Wstęp do programowania w języku C

Kalkulator z użyciem GTK+ 3.0
Abstrakcyjne typy danych w C

Marek Piotrów - Wykład 11

5 stycznia 2022

Kalkulator

ZADANIE: Napisać program kalkulator z okienkowym interfejsem, który:

- Wykonuje obliczenia na liczbach rzeczywistych z uwzględnieniem 4 podstawowych działań: +, -, * i /.
- Uwzględnia priorytety operatorów, tzn. mnożenie i dzielenie mają wyższy priorytet niż dodawanie i odejmowanie.
- Pozwala stosować nawiasy.
- Sygnalizuje błędy we wprowadzonym wyrażeniu.

Kalkulator

ROZWIĄZANIE:

- Użyć biblioteki GTK+ do zdefiniowania i obsługi interfejsu okienkowego. Będzie to pierwszy moduł rozwiązania.
- Napisać kalkulator z interfejsem tekstowym jako drugi moduł rozwiazania tak, aby czytał wyrażenie z tablicy znakowej i udostępniał drugiemu modułowi funkcję:
  `double oblicz(char *tekst_wyrazenia);`
- Do sygnalizacji błędów drugi moduł użyje funkcji:
  `void pokazBlad(char *blad);` z pierwszego modułu.

Interfejs w GTK (gtk-kalk.c) I

```
#include <string.h>
#include <math.h>
#include <gtk/gtk.h>
// kompilacja: gcc -std=c11 -Wall -Wextra -Wno-unused-parameter -Werror -o kalk gtk-kalk.c kalkulator.c ‘pkg-config
gtk+-3.0 --cflags --libs‘ -lm

#define MAKS_DL_WYR 100

void pokazBlad(gchar *komunikat); // funkcja dla modułu kalkulator.c
extern double oblicz(const char *tekst_wyrazenia); // funkcja z modułu kalkulator.c

static void oblicz_wyrazenie(GtkWidget *widget, GtkWidget *text);
static void dodaj_do_text   (GtkWidget *widget, gchar *input);

static GtkWidget *text, *window;

int main(int argc,char *argv[])
{
   gtk_init(&argc, &argv);

   window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
   gtk_window_set_title(GTK_WINDOW(window),"kalkulator");
   gtk_window_set_position(GTK_WINDOW(window),GTK_WIN_POS_CENTER);
   g_signal_connect(G_OBJECT(window), "destroy",G_CALLBACK(gtk_main_quit), NULL);
   gtk_container_set_border_width(GTK_CONTAINER(window), 10);

   GtkWidget *grid = gtk_grid_new();
```

Interfejs w GTK (gtk-kalk.c) II

```
gtk_grid_set_row_spacing(GTK_GRID(grid), 3);
gtk_grid_set_row_homogeneous(GTK_GRID(grid), TRUE);
gtk_grid_set_column_spacing(GTK_GRID(grid), 3);
gtk_grid_set_column_homogeneous(GTK_GRID(grid), TRUE);
gtk_container_add(GTK_CONTAINER(window), grid);

text = gtk_entry_new();                    // do wprowadzania wyrazenia
gtk_entry_set_max_length(GTK_ENTRY(text), MAKS_DL_WYR);
gtk_entry_set_alignment(GTK_ENTRY(text), 1); // wyrównanie do prawej strony
g_signal_connect(G_OBJECT(text), "activate",G_CALLBACK(oblicz_wyrazenie),
   (gpointer) text);
gtk_entry_set_text(GTK_ENTRY(text), "");
gtk_grid_attach(GTK_GRID(grid), text, 0, 0, 4, 1);

struct przycisk {
   gchar *opis, *wyjscie;
   gint posX,lenX,posY,lenY;
} tab[] = {
   {"CLR","\n",0,1,1,1}, {"(","(",1,1,1,1}, {")",")",2,1,1,1}, {"/","/",3,1,1,1},
     {"7","7",0,1,2,1}, {"8","8",1,1,2,1}, {"9","9",2,1,2,1}, {"*","*",3,1,2,1},
     {"4","4",0,1,3,1}, {"5","5",1,1,3,1}, {"6","6",2,1,3,1}, {"-","-",3,1,3,1},
     {"1","1",0,1,4,1}, {"2","2",1,1,4,1}, {"3","3",2,1,4,1}, {"+","+",3,1,4,2},
     {"0","0",0,1,5,1}, {".",".",1,1,5,1}, {"=","=",2,1,5,1}
    };
for (unsigned i = 0; i < sizeof(tab)/sizeof(struct przycisk); i++) {
   GtkWidget *button = gtk_button_new_with_label(tab[i].opis);
```

