OR 2. Domača naloga

Odločil sem se, da bom za drugo domačo nalogo naredil poseben kalkulator in sicer kalkulator obratne polske notacije.

Obratna polska notacija (OPN) je matematična notacija, ki se od tradicionalne (infix) razlikuje v načinu zapisovanja izrazov. V OPN se operaterji postavljajo za števila, ki jih manipulirajo, kar pomeni, da ni potrebe po uporabi oklepajev za določanje vrstnega reda operacij. To notacijo je razvil poljski matematik Jan Łukasiewicz in je pridobila priljubljenost zaradi svoje enostavnosti pri izvajanju računskih operacij, zlasti na računalniških kalkulatorjih. Omogoča enostavno izvajanje aritmetičnih operacij brez potrebe po sledenju pravilom prednosti operacij in uporabi oklepajev.

Računanje je najlažje implementirati s pomočjo sklada, pri čemer se števila pišejo na sklad in ko pridemo do znaka za operacijo npr. + se zgornji dve števili iz sklada odstranita, seštejeta in rezultat se nato ponovno shrani na sklad.

Moj kalkulator podpira 5 osnovnih operacij, to so seštevanje, odštevanje, množenje, deljenje in izračun ostanka pri deljenju. Spodaj bom obrazložil nekaj primerov delovanja.

- Dodajanje dveh števil:
  Infix: 3 + 5
  OPN: 3 5 +
  V OPN se najprej zapišeta števili 3 in 5 na sklad, nato pa sledi operator +, ki ju odstrani iz sklada, sešteje in rezultat shrani nazaj na sklad. Točen vrstni red se določi enoznačno, brez potrebe po oklepajih.
- Množenje in odštevanje:
  Infix: 4 * (7 - 2)
  OPN: 4 7 2 - *
  Pri izračunu OPN se najprej izvede odštevanje 7 in 2, nato pa se rezultat pomnoži s 4. Ponovno, vrstni red operacij je jasen in enostaven za sledenje.
- Primer bolj kompleksnega izraza:
  Infix: ((10 / 4) + 2) - (7 * 2)
  OPN: 10 4 / 2 + 7 2 * -
  REŠITEV:
  - Deli 10 z 4: 10 4 / 2 + 7 2 * - -> 2 2 + 7 2 * -
  - Seštej 2 in 2: 2 2 + 7 2 * - -> 4 7 2 * -
  - Pomnoži 7 in 2: 4 7 2 * - -> 4 14 –
  - Odštej 14 od 4: 4 14 - -> -10

Če je izraz pravilno nastavljen mora biti po koncu izračuna na skladu natanko en element in sicer rezultat. Kalkulator je nastavljen tako da zazna napako, če je na skladu več kot en element ali če je bil vnesen kakšen znak, ki ni številka, +, -, *, /, % in enter.

Vso kodo sem delal s pomočjo programskega jezika C.

Interakcijo izvajamo preko serijskega vmesnika s pomočjo programa Putty – moje nastavitve so COM3, hitrost 115200.

Obrazložitev delovanja kode

1. Struktura Stack ima dve vrednosti, prva je kazalec na kazalec za shranjevanje znakov – tabela števil, druga pa je skladovni kazalec, ki kaže na vrh sklada.

```
struct Stack {
    char **stack;
    int stackPointer;
};
```

2. Funkcija inicializira sklad. Rezervira prostor na kopici i nnastavi vrh sklada na -1.

```
void initializeStack(struct Stack *s) {
    s->stack = (char**) malloc(MAX_STACK_SIZE*sizeof(char*));
    s->stackPointer = -1;

    for (int i=0; i<MAX_STACK_SIZE; i++)
        s->stack[i] = (char*) malloc(MAX_STACK_SIZE*sizeof(char));
}
```

3. Funkcija sprosti pomnilniški prostor – uporabi se ko je izračun končan.

```
void destroyStack(struct Stack *s) {
    for (int i=0; i<MAX_STACK_SIZE; i++)
        free(s->stack[i]);

    free(s->stack);
}
```

4. Funkcija doda številko na sklad, ki je zapisana z znaki. Vrednost niza se skopira na vrh sklada.

```
void push(struct Stack *s, char *str) {
    if (s->stackPointer < MAX_STACK_SIZE - 1) {
        s->stackPointer++;
        strcpy(s->stack[s->stackPointer],str);
    } else {
        printf("Sklad je poln. Znak %s ni bil dodan.\n", str);
    }
}
```

5. Funkcija odstrani in vrne vrhnji element sklada (vrne kazalec na začetek besede).

```
char* pop(struct Stack *s) {
    if (s->stackPointer >= 0) {
        int stari = s->stackPointer;
        s->stackPointer = s->stackPointer - 1;
        return s->stack[stari];
    } else {
        printf("Sklad je prazen. Vrnjen bo NULL znak.\n");
        return '\0';
    }
}
```

