

Wstęp do programowania w języku C

Struktury dynamiczne. Operacje na plikach.

Marek Piotrów - Wykład 8

24 listopada 2020

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);
    }
}
```

System plików - operacje na plikach

- ▶ System plików jest częścią systemu operacyjnego. System plików umożliwia przechowywanie informacji na różnych urządzeniach (pamięciach zewnętrznych) w jednolity sposób.
- ▶ Różne systemy operacyjne mają różne systemy plików. W większości pliki umieszczane są w hierarchicznej strukturze katalogów i są opisane nazwą i innymi atrybutami (np. czasami utworzenia i ostatniej modyfikacji, prawami użytkowników do dostępu do zawartości pliku).
- ▶ Biblioteka standardowa C pozwala na (w miarę) jednolity dostęp do systemu plików niezależnie od systemu operacyjnego.

System plików - otwarcie i zamknięcie pliku

- ▶ Każdy dostęp do nazwanego pliku z programu w C musi rozpocząć się od operacji *otwarcia pliku*:
`FILE *fopen(const char *path, const char *mode)`
- ▶ Otwarcie pliku może zakończyć się sukcesem lub porażką.
- ▶ Jeśli kończy się sukcesem, to `fopen` zwraca wskaźnik do standardowej struktury `FILE` opisującej otwarty plik.
- ▶ Jeśli kończy się porażką (plik nie istnieje lub nie można go utworzyć, nie mamy dostatecznych praw dostępu do pliku lub katalogu, itp.), to `fopen` zwraca `NULL`.
- ▶ Każdy otwarty plik musi być zamknięty operacją *zamknięcia pliku* `int fclose(FILE *fp)` Funkcja ta zwraca 0 po poprawnym zamknięciu pliku.

System plików - operacje pisania i czytania

- ▶ Wskaźnik zwracany przez `fopen` jest potrzebny do operacji czytania pliku i pisania do pliku. Jest on pierwszym (lub ostatnim) argumentem poszczególnych operacji.
- ▶ Do sformatowanego czytania z plików tekstowych mamy funkcję: `int fscanf(FILE *stream, const char *format, ...)`
- ▶ Do sformatowanego pisania do plików tekstowych mamy funkcję: `int fprintf(FILE *stream, const char *format, ...)`
- ▶ Pojedyncze znaki możemy czytać używając `int getc(FILE *stream)` oraz pisać za pomocą `int putc(int c, FILE *stream)`.

Kopiowanie pliku I

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

/***** KOPIOWANIE PLIKU *****/

void kopiuj(FILE *zfp, FILE *dofp);

int main(int argc, char *argv[])
{
    FILE *zfp, *dofp;
    char *nazwa_z=NULL;
    char *nazwa_do=NULL;
    int i, binarnie=0;

    if (argc == 1) {
        fprintf(stderr, "\nUzycie: %s <plik zrodlowy> <plik docelowy> [-b]\n",
            argv[0]);
        fputs("\n    -b oznacza kopiowanie pliku binarnego\n\n", stderr);
        exit(1);
    }
}
```

Kopiowanie pliku II

```
for (i=1; i < argc; ++i)
    if (argv[i][0] == '-' )
        if (tolower(argv[i][1]) == 'b' && argv[1][2] == '\0')
            binarnie=1;
        else {
            fprintf(stderr, "%s: nieznana opcja: %s\n", argv[0], argv[i]);
            exit(2);
        }
    else if (nazwa_z == NULL)
        nazwa_z=argv[i];
    else if (nazwa_do == NULL)
        nazwa_do=argv[i];

if (nazwa_z == NULL)
    zfp=stdin;
else if ((zfp=fopen(nazwa_z, (binarnie? "rb" : "r"))) == NULL) {
    fprintf(stderr, "%s: blad otwarcia pliku: %s\n", argv[0], nazwa_z);
    exit(2);
}
if (nazwa_do == NULL)
    dofp=stdout;
else if ((dofp=fopen(nazwa_do, (binarnie? "wb" : "w"))) == NULL) {
    fprintf(stderr, "%s: blad otwarcia pliku: %s\n", argv[0], nazwa_do);
    exit(2);
}
kopiuje(zfp, dofp);
```

Kopiowanie pliku III

Kopiowanie pliku IV

```
if (ferror(zfp)) {  
    fprintf(stderr,"%s: blad odczytu pliku: %s\n",argv[0],  
            (nazwa_z != NULL ? nazwa_z : "standardowe wejście"));  
    exit(2);  
}  
if (ferror(dofp)) {  
    fprintf(stderr,"%s: blad zapisu pliku: %s\n",argv[0],  
            (nazwa_do != NULL ? nazwa_do : "standardowe wyjście"));  
    exit(2);  
}  
fclose(zfp);  
fclose(dofp);  
return 0;  
}  
  
void kopiuuj(FILE *zfp,FILE *dofp)  
{  
    int c;  
  
    while ((c=getc(zfp)) != EOF)  
        putc(c,dofp);  
}
```

System plików - operacje na plikach binarnych

- ▶ W plikach są też przechowywane dane binarne np. pliki spakowane, obrazy, itp.

