosc, jezeli ta wartosc nie bedzie NULL to
    bedzie wykonywac sie dalej */
        }
        printf("\n");
    }
    else
    {
        printf("Stos jest pusty! \n");  /* komunikat gdy stos jest pusty */
    }
}
```

Listing 5: Stos 1

3.2 Nagłówek funkcji stos.c

```


    /*************************************************************/
    /* Funkcja inicjuje stos                                     */
    /* Argumenty funkcji:                                        */
    /*   element - struktura dzieki ktorej zostal zrealizowany stos */
    /*   **szczyt - zmienna typu elem odwolujaca sie do struktury   */
    /*              element dzieki ktorej mamy dostep do listy     */
```

```
9     /* PRE:                                                          */
10    /*      brak                                                     */
11    /* POST:                                                         */
12    /*      zainicjowany stos                                        */
13    /*****************************************************************/
14 void inicjuj(elem **szczyt);
15
16
17    /*****************************************************************/
18    /* Funkcja wykonuje odlozenie elementu na gorze stosu            */
19    /* Argumenty funkcji:                                            */
20    /*   element - struktura dzieki ktorej zostal zrealizowany stos */
21    /*   **szczyt - zmienna typu elem odwolujaca sie do struktury   */
22    /*              element dzieki ktorej mamy dostep do listy       */
23    /*   dana - wpisana dana majaca zostac odlozona na stosie        */
24    /* PRE:                                                          */
25    /*      poprawnie wpisana zmienna typu int                       */
26    /* POST:                                                         */
27    /*      stos zawierajacy jeden element wiecej                    */
28    /*****************************************************************/
29 void push(int dana, elem **szczyt);
30
31
32    /*****************************************************************/
33    /* Funkcja wykonuje usuniecie elementu z gory stosu              */
34    /* Argumenty funkcji:                                            */
35    /*   element - struktura dzieki ktorej zostal zrealizowany stos */
36    /*   **szczyt - zmienna typu elem odwolujaca sie do struktury   */
37    /*              element dzieki ktorej mamy dostep do listy       */
38    /* PRE:                                                          */
39    /*      stos zawierajacy co najmniej jedna dana                  */
40    /* POST:                                                         */
41    /*      stos zawierajacy jeden element mniej                     */
42    /*****************************************************************/
43 void pop(elem **szczyt);
44
45
46    /*****************************************************************/
47    /* Funkcja sprawdza czy stos jest pusty i ile ma elementow       */
48    /* Argumenty funkcji:                                            */
49    /*   element - struktura dzieki ktorej zostal zrealizowany stos */
50    /*   **szczyt - zmienna typu elem odwolujaca sie do struktury   */
51    /*              element dzieki ktorej mamy dostep do listy       */
52    /* PRE:                                                          */
53    /*      zainicjowany stos (pusty lub z elementami)               */
54    /* POST:                                                         */
55    /*      funkcja zwraca liczbe elementow na stosie                */
56    /*****************************************************************/
57 int empty(elem *szczyt);
58
59
60    /*****************************************************************/
61    /* Funkcja wypisuje stos                                         */
62    /* Argumenty funkcji:                                            */
63    /*   element - struktura dzieki ktorej zostal zrealizowany stos */
64    /*   **szczyt - zmienna typu elem odwolujaca sie do struktury   */
65    /*              element dzieki ktorej mamy dostep do listy       */
```

```
66      /* PRE:                                                          */
67      /*       stos zawierajacy co najmniej jedna dana                 */
68      /* POST:                                                         */
69      /*       wypisany stos                                           */
70      /*****************************************************************/
71  void print(elem *szczyt);
```

Listing 6: Stos 2

4 Testy funkcji 'opcje.c'

Kalkulator był testowany na diablo i na moim prywatnym komputerze. Testy na obu komputerach przebiegały tak samo i były porównywane z Unixowym kalkulatorem 'dc'. Dla ułatwienia czytania połączyłem listingi kodu głównego i nagłówka.

4.1 Switch, zakończenie działania (q), zapisywanie na stos