6. Funkcija vrne skladovni kazalec

```
int top(struct Stack *s)
{
    return s->stackPointer;
}
```

7. Funkcija vrne vrednost vrhnjega elementa.

```
char* topElement(struct Stack *s)
{
    return s->stack[top(s)];
}
```

8. Funkcija izračun se pokliče, kadar je zaznan znak za operacije (+, -, *, / ali %). Najprej odstrani znak za operacije iz sklada, nato odstrani najprej drugo število in nato še prvo ter ju pretvori v int vrednost, da se bo lahko izračunalo. Za tem se z uporabo pogojev ugotovi, za katero operacijo gre. Števili se nato poračunata in shranita v spremenljivko rezultat, nakar se ta vrednost pretvori v niz in shrani na sklad. Na koncu se sprosti prostor začasne spremenljivke rezultata.

```
void izracun(struct Stack *s)
{
    char *operacija = pop(s);
    int drugo = atoi(pop(s));
    int prvo = atoi(pop(s));

    char *rez = (char*) malloc(MAX_STACK_SIZE*sizeof(char));
    int rezultat = 0;

    if (strcmp(operacija,"+") == 0)
    {
        rezultat = prvo + drugo;
        sprintf(rez, "%d", rezultat);
        push(s,rez);
    }
    else if (strcmp(operacija,"-") == 0)
    {
        rezultat = prvo - drugo;
        sprintf(rez, "%d", rezultat);
        push(s,rez);
    }
    else if (strcmp(operacija,"*") == 0)
    {
        rezultat = prvo * drugo;
        sprintf(rez, "%d", rezultat);
        push(s,rez);
    }
    else if (strcmp(operacija,"/") == 0)
    {
        rezultat = prvo / drugo;
        sprintf(rez, "%d", rezultat);
        push(s,rez);
    }
    else if (strcmp(operacija,"%") == 0)
    {
        rezultat = prvo % drugo;
        sprintf(rez, "%d", rezultat);
        push(s,rez);
    }

    free(rez);
}
```

9. Funkcija **main**

Pred neskončno zanko je najprej inicializacija vseh potrebnih spremenljivk in konstant.

Spremenljivka izraz je celoten izraz, ki ga vpišemo npr. 5 9 3 + * (najprej napišemo celoten izraz, nato se stvari izračunajo), spremenljivko nastavimo na prazno vrednost. Spremenljivka vpisan znak je trenuten vpisan znak, ki ga preberemo in serijskega vhoda.

Spremenljivka dolžina izraza predstavlja dolžino vpisanega izraza.

Spremenljivke zacetnoBesedilo, napaka, napakaIzraz in meja pa so izključno namenjene pomoči in lepšemu izpisu ter opozorilu za napake.

Na koncu se vrednost začetnega besedila vpiše na serijski izhod.

```
char izraz[BUFSIZE];
char vpisanZnak;

strcpy(izraz,"");
int dolzinaIzraza = 0;

char *zacetnoBesedilo = {"Kalkulator obrnjenega poljskega zapisa"};
char *napaka = {"Neveljaven vnos - vneses lahko samo stevilke, presledke, +, -, *, /, % in tipko enter za izračun"};
char *napakaIzraz = {"Sklad vsebuje več kot en element, zato izraz, ki je bil vpisan ni bil pravilen - poskusite ponovno"};
char *meja = {"-------------"};

HAL_UART_Transmit(&huart3,zacetnoBesedilo,strlen(zacetnoBesedilo),1000);
HAL_UART_Transmit(&huart3,"\r\n",2,10);
HAL_UART_Transmit(&huart3,meja,strlen(meja),100);
HAL_UART_Transmit(&huart3,"\r\n",2,10);
```

10. Notranjost neskončne while zanke v funkciji main

```
HAL_UART_Receive(&huart3,&vpisanZnak,1,1);
HAL_UART_Transmit(&huart3,&vpisanZnak,1,1);
```

Prejem in izpis posameznega znaka preko UART-a. Znak se zapiše v spremenljivko vpisanZnak.

```
if ((vpisanZnak >= '0' && vpisanZnak <= '9') || vpisanZnak == ' '
        || vpisanZnak == '+' || vpisanZnak == '-' || vpisanZnak == '*' || vpisanZnak == '/' || vpisanZnak == '%')
{
    strncat(izraz, &vpisanZnak, 1);
    dolzinaIzraza++;
}
```

Če vpisan znak ustreza pogoju, da je številka, znak za presledek, plus, minus, krat, deljeno ali modulo se vpisan znak konkatenira, zlepi, doda na konec celotnemu nizu in vrednost dolžine izraza se poveča za 1.