Interfejs w GTK (gtk-kalk.c) III

```
        g_signal_connect(G_OBJECT(button), "clicked",G_CALLBACK(dodaj_do_text), (gpointer) tab[i].wyjscie);
        gtk_grid_attach(GTK_GRID(grid), button, tab[i].posX, tab[i].posY, tab[i].lenX, tab[i].lenY);
    }

    gtk_widget_show_all(window);
    gtk_main();
    return 0;
}

void pokazBlad(char *str)
{
    GtkWidget *dialog = gtk_message_dialog_new(GTK_WINDOW(window),
        GTK_DIALOG_DESTROY_WITH_PARENT, GTK_MESSAGE_ERROR, GTK_BUTTONS_CLOSE, "%s", str);
    gtk_dialog_run(GTK_DIALOG(dialog));
    gtk_widget_destroy(dialog);
}

static void oblicz_wyrazenie( GtkWidget *widget,GtkWidget *text)
{
    gchar wejscie[MAKS_DL_WYR+2];

    strcpy(wejscie, gtk_entry_get_text(GTK_ENTRY(text)));
    if (wejscie[strlen(wejscie)-1] != '=') strcat(wejscie,"=");

    double wynik=oblicz(wejscie);
    if (isnan(wynik)) return;
```

Interfejs w GTK (gtk-kalk.c) IV

```
    sprintf(wejscie,"%.8g",wynik);
    gtk_entry_set_text(GTK_ENTRY(text),wejscie);
    gtk_editable_select_region(GTK_EDITABLE(text),0,gtk_entry_get_text_length(GTK_ENTRY(text)));
}

static void dodaj_do_text(GtkWidget *widget, gchar *input)
{
    if(strcmp(input, "\n") == 0)
        gtk_entry_set_text(GTK_ENTRY(text), "");
    else {
        gint tmp_pos = gtk_entry_get_text_length(GTK_ENTRY(text));

        gtk_editable_insert_text(GTK_EDITABLE(text), input, -1, &tmp_pos);
        if (strcmp(input, "=") == 0)
            oblicz_wyrazenie(widget, text);
    }
}
```

Moduł kalkulator.c I

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>

/*********   kalkulator.c: kalkulator dla wyrazen rzeczywistych   **********
 * Moduł udostępnia funkcję double oblicz(char *tekst_wyrazenia), która
 * czyta z tekst_wyrazenia zapisane z uzyciem nawiasow i czterech
 * podstawowych dzialan wyrazenie (zakończone znakiem =) i oblicza je
 * rekurencyjnie. Wartość wyrażenia zwracana jest jako wynik obliczenia.
 ******************************************************************************/

#define LICZBA '0'

/***************** PROTOTYPY FUNKCJI ********************/
static int czytaj_znak(char **inp); // czyta kolejny widoczny znak z wejścia
static void zwroc_znak(int z, char **inp); // oddaje znak na wejście

static double czytaj_liczbe(char **inp); // czyta kolejną liczbe z wejścia

static double wyrazenie(char **inp); // analizuje składnie wyrażenia i wylicza jego wartość
static double skladnik(char **inp);  // analizuje składnie składnika i wylicza jego wartość
static double czynnik(char **inp);   // analizuje składnie czynnika i wylicza jego wartość

extern void pokazBlad(char *blad); // funkcja z zewnętrznego modułu do sygnalizacji błędu

/***************** DEFINICJE FUNKCJI ********************/
```

Moduł kalkulator.c II

```
double oblicz(char *wejscie)
{
   int z;
   double wyn;
   char blad[100];
   char *inptr=wejscie;

   while ((z=czytaj_znak(&inptr)) != EOF) {
    zwroc_znak(z,&inptr);
     wyn=wyrazenie(&inptr);
     if ((z=czytaj_znak(&inptr)) == '=')
        return wyn;
     else {
        sprintf(blad,"Nieoczekiwany znak na koncu wyrazenia %c\n",z);
        pokazBlad(blad);
        return nan("BLAD");
     }
   }
   return 0;
}

static void zwroc_znak(int z, char **inp)
{
  if (z != EOF && z != LICZBA)
     --*inp;
}
```

Moduł kalkulator.c III