```
1   /*****************************************************************/
2   /* Funkcja wykonujaca operacje kalkulatora                       */
3   /* Argumenty funkcji:                                            */
4   /*   char *tab - wskaznik do tablicy przechowujacej flagi        */
5   /*               zadanych operacji                               */
6   /* Opreracje: +, -, x, /, P, c, r, d, p, f, q                    */
7   /* PRE:                                                          */
8   /*       dane wprowadzane jako argumenty wywolania lub wpisywane */
9   /*       recznie na bierzoco                                     */
10  /* POST:                                                         */
11  /*       wykonanie sie zadanych operacji                         */
12  /*****************************************************************/
13
14  int opcje(char *tab)
15  {
16      switch(tab[0])  /* funkcja wyboru */
17      {
18          case '+':
19          .
20          .
21          .
22          .
23          .
24
25          case 'q':   /* realizcja zakonczenia dzialania programu */
26          return 0;   /* zwracanie '0' co w 'main' bedzie skutkowac zakonczeniem
        dzialania programu */
27          break;
28
29
30          default:
31          push(atoi(tab),&stos);        /* konwersja char na int */
32          break;
33      }
34      return 1;
35  }
```

Listing 7: Switch, zakończenie działania, zapisywanie na stos

- Poniższy test ma na celu sprawdzenie operacji switch, kończenia pracy programu oraz zapisywania liczb na stos
- Wprowadzam wywołanie skryptu: /bin/bash test0.sh
- Skrypt zawiera ciąg znaków: ./a.out u 3 4 5 f q
- Na wyjściu otrzymałem: 5 4 3 0 ; oraz program skończył swoje działanie
- 0 zamiast u jest poprawnym działaniam, zgodnym z zapisem w instrukcji dołaczonej do programu
- Operacje zostały wykonane poprawnie, więc stwierdzam, że zapisywanie liczb przy pomocy atoi, które zamienia char na int, i kończenie pracy programu działa poprawnie.

4.2 Dodawanie (+) i odejmowanie (-)

```
1
2       case '+':                       /* realizacja dodawania */
3       if(empty(stos)>1)               /* sprawdzenie czy na stosie jest wiecej
    niz jedna dana, aby moc wykonac operacje */
4       {
5           dana1=stos->dana;           /* przypisanie danej1 szczytu stosu */
6           pop(&stos);                 /* usuniecie szczytu stosu */
7
8           dana2=stos->dana;           /* przypisanie danej2 szczytu stosu */
9           pop(&stos);                 /* usuniecie szczytu stosu */
10
11          push(dana1+dana2,&stos);
12          /* polozenie na szczyt stosu wyniku z dodawania tych dwoch
    zmiennych */
13      }
14      else
15      {
16          printf("Za malo danych! \n"); /* komunikat o braku danych */
17      }
18      break;
19
20
21
22      case '-':                       /* realizacja odejmowania */
23      if(empty(stos)>1)               /* sprawdzenie czy na stosie jest wiecej
    niz jedna dana, aby moc wykonac operacje */
24      {
25          dana1=stos->dana;           /* przypisanie danej1 szczytu stosu */
26          pop(&stos);                 /* usuniecie szczytu stosu */
27
28          dana2=stos->dana;           /* przypisanie danej2 szczytu stosu */
29          pop(&stos);                 /* usuniecie szczytu stosu */
30
31          push(dana1-dana2,&stos);
32          /* polozenie na szczyt stosu wyniku z odejmowania tych dwoch
    zmiennych */
33      }
34      else
35      {
```

```
            printf("Za malo danych! \n"); /* komunikat o braku danych */
        }
        break;
```

Listing 8: Dodawanie i odejmowanie

- Poniższy test ma na celu sprawdzenie operacji dodawania i odejmowania.
- Wprowadzam wywołanie skryptu: /bin/bash test1.sh
- Skrypt zawiera ciąg znaków: ./a.out 4 4 2 1 + - f q
- Na wyjściu otrzymałem: -1 4
- Operacje zostały wykonane poprawnie.

4.3 Mnożenie (x) i Dzielenie (/)