Če prejšnji pogoj ni izpolnjen in je dolžina izraza več kot 0 ter je vpisan znak \r, kar pomeni enter – izračun se izvede naslednji del kode.

Rezervira se dodaten pomnilnik na skladu, prejsnjiZnak se nastavi na presledek – to nam bo služilo, da bo izračun deloval, tudi če bo več presledkov med znaki. Za tem ustvarimo in inicializiramo sklad.

Sledi for zanka, ki se sprehodi znak po znak skozi vpisan izraz. V for zanki so tri pogoji. Če je prvi pogoj izpolnjen pomeni, da smo prišli do presledka in mora it vsa vsebina, ki je bila prej konkatenirana v spremenljivki trenuten na sklad kot eno število. Če je drugi pogoj izpolnjen to pomeni operacijo, zato se kliče funkcija izračun, ki izračuna in shrani rezultat na vrh sklada. Če je tretji pogoj izpolnjen pomeni samo, da gre za številko – to številko samo konkateniramo na konec izraza trenuten, ki služi kot pomožna spremenljivka (s tem v bistvu sestavljamo številko dokler ne pridemo do presledka).

```
else if (vpisanZnak == '\r' && strlen(izraz) > 0)
{
    char *trenuten = (char*) malloc(MAX_STACK_SIZE*sizeof(char));

    char prejsnjiZnak = ' ';

    struct Stack sklad;
    initializeStack(&sklad);

    for (int i=0; i<dolzinaIzraza; i++)
    {
        if (prejsnjiZnak != ' ' && izraz[i] == ' ')
        {
            push(&sklad,trenuten);
            trenuten[0] = '\0';
            prejsnjiZnak = ' ';
        }
        if (izraz[i] == '+' || izraz[i] == '-' || izraz[i] == '*' || izraz[i] == '/' || izraz[i] == '%')
        {
            strncat(trenuten, &(izraz[i]), 1);
            push(&sklad,trenuten);
            trenuten[0] = '\0';
            izracun(&sklad);
        }
        if (izraz[i] >= '0' && izraz[i] <= '9')
        {
            strncat(trenuten, &(izraz[i]), 1);
            prejsnjiZnak = izraz[i];
        }
    }
```

Ko izračunamo vse preverimo, če ima stack pointer vrednost 0 – to pomeni, da je bil vpisan izraz pravilen in da je na skladu samo rezultat izraza in nič drugega. Če je pogoj izpolnjen izpišemo rezultat, če ni pa izpišemo napako.

```
if (top(&sklad) == 0)
{
    char *rezultat = (char*) malloc(MAX_STACK_SIZE*sizeof(char));
    strcpy(rezultat,topElement(&sklad));

    HAL_UART_Transmit(&huart3,"\r\n",2,10);
    HAL_UART_Transmit(&huart3,rezultat,sizeof(rezultat),100);
    HAL_UART_Transmit(&huart3,"\r\n",2,10);

    free(rezultat);
}
else
{
    HAL_UART_Transmit(&huart3,"\r\n",2,10);
    HAL_UART_Transmit(&huart3,napakaIzraz,strlen(napakaIzraz),1000);
    HAL_UART_Transmit(&huart3,"\r\n",2,10);
}
```

Ko se to konča izpišemo črtice – mejo, saj naslednji del bo spet nov izraz, ki je neodvisen od tega. Zato sklad uničimo z pomočjo funkcije destroyStack, spremenljivko trenuten sprostimo, vpisan izraz nastavimo na prazen niz in dolžino izraza na 0 (s tem smo spet na začetku).

```
HAL_UART_Transmit(&huart3,meja,strlen(meja),100);
HAL_UART_Transmit(&huart3,"\r\n",2,10);

free(trenuten);

destroyStack(&sklad);

strcpy(izraz,"");
dolzinaIzraza = 0;
```

Če prejšnja dva pogoja nista bila izpolnjena, torej ni bil vnesen pravilen znak – npr. če je bila vnesena črka se izpolni zadnji zunanji pogoj, ki izpiše napako, da moramo vnesti pravilen znak in ne npr. črke.

```
else if (vpisanZnak != '\0')
{
    HAL_UART_Transmit(&huart3,"\r\n",2,10);
    HAL_UART_Transmit(&huart3,napaka,strlen(napaka),1000);
    HAL_UART_Transmit(&huart3,"\r\n",2,10);
    HAL_UART_Transmit(&huart3,meja,strlen(meja),100);
    HAL_UART_Transmit(&huart3,"\r\n",2,10);
}
```

Primer delovanja