

Wstęp do programowania w języku C

Wskaźniki i ich podstawowe zastosowania

Preprocesor języka C

Marek Piotrów - Wykład 6

17 listopada 2021

Dlaczego potrzebujemy wskaźników

- Współdzielenie struktur danych przez różne fragmenty kodu (a kopiowanie danych może być nieefektywne).
- Budowanie złożonych struktur danych (np. drzew czy grafów) poprzez dynamiczne łączenie ich fragmentów.

Podstawowe własności wskaźników

- Pamięć zawiera wartości różnych typów; położenie wartości w pamięci opisuje adres.
- Zmienna jest abstrakcją miejsca w pamięci, w którym znajduje się wartość określonego typu.
- Zmienna jest opisywana przez R-wartość (wartość zmiennej) oraz L-wartość (adres zmiennej).
- Wskaźnik jest abstrakcją adresu - wskazuje wartość określonego typu.
- Wartością wskaźnika może być stała `NULL`, która nie wskazuje żadnej wartości (tzw. *pusty wskaźnik*).
- Operacja *dereferencji* wskaźnika (`*`) identyfikuje wskazywaną wartość (zwana też wyłuskaniem wartości).

Wskaźniki do parametrów wyjściowych - szesciokat.c

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

```
/* ***** WYZNACZANIE WSPOLRZEDNYCH SZESCIOKATA FOREMNEGO ***** */
```

```
/*
```

```
 * Ten program należy kompilować łącznie z modulem trojkat.c
```

```
 * np. poleceniem: gcc -std=c99 -o szesciokat szesciokat.c trojkat.c
```

```
 * lub tworząc odpowiedni projekt w Code::Blocks
```

```
*/
```

```
void trojkat_rb(float x1,float y1,float x2,float y2,float *x3,float *y3);
```

```
int main(void)
```

```
{
```

```
    float x1,y1,x2,y2,xs,ys,xp,yp,xn,yn;
```

```
    int i;
```

```
    printf("Podaj wspolrzedne dwuch wierzchoлков szesciokata foremego\n");
```

```
    printf("x1  y1 = "); scanf("%f %f",&x1,&y1);
```

```
    printf("x2  y2 = "); scanf("%f %f",&x2,&y2);
```

```
    trojkat_rb(x1,y1,x2,y2,&xs,&ys);
```

```
    for (i=3,xp=x2,yp=y2; i <= 6; ++i,xp=xn,yp=yn) {
```

```
        trojkat_rb(xs,ys,xp,yp,&xn,&yn);
```

```
        printf("%i-ty wierzcholek ma wspolrzedne (x%i,y%i) = (%.2f,%.2f)\n",
```

```
            i,i,xn,yn);
```

```
    }
```

```
    return 0;
```

```
}
```

Wskaźniki do parametrów wyjściowych - trojkat.c

```
#include <math.h>
```

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

```
void punkty_na_wektor(float x0,float y0,float x1,float y1,float *x,float *y);
```

```
void dodaj_wektor(float *x,float *y,float x1,float y1);
```

```
void mnoz_przez_liczbe(float *x,float *y,float a);
```

```
void trojkat_rb(float x1,float y1,float x2,float y2,float *x3, float *y3);
```

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

```
void punkty_na_wektor(float x0,float y0,float x1,float y1,float *x,float *y)
```

```
{  
    *x=x1-x0;  
    *y=y1-y0;  
}
```

```
void dodaj_wektor(float *x,float *y,float x1,float y1)
```

```
{  
    *x+=x1;  
    *y+=y1;  
}
```

```
void mnoz_przez_liczbe(float *x,float *y,float a)
```

```
{  
    *x*=a;  
    *y*=a;  
}
```

```
void trojkat_rb(float x1,float y1,float x2,float y2,float *x3, float *y3)
{
    float x12,y12,x14,y14,x43,y43;

    punkty_na_wektor(x1,y1,x2,y2,&x12,&y12);
    x14=x12; y14=y12;
    mnoz_przez_liczbe(&x14,&y14,0.5);
    x43=-y14; y43=x14;
    mnoz_przez_liczbe(&x43,&y43,sqrt(3.0));
    *x3=x1; *y3=y1;
    dodaj_wektor(x3,y3,x14,y14);
    dodaj_wektor(x3,y3,x43,y43);
}
```

Podstawowe operacje na wskaźnikach

- Tworzenie wskaźnika do zmiennej (&).
- Podstawianie wskaźników tego samego typu (nazwa tablicy jest w wielu kontekstach traktowana jako stała wskaźnikowa na zerowy element tablicy).
- Zwiększanie/zmniejszanie wskaźnika o stałą.
- Odejmowanie dwóch wskaźników tego samego typu (powinny wskazywać wartości w tej samej tablicy).
- Indeksacja wskaźnika.
- Porównywanie wskaźników (`==`, `!=`, `<`, `<=`, `>`, `>=`).
- Porównywanie wskaźnika ze stałą `NULL` (`==`, `!=`).