```
            case 'x':                       /* realizacja mnozenia */
            if(empty(stos)>1)               /* sprawdzenie czy na stosie jest
    wiecej niz jedna dana, aby moc wykonac operacje */
            {
                dana1=stos->dana;         /* przypisanie danej1 szczytu stosu */
                pop(&stos);               /* usuniecie szczytu stosu */

                dana2=stos->dana;         /* przypisanie danej2 szczytu stosu */
                pop(&stos);               /* usuniecie szczytu stosu */

                push(dana1*dana2,&stos);
                /* polozenie na szczyt stosu wyniku z mnozenia tych dwoch
    zmiennych */
            }
            else
            {
                printf("Za malo danych! \n"); /* komunikat o braku danych */
            }
            break;



            case '/':                       /* realizacja dzielenia */
            if(empty(stos)>1)               /* sprawdzenie czy na stosie jest
    wiecej niz jedna dana, aby moc wykonac operacje */
            {
                dana1=stos->dana;         /* przypisanie danej1 szczytu stosu */
                pop(&stos);               /* usuniecie szczytu stosu */

                dana2=stos->dana;         /* przypisanie danej2 szczytu stosu */
                pop(&stos);               /* usuniecie szczytu stosu */

                push(dana2/dana1,&stos);
                /* polozenie na szczyt stosu wyniku z dzielenia tych dwoch
    zmiennych */
            }
            else
            {
                printf("Za malo danych! \n"); /* komunikat o braku danych */
```

```
36              }
37              break;
```

Listing 9: Mnożenie i dzielenie

- Poniższy test ma na celu sprawdzenie operacji mnożenia i dzielenia.
- Wprowadzam wywołanie skryptu: /bin/bash test2.sh
- Skrypt zawiera ciąg znaków:./a.out 3 2 4 / x f q
- Na wyjściu otrzymałem: 6
- Operacje zostały wykonane poprawnie.

4.4 Usunięcie szczytu stosu (P)

```
1               case 'P':                       /* realizacja usuniecia ostatniej
    wprowadzonej liczby */
2               if(empty(stos)>0)               /* sprawdzenie czy na stosie jest co
    najmniej jedna dana, aby moc wykonac operacje */
3               {
4                   pop(&stos);                 /* usuniecie szczytu stosu */
5               }
6               break;
```

Listing 10: Usunięcie szczytu stosu

- Poniższy test ma na celu sprawdzenie operacji usunięcia szczytu stosu.
- Wprowadzam wywołanie skryptu: /bin/bash test3.sh
- Skrypt zawiera ciąg znaków: ./a.out P 2 3 4 5 f P f q
- Na wyjściu otrzymałem:
  Stos jest pusty! - komunikat o błędzie
  5 4 3 2
  4 3 2
- Operacje zostały wykonane poprawnie, błędy zostały odpowiednio zakomunikowane.

4.5 Usunięcie stosu (c)

```
1               case 'c':                       /* realizcja usuniecia wszystkich
    wprowadzonych liczb */
2               while(empty(stos)>0)
3               /* usuwanie kolejnych elemenow stosu, az zabraknie danych */
4               {
5                   pop(&stos);                 /* usuniecie szczytu stosu */
6               }
7               break;
```

Listing 11: Usunięcie stosu

- Poniższy test ma na celu sprawdzenie operacji usunięcia stosu.
- Wprowadzam wywołanie skryptu: /bin/bash test4.sh
- Skrypt zawiera ciąg znaków: ./a.out c 2 3 4 5 f c f q
- Na wyjściu otrzymałem:
  Stos jest pusty! - komunikat o błędzie (c)
  5 4 3 2
  Stos jest pusty! - komunikat po usunięciu stosu
  Stos jest pusty! - komunikat o błędzie (f)
- Operacje zostały wykonane poprawnie, błędy zostały odpowiednio zakomunikowane.

4.6 Zamiana dwóch danych na szczycie stosu (r)