- ▶ Do odczytu danych binarnych używamy najczęściej operacji:

```
size_t fread(void *ptr, size_t size, size_t  
nmemb, FILE *stream)
```

- ▶ Do zapisu danych binarnych używamy najczęściej operacji:

```
size_t fwrite(const void *ptr, size_t size,  
size_t nmemb, FILE *stream)
```

- ▶ Odczytać pozycje w pliku binarnym pozwala funkcja: `long ftell(FILE *stream)` a powrócić do ustalonej pozycji: `int fseek(FILE *stream, long offset, int whence)`

Operacje na plikach binarnych - prosta baza danych I

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

#define NAZWA_BAZY "karty.dat"

typedef struct karta {
    long numer;
    char imie[20];
    char nazwisko[30];
    double kwota;
} Karta;

typedef enum oper {nowa=1,zmiana,stan} Operacja;

FILE *otworz_baze(void);
int czytaj_karte(FILE *baza,long numer,Karta *wsk);
int zapisz_karte(FILE *baza,long numer,Karta *wsk);
```

Operacje na plikach binarnych - prosta baza danych II

```
int main(int argc, char *argv[])
{
    Karta pierwsza, aktualna;
    double zmiana;
    FILE *baza;

    if (argc == 1) {
        fprintf(stderr, "Uzycie: %s <operacja> <argumenty>\n", argv[0]);
        fputs("dozwolone operacje:\n", stderr);
        fputs("    -nowa <imie> <nazwisko> <kwota>\n", stderr);
        fputs("    -zmiana <nr-karty> <kwota>\n", stderr);
        fputs("    -stan <nr-karty>\n\n", stderr);
        exit(1);
    }
    if ((baza = otworz_baze()) == NULL) {
        fprintf(stderr, "%s: blad otwarcia bazy\n", argv[0]);
        exit(2);
    }
    czytaj_karte(baza, 0L, &pierwsza);
    if (strcmp(argv[1], "-nowa") == 0 && argc == 5) {
        aktualna.numer = ++pierwsza.numer;
        strncpy(aktualna.imie, argv[2], 20);
        strncpy(aktualna.nazwisko, argv[3], sizeof(aktualna.nazwisko));
        aktualna.kwota = atof(argv[4]);
        zapisz_karte(baza, 0L, &pierwsza);
        zapisz_karte(baza, aktualna.numer, &aktualna);
    }
```


Operacje na plikach binarnych - prosta baza danych III

```
} else
if (strcmp(argv[1], "-zmiana") == 0 && argc == 4) {
    aktualna.numer=atoi(argv[2]);
    zmiana=atof(argv[3]);
    if (0 < aktualna.numer && aktualna.numer <= pierwsza.numer) {
        czytaj_karte(baza, aktualna.numer, &aktualna);
        aktualna.kwota+=zmiana;
        zapisz_karte(baza, aktualna.numer, &aktualna);
    } else {
        fprintf(stderr, "%s: zly numer karty\n", argv[0]);
        exit(2);
    }
} else
if (strcmp(argv[1], "-stan") == 0 && argc == 3) {
    aktualna.numer=atoi(argv[2]);
    if (0 < aktualna.numer && aktualna.numer <= pierwsza.numer)
        czytaj_karte(baza, aktualna.numer, &aktualna);
    else {
        fprintf(stderr, "%s: zly numer karty\n", argv[0]);
        exit(2);
    }
} else {
    fprintf(stderr, "%s: zla operacja lub liczba jej argumentow\n", argv[0]);
    exit(2);
}
```

Operacje na plikach binarnych - prosta baza danych IV

```
printf("Karta: %ld\nImie: %s\nNazwisko: %s\nKwota: %.2lf\n",
      aktualna.numer, aktualna.imie, aktualna.nazwisko, aktualna.kwota);
fclose(baza);
return 0 ;
}

FILE *otworz_baze(void)
{
    FILE *fp;
    Karta pierwsza;

    if ((fp=fopen(NAZWA_BAZY, "r+b")) == NULL) { /* baza nie istnieje */
        fp=fopen(NAZWA_BAZY, "w+b");
        pierwsza.numer=0L;
        fwrite(&pierwsza, sizeof(Karta), 1, fp);
        fclose(fp);
    }
    return fopen(NAZWA_BAZY, "r+b");
}
```

Operacje na plikach binarnych - prosta baza danych V

```
int czytaj_karte(FILE *baza, long numer, Karta *wsk)
{
    fseek(baza, numer*sizeof(Karta), SEEK_SET);
    return fread(wsk, sizeof(Karta), 1, baza);
}

int zapisz_karte(FILE *baza, long numer, Karta *wsk)
{
    fseek(baza, numer*sizeof(Karta), SEEK_SET);
    return fwrite(wsk, sizeof(Karta), 1, baza);
}
```