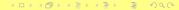
Wstęp do programowania w języku C

Marek Piotrów - Wykład 6 Wskaźniki i ich podstawowe zastosowania Preprocesor języka C

12 listopada 2014

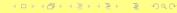


Dlaczego potrzebujemy wskaźników

- 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 modułem trojkat.c
   np. poleceniem: acc -std=c99 -o szesciokat szesciokat.c troikat.c
   lub tworzac odpowiedni proiekt w Code::Blocks
void troikat rb(float x1.float v1.float x2.float v2.float *x3.float *v3);
int main(void)
  float x1,y1,x2,y2,xs,ys,xp,yp,xn,yn;
  int i:
  printf("Podaj wspolrzedne dwoch wierzcholkow szesciokata foremnego\n"):
  printf("x1 v1 = "); scanf("%f %f",&x1,&v1);
  printf("x2 v2 = "): scanf("%f %f".&x2.&v2):
  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,v%i) = (%.2f,%.2f)\n".
      i,i,i,xn,yn);
  return 0:
```

Wskaźniki do parametrów wyjściowych - trojkat.c

```
#include < math.h >
/******* PROTOTYPY FUNKCJI ***********/
void punkty na wektor(float x0.float v0.float x1.float v1.float *x.float *v);
void dodaj wektor(float *x,float *y,float x1,float y1);
void mnoz przez liczbe(float *x,float *y,float a);
void troikat rb(float x1.float v1.float x2.float v2.float *x3. float *v3);
/****** DEFINICJE FUNKCJI ************/
void punkty na wektor(float x0.float v0.float x1.float v1.float *x.float *v)
  x=x1-x0:
  *y=y1-y0;
void dodai wektor(float *x.float *v.float x1.float v1)
  x_{+=x1}
  *y+=y1;
void mnoz przez liczbe(float *x,float *y,float a)
  *x*=a:
  *v*=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 dostepnei pamieci */
static char allocbuf[ALLOCSIZE]: // pamiec do dystrybucii
static char *allocp=allocbuf: // wskaznik poczatku wolnego miejsca
char *moj alloc(int n) // zwroc wskaznik do n znakow
 if (allocbuf+ALLOCSIZE-allocp >= n) {
  allocp+=n;
  return allocp-n:
 else
  return NULL:
void moj free(char *p)
                          // zwolnij pamiec wskazywana przez p
 if (p >= allocbuf && p < allocp)
  allocp=p;
```

Sortowanie pliku według wybranej kolumny

```
#include <stdio h>
#include <stdlib.h>
#include "mygsort.h"
// Ten program należy kompilować łacznie z modułami mygsort.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 czytai 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';
    strcpv(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.

myqsort.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</pre>
```

Implementacja algorytmu szybkiego sortowania quicksort

```
#include "mygsort.h"
 * implementacia algorytmu szybkiego sortowania: guicksort *
* dla krotkich ciagow: sortowanie przez wstawiania
#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)
    sortui(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;
}</pre>
```

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