```
1           case 'r':                       /* realizcja zamiany dwoch ostatnio
    wprowadzonych liczb */
2           if(empty(stos)>1)               /* sprawdzenie czy na stosie jest
    wiecej niz jedna dana, aby moc wykonac operacje */
3           {
4               dana1=stos->dana;         /* przypisanie danej1 szczytu stosu */
5               pop(&stos);               /* usuniecie szczytu stosu */
6
7               dana2=stos->dana;         /* przypisanie danej2 szczytu stosu */
8               pop(&stos);               /* usuniecie szczytu stosu */
9
10              push(dana1,&stos);        /* polozenie na szczyt stosu danych w
    odwrotnej kolejnosci */
11              push(dana2,&stos);
12          }
13              else
14          {
15              printf("Za malo danych! \n"); /* komunikat o braku danych */
16          }
17          break;
```

Listing 12: Zamiana dwóch danych na szczycie stosu

- Poniższy test ma na celu sprawdzenie operacji zamiany dwóch elementów na szczycie stosu.
- Wprowadzam wywołanie skryptu: /bin/bash test5.sh
- Skrypt zawiera ciąg znaków: ./a.out r 1 r 2 3 4 5 f r f q
- Na wyjściu otrzymałem:
  Stos jest pusty! - komunikat o błędzie (empty)
  Za malo danych! - komunikat o błędzie (r)
  Za malo danych! - komunikat o błędzie (r)
  5 4 3 2 1
  4 5 3 2 1
- Operacje zostały wykonane poprawnie, błędy zostały odpowiednio zakomunikowane.

4.7 Duplikacja szczytu stosu (d)

```
1            case 'd':                      /* realizcja zduplikowania ostanio
   wprowadzonej liczby */
2            if(empty(stos)>=1)             /* sprawdzenie czy na stosie co
   najmniej jedna dana, aby moc wykonac operacje */
3            {
4                dana1=stos->dana;          /* przypisanie danej1 szczytu stosu */
5                push(dana1,&stos);         /* polozenie na szczyt stosu danej1 */
6            }
7            else
8            {
9                printf("Za malo danych! \n"); /* komunikat o braku danych */
10           }
11           break;
```

Listing 13: Duplikacja szczytu stosu

- Poniższy test ma na celu sprawdzenie operacji duplikowania szczytu stosu.
- Wprowadzam wywołanie skryptu: /bin/bash test6.sh
- Skrypt zawiera ciąg znaków: ./a.out d 2 3 4 5 d f q
- Na wyjściu otrzymałem:
  Stos jest pusty! - komunikat o błędzie (empty)
  Za malo danych! - komunikat o błędzie (d)
  5 5 4 3 2
- Operacje zostały wykonane poprawnie, błędy zostały odpowiednio zakomunikowane.

4.8 Wypisanie szczyu stosu (p)

```
1            case 'p':                     /* realizcja wypisanie szczytu stosu */
2            if(empty(stos)>=1)            /* sprawdzenie czy na stosie jest co
   najmniej jedna dana, aby moc wykonac operacje */
3            {
4                dana1=stos->dana;         /* przypisanie danej1 szczytu stosu */
5                printf("%d \n",dana1);     /* wypisanie szczytu stosu */
6            }
7            break;
```

Listing 14: Wypisanie szczyu stosu

- Poniższy test ma na celu sprawdzenie operacji wypisanie szczytu stosu.
- Wprowadzam wywołanie skryptu: /bin/bash test7.sh
- Skrypt zawiera ciąg znaków: ./a.out p 2 3 4 5 p f q
- Na wyjściu otrzymałem:
  Stos jest pusty! - komunikat o błędzie (empty)
  5
  5 4 3 2
- Operacje zostały wykonane poprawnie.

4.9 Wypisanie stosu (f)

