

Wstęp do programowania w języku C

Marek Piotrów - Wykład 8
Struktury dynamiczne. Efektywne użycie rekursji.

26 listopada 2014

Operacje na stercie

- ▶ Operacje na stercie nie są częścią definicji języka C (pojawiają się dopiero w definicji C++), ale są dostępne w standardowej bibliotece C `stdlib`.

- ▶ Do przydziału pamięci ze sterty używa się jednej z funkcji:

```
void *malloc(size_t size);  
void *calloc(size_t nmemb, size_t size);
```

- ▶ Do zwolnienia przydzielonej pamięci służy funkcja:

```
void free(void *ptr);
```

- ▶ Do zmiany rozmiaru przydzielonej pamięci można użyć funkcji:

```
void *realloc(void *ptr, size_t size);
```

Stos jako abstrakcyjna struktura danych

Stos jest abstrakcyjną strukturą danych (kolekcją), w której:

- ▶ przechowywane są w uporządkowany sposób obiekty tego samego typu;
- ▶ elementy stosu są uporządkowane według czasu dołączenia do kolekcji - na wierzchołku stosu jest element najnowszy;
- ▶ dostęp do elementów stosu jest tylko przez jego wierzchołek - można z niego pobrać element lub położyć na nim nowy.

ZADANIE: Zaimplementować stos w postaci jednostronnej listy łączonej.

Definicja interfejsu stosu - stos.h

```
#ifndef MOJ_STOS
#define MOJ_STOS

#include <stdlib.h>
#include <stdbool.h>
#include <math.h>

#define TYP_INFO    double
#define TYP_NULL    NAN

typedef struct stack *StackPtr;

void init(StackPtr *stck);
void clear(StackPtr *stck);
bool isempty(StackPtr stck);
bool isfull(StackPtr stck);
bool push(StackPtr *stck, TYP_INFO info);
TYP_INFO top(StackPtr stck);
TYP_INFO pop(StackPtr *stck);

#endif
```

Implementacja stosu jako listy jednostronnej - stos.c

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

struct stack {
    TYP_INFO info;
    struct stack *next;
};

void init(StackPtr *stck)
{
    *stck=NULL;
}

void clear(StackPtr *stck)
{
    for (StackPtr p=*stck, q; p != NULL; p=q) {
        q=p->next;
        free(p);
    }
    *stck=NULL;
}

bool isempty(StackPtr stck)
{
    return (stck == NULL);
}

bool isfull(StackPtr stck)
{
    return false;
}
```

Struktury dynamiczne - stos.c (cd.)

```
bool push(StackPtr *stck, TYP_INFO info)
{
    StackPtr p;
    if ((p=(StackPtr)malloc(sizeof(struct stack))) == NULL)
        return true;
    else {
        p->info=info;
        p->next=*stck;
        *stck=p;
        return false;
    }
}

TYP_INFO top(StackPtr stck)
{
    return (stck == NULL ? TYP_NULL : stck->info);
}

TYP_INFO pop(StackPtr *stck)
{
    TYP_INFO info;
    StackPtr p;
    if (*stck == NULL)
        return TYP_NULL;
    else {
        info=(*stck)->info;
        p=*stck;
        *stck=(*stck)->next;
        free(p);
        return info;
    }
}
```

Kolejka jako abstrakcyjna struktura danych

- Kolejka jest abstrakcyjną strukturą danych (kolekcją), w której:
- ▶ przechowywane są w uporządkowany sposób obiekty tego samego typu;
 - ▶ elementy kolejki są uporządkowane według czasu dołączenia do kolekcji - na początku kolejki jest element najstarszy, a na końcu - najnowszy;
 - ▶ dostęp do elementów kolejki jest tylko przez jej skrajne elementy - można z jej początku pobrać (najstarszy) element lub dołączyć na jej końcu nowy obiekt.

ZADANIE: Zaimplementować kolejkę w postaci jednostronnej listy łączonej.

Definicja interfejsu kolejki - kolejka.h