```
static int czytaj_znak(char **inp)
{
   int z;

   if (**inp == '\0') return EOF;
   while ((z=*(*inp)++) != '\0' && isspace(z)) ;
   if (isdigit(z) || z == '.' || z == ',') {
      zwroc_znak(z, inp);
      return LICZBA;
   }
   return z == 0 ? EOF : z;
}

static double czytaj_liczbe(char **inp)
{
   int z;
   double n=0.0, pot10=1.0;

   while ((z=*(*inp)++) != '\0' && isdigit(z))
      n=10.0 * n + (z-'0');
   if (z == '.' || z == ',')
      while ((z=*(*inp)++) != '\0' && isdigit(z)) {
         n=10.0 * n + (z-'0');
         pot10*=10.0;
      }
```

Moduł kalkulator.c IV

```
    zwroc_znak(z == 0 ? EOF : z, inp);
    return n/pot10;
}

static double wyrazenie(char **inp)
{
    int z;
    double wyn, x2;

    if ((z=czytaj_znak(inp)) != '-' && z != '+')
      zwroc_znak(z, inp);
    wyn=skladnik(inp);
    if (z == '-') wyn=-wyn;
    while ((z=czytaj_znak(inp)) == '+' || z == '-') {
        x2=skladnik(inp);
        wyn=(z == '+' ? wyn+x2 : wyn-x2);
    }
    zwroc_znak(z, inp);
    return wyn;
}

static double skladnik(char **inp)
{
    int z;
    double wyn,x2;
```

Moduł kalkulator.c V

```
    wyn=czynnik(inp);
    while ((z=czytaj_znak(inp)) == '*' || z == '/') {
        x2=czynnik(inp);
        wyn=(z == '*' ? wyn*x2 : wyn/x2);
    }
    zwroc_znak(z, inp);
    return wyn;
}

static double czynnik(char **inp)
{
    int z;
    double wyn;
    char blad[100];

    if ((z=czytaj_znak(inp)) == LICZBA)
        return czytaj_liczbe(inp);
    else if (z == '(') {
        wyn=wyrazenie(inp);
        if ((z=czytaj_znak(inp)) == ')')
            return wyn;
        else {
            sprintf(blad,"BLAD: oczekiwano ')', a wystapil znak: '%c'\n",z);
            pokazBlad(blad);
            return nan("BLAD");
        }
```

Moduł kalkulator.c VI

```
    }
    else {
        sprintf(blad,"BLAD: oczekiwano liczby lub '(', a wystapil znak: '%c'\n",z);
        pokazBlad(blad);
      return nan("BLAD");
    }
}
```

Abstrakcyjne typy danych (ATD)

- ATD definiują klasę struktur danych o podobnym zachowaniu.
- ATD definiowane są pośrednio za pomocą operacji, które można na nich wykonywać i oczekiwanych efektach ich wykonania.
- W językach programowania ATD dostępne są za pomocą interfejsu modułu.
- ATD mogą mieć różne implementacje. Chcemy, aby szczegóły implementacji były niedostępne dla osoby korzystającej z ATD.

Definicja interfejsu stosu (stos.h)

```
#ifndef STACK_ADT
#define STACK_ADT

#include <stdbool.h>

typedef struct str_stack *StrStack;

StrStack strStackCreate(size_t minSize);
void strStackDestroy(StrStack s);
bool strStackEmpty(StrStack s);
int strStackPush(StrStack s,char *str);
char *strStackTop(StrStack s);
char *strStackPop(StrStack s);

#endif
```

Użycie abstrakcyjnego typu danych (stostest.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include "stos.h"

int main(int argc,char *argv[])
{
    StrStack stack = strStackCreate(0);

    for (int i=1; i < argc; ++i)
        if (strStackPush(stack,argv[i]) != 0)
            return EXIT_FAILURE;

    while (!strStackEmpty(stack))
        printf("%s ",strStackPop(stack));
    putchar('\n');

    strStackDestroy(stack);
    return EXIT_SUCCESS;
}
```

Implementacja w rozszerzalnej tablicy (stostab.c) I

```c
#include <stdlib.h>
#include "stos.h"

#define MY_MIN_SIZE 10

struct str_stack {
    size_t actSize, minSize;
    char **top;
    char **strTab;
} ;

StrStack strStackCreate(size_t minSize)
{
    if (minSize < MY_MIN_SIZE) minSize=MY_MIN_SIZE;
    StrStack s=malloc(sizeof(struct str_stack));
    if (s == NULL) return NULL;
    s->strTab=malloc(minSize*sizeof(char *));
    if (s->strTab == NULL) { free(s); return NULL; }
    s->actSize = s->minSize = minSize;
    s->top = s->strTab;
    return s;
}

void strStackDestroy(StrStack s)
{
    if (s == NULL) return;
    free(s->strTab);
```