```
            case 'f':                   /* realizcja wypisanie calego stosu */
            if(empty(stos)>=1)          /* sprawdzenie czy na stosie jest co
    najmniej jedna dana, aby moc wykonac operacje */
            {
                print(stos);            /* wypisanie stosu */
            }
            break;
```

Listing 15: Wypisanie stosu

- Poniższy test ma na celu sprawdzenie operacji wypisanie stosu.
- Wprowadzam wywołanie skryptu: /bin/bash test8.sh
- Skrypt zawiera ciąg znaków: ./a.out f 2 3 4 5 f q
- Na wyjściu otrzymałem:
  Stos jest pusty! - komunikat o błędzie (empty)
  5 4 3 2
- Operacje zostały wykonane poprawnie.

4.10 Kilka przykładów łączących kilka opcji

TEST 1

- Wprowadzam wywołanie skryptu: /bin/bash test9.sh
- Skrypt zawiera ciąg znaków: ./a.out 2 3 + 4 5 - r d d P f q
- Na wyjściu otrzymałem: 5 5 1
- Operacje zostały wykonane poprawnie.

TEST 2

- Poniższy test ma na celu sprawdzenie operacji wypisanie stosu.
- Wprowadzam wywołanie skryptu: /bin/bash test10.sh
- Skrypt zawiera ciąg znaków: ./a.out 1 2 3 4 u f d f p q
- Na wyjściu otrzymałem:
  0 4 3 2 1
  0 0 4 3 2 1
  0
- Operacje zostały wykonane poprawnie.

Stwierdzam, że program działa poprawnie dla wszystkich poprawnych danych wejściowych!!!

5 Napotkane problemy

5.1 Odczytywanie char jako liczbę

Na początku miałem problem z dodawaniem liczb na stos, bo wczytywane znaki były jako char, a stos potrzebował int, co generowało błędy. Zaminienie wszystkich danych wejsciowych na int również dawało błędy, bo opcje programu, np P, p, d, r... są literami. Bezskutecznie próbowałem też ręcznie rzutować dane. Wtedy znalazłem funkcję atoi, która zamienia char na int bez błędów. Wstawiłem tę funkcję do push w default wewnątrz switch'a i teraz liczby i opcje poprawnie się wczytywają.

5.2 Asercje

Gdy za pierwszym razem testowałem funkcję r - zamianę dwóch elementów na szczycie stosu, to przez przypadek był to jedyny arguent jaki wprowadziłem do programu i wyskoczyło mi naruszenie ochrony pamięci. Wtedy zdałem sobie sprawę, że należy jeszcze dodać kilka asercji do niektórych opcji, aby uodpornić program na tego typu błędy. Trochę to trwało zanim wszystkie (mam nadzieję) przypadki zostały przeze mnie rozpatrzone, ale teraz program działa poprawnie.

5.3 Realizacja stosu

Pomimo udostępnionych materaiłów (na stronie kursu) imlpementacja stosu sprawiła mi nie mały kłopot. Badzo trudno jest dobrać odpowiednio wskaźniki w odpowiednich miejscach. Nie raz miałem taką sytuację, że wskaźnik pokazywał na zupełnie inne dane niż chciałem. Kolejnym problemem było gubienie danych w czasie trawnia programu. Potem okazało się, że przy dodawaniu kolejnych elementów do listy, która reprezntuje stos, nie było przekazywania wskaźnika na poprzedni element co właśnie rodziło gubienie danych.

5.4 Mnożenie nie działające przy wywołaniu programu z argumentami

Ten problem polegał na wypisywaniu się dziwnego ciągu zer przy korzystaniu z funkcji mnożenia za pomocą gwiazdki (*). Okazało się, że bash interpretuje gwiazdkę w inny sposób i przez to program nie chciał działać poprawnie. Zamiast gwiazdki użyłem znaku (x) i program dla tego znaku działa poprawnie.

5.5 Liczby ujemne w programie

Z tym problemem nietyety nie udało mi się wygrać nawet mając fragment kodu napisany na konsultacjach. Nie potrafiłem napisać odpowiednich warunków co do wykonywania się tych fragmentów kodu, tak, aby wszystkie działania (w tym przypadku odejmowanie) działały poprawnie.

FINITO!