Proste operacje na napisach z biblioteki standardowej

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

```
int strlen(char *s);           // zwroc dlugosc napisu s  
char *strcpy(char *dop, char *zp); // skopiuj napis zp do napisu do dop
```

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

```
int strlen(char *s)           // zwroc dlugosc napisu s  
{  
    char *p;  
  
    for (p=s; *p != '\0'; ++p);  
    return p-s;  
}
```

```
char *strcpy(char *dop, char *zp) // skopiuj napis zp do napisu do dop  
{  
    char *p=dop;  
  
    while ((*p++ = *zp++) != '\0');  
    return dop;  
}
```


Używanie wskaźników - alloc.c

```
/****** PROSTY DYSTRYBUTOR PAMIECI *****/  
#include <stdlib.h>  
  
#define ALLOCSIZE 10000      /* rozmiar dostępnej pamięci */  
  
static char allocbuf[ALLOCSIZE]; // pamięć do dystrybucji  
static char *allocp=allocbuf;    // wskaźnik początku wolnego miejsca  
  
char *moj_alloc(int n)          // zwróc wskaźnik do n znaków  
{  
    if (allocbuf+ALLOCSIZE-allocp >= n) {  
        allocp+=n;  
        return allocp-n;  
    }  
    else  
        return NULL;  
}  
  
void moj_free(char *p)          // zwolnij pamięć wskazywaną przez p  
{  
    if (p >= allocbuf && p < allocp)  
        allocp=p;  
}
```

Sortowanie pliku według wybranej kolumny

```
#include <stdio.h>
#include <stdlib.h>
#include "mysort.h"
// Ten program należy kompilować łącznie z modułami mysort.c i alloc.c
#define MAX 1000
#define MAXDL 100

unsigned int pozycja=0, dlugosc=MAXDL;
char *moj_alloc(int n); // funkcja z modulu alloc.c
static int czytaj_wiersz(char wiersz[], int max);

int main(int argc, char *argv[])
{
    int licz_wiersz=0;
    char akt_wiersz[MAXDL+1], *p;
    char *wiersze[MAX];

    if (argc > 1 && atoi(argv[1]) > 0) {
        pozycja=atoi(argv[1])-1;
        if (argc > 2 && atoi(argv[2]) > 0) dlugosc=atoi(argv[2]);
    }
    for (int i; licz_wiersz < MAX && (i=czytaj_wiersz(akt_wiersz, MAXDL)) > 0 && (p=moj_alloc(i)) != NULL; ) {
        akt_wiersz[--i]='\0';
        strcpy(p, akt_wiersz);
        wiersze[licz_wiersz++]=p;
    }
    quicksort(wiersze, 0, licz_wiersz-1);
    for (int i=0; i < licz_wiersz; ++i) printf("%s\n", wiersze[i]);
    return 0;
}
```

sortowanie - cd.

/ funkcja czytaj_wiersz: czyta wiersz znakow z wejscia lacznie z '\n',
* zwraca dlugosc wiersza lub 0 jesli jest to koniec danych */*

```
static int czytaj_wiersz(char wiersz[],int max)
{
    int c,i;

    for (i=0; i < max-1 && (c=getchar()) != EOF; ++i)
        if ((wiersz[i]=c) == '\n') {
            ++i; break;
        }
    wiersz[i]='\0';
    return i;
}
```

mysort.h

```
#ifndef MYQSORTH

#define MYQSORTH
#include <string.h>

typedef char *TYP_ELEM;

#define MNIEJSZY(x,y) (strlen(x) <= pozycja ? \
    (strlen(y) <= pozycja ? (strcmp(x,y) < 0) : 1) :\
    (strlen(y) <= pozycja ? 0 : (strncmp(x+pozycja,y+pozycja,dlugosc) < 0)))

extern unsigned int dlugosc,pozycja;

void quicksort(TYP_ELEM tab[],int dol,int gora);

#endif
```

Implementacja algorytmu szybkiego sortowania quicksort

```
#include "mysort.h"
/***** sort.c *****/
* implementacja algorytmu szybkiego sortowania: quicksort *
* dla krótkich ciągów: sortowanie przez wstawianie *
/*****/

#define MALO 16
#define ZAMIEN(x,y,typ) {typ _5_6_; _5_6_=x; x=y; y=_5_6_; }

static int podziel(TYP_ELEM tab[], TYP_ELEM x, int dol, int gora);
static void sortuj(TYP_ELEM tab[], int dol, int gora);

void quicksort(TYP_ELEM tab[], int dol, int gora)
{
    if (gora-dol+1 < MALO)
        sortuj(tab, dol, gora);
    else {
        int srodek=podziel(tab, tab[dol], dol, gora);

        if (dol < srodek)
            quicksort(tab, dol, srodek-1);
        if (srodek < gora)
            quicksort(tab, srodek+1, gora);
    }
}
```

Implementacja quicksort'a - podział ciągu względem elementu x

```
static int podziel(TYP_ELEM tab[],register TYP_ELEM x,int dol,int gora)
{
    register int i=dol,j=gora+1;

    while (1) {
        do ++i; while (i <=gora && MNIEJSZY(tab[i],x));
        do --j; while (j >= dol && MNIEJSZY(x,tab[j]));
        if (i < j)
            ZAMIEN(tab[i],tab[j],TYP_ELEM )
        else
            break;
    }
    ZAMIEN(tab[dol],tab[j],TYP_ELEM )
    return j;
}
```

Implementacja quicksort'a - sortowanie (przez wstawianie) krótkich ciągów

```
static void sortuj(TYP_ELEM tab[], int dol, int gora)
{
    register int i, j;

    for (i=dol+1; i <= gora; ++i) {
        TYP_ELEM x=tab[i];

        for (j=i-1; j >= dol && MNIEJSZY(x, tab[j]); --j)
            tab[j+1]=tab[j];
        tab[j+1]=x;
    }
}
```