Implementacja w rozszerzalnej tablicy (stostab.c) II

```
    free(s);
}

bool strStackEmpty(StrStack s)
{
    return s == NULL || s->top == s->strTab;
}

int strStackPush(StrStack s,char *str)
{
    if (s == NULL) return 1;
    if (s->top == s->strTab + s->actSize) {
        char **newTab=realloc(s->strTab,(s->actSize+s->minSize)*sizeof(char *));
        if (newTab == NULL) return 2;
        s->strTab = newTab;
        s->top = newTab+s->actSize;
        s->actSize += s->minSize;
    }
    *s->top++ = str;
    return 0;
}

char *strStackTop(StrStack s)
{
    if (s == NULL || s->top == s->strTab) return NULL;
    return s->top[-1];
```

Implementacja w rozszerzalnej tablicy (stostab.c) III

```
}

char *strStackPop(StrStack s)
{
    if (s == NULL || s->top == s->strTab) return NULL;
    return *--s->top;
}
```

Implementacja w postaci listy tablic (stoslst.c) I

```
#include <stdlib.h>
#include "stos.h"

#define MY_MIN_SIZE 10

struct str_lst_el {
   struct str_lst_el *next;
   char *strTab[]; // the allocated size will be minSize
} ;

struct str_stack {
   size_t minSize;
   char **top;
   struct str_lst_el *last;
}  ;

StrStack strStackCreate(size_t minSize)
{
   if (minSize < MY_MIN_SIZE) minSize=MY_MIN_SIZE;
   StrStack s=malloc(sizeof(struct str_stack));
   if (s == NULL) return NULL;
   s->last=malloc(sizeof(struct str_lst_el) + minSize*sizeof(char *));
   if (s->last == NULL) { free(s); return NULL; }
   s->last->next = NULL;
   s->minSize = minSize;
   s->top = s->last->strTab;
   return s;
```

Implementacja w postaci listy tablic (stoslst.c) II

```
}

void strStackDestroy(StrStack s)
{
   if (s == NULL) return;
   for (struct str_lst_el *p=s->last, *q; p != NULL; p=q) {
      q = p->next;
      free(p);
   }
   free(s);
}

bool strStackEmpty(StrStack s)
{
   return s == NULL || (s->last->next == NULL && s->top == s->last->strTab);
}

int strStackPush(StrStack s,char *str)
{
   if (s == NULL) return 1;
   if (s->top == s->last->strTab + s->minSize) {
      struct str_lst_el *newEl=malloc(sizeof(struct str_lst_el) +
                                      s->minSize*sizeof(char *));
      if (newEl == NULL) return 2;
      newEl->next = s->last;
      s->last = newEl;
```

Implementacja w postaci listy tablic (stoslst.c) III

```
        s->top = newEl->strTab;
    }
    *s->top++ = str;
    return 0;
}

char *strStackTop(StrStack s)
{
    if (strStackEmpty(s)) return NULL;
    if (s->top == s->last->strTab)
      return s->last->next->strTab[s->minSize-1];
    return s->top[-1];
}

char *strStackPop(StrStack s)
{
    if (strStackEmpty(s)) return NULL;
    if (s->top == s->last->strTab) {
        struct str_lst_el *p=s->last;
        s->last = s->last->next;
        s->top = s->last->strTab + s->minSize;
        free(p);
    }
    return *--s->top;
}
```

Implementacja w postaci listy tablic (stoslst.c) IV

Funkcje ze zmienną liczba argumentów I

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>

double suma(int n,...)
{
   va_list ap;
   double s=0.0;

   va_start(ap,n);
   for (int i=0; i < n; ++i) {
      double x=va_arg(ap,double);
      s+=x;
   }
   va_end(ap);
   return s;
}

int main(int argc, char *argv[])
{
   double x=0.0;
   switch(argc) {
   case 1:   break;
   case 2:   x=atof(argv[1]); break;
   case 3:   x=suma(2,atof(argv[1]),atof(argv[2])); break;
   case 4:   x=suma(3,atof(argv[1]),atof(argv[2]),atof(argv[3])); break;
   default: x=suma(4,atof(argv[1]),atof(argv[2]),atof(argv[3]),atof(argv[4])); break;
```

Funkcje ze zmienną liczba argumentów II

```
    }
    printf("Suma = : %.2lf\n",x);
    return EXIT_SUCCESS;
}
```