```
#include <stdlib.h>

#define TYP_INFO  char*
#define TYP_NULL  NULL

struct e_listy {
    TYP_INFO info;
    struct e_listy *nast;
};

typedef struct kol {
    struct e_listy *pierwszy;
    struct e_listy *ostatni;
} Kolejka;

Kolejka *nowa(void);
int pusta(Kolejka *kol);
int do_kolejki(Kolejka *kol, TYP_INFO info);
TYP_INFO z_kolejki(Kolejka *kol);
```


Implementacja kolejki jako listy łączonej - kolejka.c

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

Kolejka *nowa(void)
{
    Kolejka *p;

    if ((p=(Kolejka *)malloc(sizeof(Kolejka))) == NULL)
        return NULL;
    else {
        p->pierwszy=p->ostatni=NULL;
        return p;
    }
}

int pusta(Kolejka *kol)
{
    return (kol->pierwszy == NULL);
}
```

```
int do_kolejki(Kolejka *kol, TYP_INFO info)
{
    struct e_listy *p;

    if ((p=(struct e_listy *)malloc(sizeof(struct e_listy))) == NULL)
        return 1;
    else {
        p->info=info;
        p->nast=NULL;
        if (kol->pierwszy == NULL)
            kol->pierwszy=kol->ostatni=p;
        else
            kol->ostatni=kol->ostatni->nast=p;
        return 0;
    }
}
```

```
TYP_INFO z_kolejki(Kolejka *kol)
{
    struct e_listy *p;
    TYP_INFO info;

    if ((p=kol->pierwszy) == NULL)
        return TYP_NULL;
    else {
        if ((kol->pierwszy=p->nast) == NULL)
            kol->ostatni=NULL;
        info=p->info;
        free(p);
        return info;
    }
}
```

Drzewa binarne - sortowanie liczb

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

typedef struct e_drzewa *Wsk_drzewa;
typedef struct e_drzewa {
    double liczba;
    int ile_razy;
    Wsk_drzewa lewy;
    Wsk_drzewa prawy;
} Wezel_drzewa;

static Wsk_drzewa dopisz_liczbe(Wsk_drzewa p, double dana);
static int wypisz_drzewo(Wsk_drzewa p, int n);
static void usun_drzewo(Wsk_drzewa p);

int main(void)
{
    Wsk_drzewa Korzen=NULL;
    double dana;

    while (scanf("%lf",&dana) == 1)
        Korzen=dopisz_liczbe(Korzen,dana);
    wypisz_drzewo(Korzen,0);
    usun_drzewo(Korzen);
    putchar('\n');
    return 0;
}
```

Sortowanie liczb - funkcja dopisz_liczbe

```
static Wsk_drzewa dopisz_liczbe(Wsk_drzewa korzen,double dana)
{
    Wsk_drzewa *pop,akt;

    for (pop=&korzen,akt=korzen; akt != NULL; )
        if (dana < akt->liczba)
            pop=&akt->lewy, akt=akt->lewy;
        else if (dana > akt->liczba)
            pop=&akt->prawy, akt=akt->prawy;
        else {
            ++akt->ile_razy;
            return korzen;
        }
    if ((akt=(Wsk_drzewa)malloc(sizeof(Wezel_drzewa))) == NULL) {
        fprintf(stderr,"Brak pamieci w stercie dla %lf\n",dana);
        exit(1);
    }
    akt->liczba=dana;
    akt->ile_razy=1;
    akt->lewy=akt->prawy=NULL;
    *pop=akt;
    return korzen;
}
```

Sortowanie liczb - rekurencyjna funkcja dopisz_liczbe

```
/****** wersja rekurencyjna *****/  
static Wsk_drzewa dopisz_liczbe(Wsk_drzewa p, double dana)  
{  
    if (p == NULL) {  
        if ((p=(Wsk_drzewa)malloc(sizeof(Wezel_drzewa))) == NULL) {  
            fprintf(stderr, "Brak pamieci w stercie dla %lf\n", dana);  
            exit(1);  
        }  
        p->liczba=dana;  
        p->ile_razy=1;  
        p->lewy=p->prawy=NULL;  
    } else if (dana < p->liczba)  
        p->lewy=dopisz_liczbe(p->lewy, dana);  
    else if (dana > p->liczba)  
        p->prawy=dopisz_liczbe(p->prawy, dana);  
    else  
        ++p->ile_razy;  
    return p;  
}
```

Sortowanie liczb - funkcje wypisz_drzewo i usun_drzewo

```
static int wypisz_drzewo(Wsk_drzewa p,int n)
{
    if (p != NULL) {
        n=wypisz_drzewo(p->lewy,n);
        for (int i=p->ile_razy; i > 0; --i)
            printf(" %21f%c",p->liczba,(++n%8 ? ' ' : '\n'));
        n=wypisz_drzewo(p->prawy,n);
    }
    return n;
}

static void usun_drzewo(Wsk_drzewa p)
{
    if (p != NULL) {
        usun_drzewo(p->lewy);
        usun_drzewo(p->prawy);
        free(p);
    }
}
```

Rekurencyjny kalkulator dla liczb rzeczywistych I

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

/***** kalkulator.c: kalkulator dla wyrazen rzeczywistych *****/
* Program czyta ze standardowego wejscia zapisane z uzyciem nawiasow i
* czterech podstawowych dzialan wyrazenie i oblicza je rekurencyjnie.
* Wynik wyswietlany jest po znaku =.
*****/

#define LICZBA '0'

/***** PROTOTYPY FUNKCJI *****/

static void zwroc_znak(int z);
static double czytaj_liczbe(void);
static int czytaj_znak(void);

double wyrazenie(void);
static double skladnik(void);
static double czynnik(void);
```

Rekurencyjny kalkulator dla liczb rzeczywistych

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

int main(void)
{
    int z;
    double wyn;

    while ((z=czytaj_znak()) != EOF) {
        zwroc_znak(z);
        wyn=wyrazenie();
        if ((z=czytaj_znak()) == '=')
            printf("WYNIK = %.8g\n",wyn);
        else {
            printf("BLAD: nieoczekiwany znak: '%c'\n",z);
            return 1;
        }
    }
    return 0;
}

static void zwroc_znak(int z)
{
    if (z != EOF && z != LICZBA)
        ungetc(z,stdin);
}
```


Rekurencyjny kalkulator - czytanie danych

```
static int czytaj_znak(void)
{
    int z;

    if (feof(stdin)) return EOF;
    while ((z=getchar()) != EOF && isspace(z)) ;
    if (isdigit(z) || z == '.' || z == '-') {
        ungetc(z, stdin);
        return LICZBA;
    }
    return z;
}

static double czytaj_liczbe(void)
{
    int z;
    double n=0.0, pot10=1.0;

    while ((z=getchar()) != EOF && isdigit(z))
        n=10.0 * n + (z-'0');
    if (z == '.' || z == '-')
        while ((z=getchar()) != EOF && isdigit(z)) {
            n=10.0 * n + (z-'0');
            pot10*=10.0;
        }
    zwroc_znak(z);
    return n/pot10;
}
```

Rekurencyjny kalkulator - analiza wyrażenia

```
static double wyrażenie(void)
{
    int z;
    double wyn, x2;

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

Rekurencyjny kalkulator - analiza składnika

```
static double skladnik(void)
{
    int z;
    double wyn,x2;

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

Rekurencyjny kalkulator - analiza czynnika

```
static double czynnik(void)
{
    int z;
    double wyn;

    if ((z=czytaj_znak()) == LICZBA)
        return czytaj_liczbe();
    else if (z == ' ( '){
        wyn=wyrazenie();
        if ((z=czytaj_znak()) == ' ) '){
            return wyn;
        }
        else {
            printf("BLAD: oczekiwano ' ) ', a wystapil znak: '%c'\n",z);
            exit(1);
        }
    }
    else {
        printf("BLAD: oczekiwano liczby lub ' ( ', a wystapil znak: '%c'\n",z);
        exit(1);
    }
}
```