Pointeri. Abordarea lucrului cu tablouri folosind pointeri.

Responsabili:

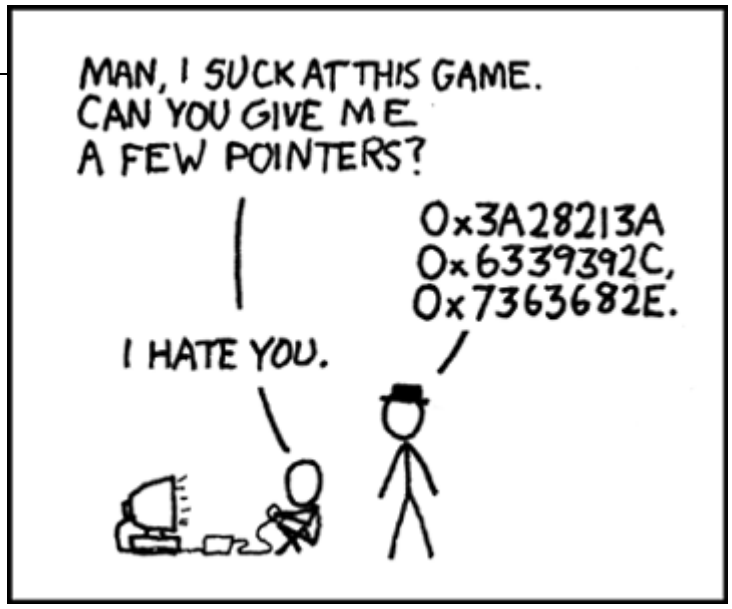
- Darius Neațu (CA 2019-2020) [mailto:neatudarius@gmail.com]
- Dorinel Filip (CA 2019-2020) [mailto:ion_dorinel.filip@cti.pub.ro]
- Laura Vasilescu [mailto:laura.vasilescu@cti.pub.ro]

Obiective

În urma parcurgerii acestui laborator, studentul va fi capabil:

- să înţeleagă noţiunea de pointer şi modurile în care se poate opera cu memoria în limbajul C
- să cunoască modul în care pointerii sunt folosiţi în a returna sau modifica parametri în cadrul unei funcţii
- să înţeleagă noţiunea de pointer la o funcţie şi să-l folosească în situaţiile în care acesta este necesar;
- să folosească funcţiile de alocare şi de eliberare a memoriei

Noţiunea de pointer



Un **pointer** este o variabilă care reţine o adresă de memorie.

În C, un pointer poate reprezenta:

1. adresa unor date de un anumit tip
    - tip elementar, structură, şir de caractere etc.
    - operațiile cu pointeri sunt determinate de dimensiunea tipului de date
2. adresa unei funcții
    - adresa la care punctul curent de execuţie va sări, în cazul în care acea funcţie este apelată
3. adresa unei adrese de memorie
    - acest tip de pointer poate fi redus la prima situaţie
4. adresa unei zone cu conţinut necunoscut (pointer către void)

Dimensiunea unui pointer depinde de arhitectura și sistemul de operare pe care a fost compilat programul. Dimensiunea unui pointer se determină cu **`sizeof(void *)`** și nu este în mod necesar egală cu dimensiunea unui tip de date întreg.

În cadrul laboratorului și al temelor de casă se vor utiliza mașini și pointeri pe 32 de biți.

Operatori

Operatorul de referențiere

& - apare în fața variabilei asupra căreia acționează

Este aplicat unei **variabile, avand ORICE tip de date,** și **obține ADRESA de (din) memorie** a variabilei respective.

Operatorul de dereferențiere

***** - apare în fața variabilei asupra căreia acționează

Este aplicat unei **variabile de tip pointer** și **obține valoarea stocată** la adresa respectivă (indicata de pointer).

Declararea unui pointer nu înseamnă alocarea unei zone de memorie în care pot fi stocate date. Un pointer este tot un tip de date, a cărui valoare este un număr ce reprezintă o adresă de memorie.

Pentru ca dereferențierea să aibă loc cu succes, pointer-ul trebuie să indice o adresă de memorie validă, la care programul are acces. Această adresă poate fi adresa unei variabile declarate în prealabil sau adresa unui bloc de memorie alocat dinamic (după cum vom vedea mai departe).

Este indicată inițializarea pointerilor cu constanta NULL [http://www.cplusplus.com/reference/cstddef/NULL/], compatibilă cu orice tip de pointer, care indica, prin convenție, un pointer neinițializat.

Particularități

```
int *a;             /* Pointer */
int b = 5;          /* Variabila */
char *c;            /* Pointer catre un caracter (sau sir de caractere) */
void *buff = NULL;  /* Pointer catre void, initializat la NULL */

/* Asignare NEVALIDA; a este un pointer neinitializat */
*a = 1;

/* a ia adresa variabilei b */
a = &b;

/* Continutul memoriei de la adresa a (care a fost initializata mai sus)
   ia valoarea 5. Acest lucru este echivalent cu "b = 5;"
 */
*a = 5;
```

În cazul declaraţiilor de pointeri, operatorul *este asociat numelui variabilei, şi nu numelui tipului, astfel că, pentru o declaraţie de mai multe variabile, operatorul * trebuie să apară pentru fiecare variabilă în parte şi este recomandat ca şi formatarea codului să indice această asociere. De exemplu:

```
/* sir1 e pointer, sir2 si sir3 sunt caractere */
char *sir1, sir2, sir3;

/* a, b si c sunt pointeri */
int *a, *b, *c;

/* Doar a este pointer; formatarea codului este nerecomandata */
char* a, b;
```

Operatorul * poate fi folosit şi în specificarea numelui unui tip (de exemplu în cazul unui `cast`), şi în acest caz el apare după numele tipului. De exemplu:

```
void *var = NULL;
int *num = (int *)var; // Operatie valida, dar riscanta
```

Un pointer către `void` nu poate fi folosit direct în operaţii cu pointeri, ci trebuie convertit mai întâi la un pointer către un tip de date.

De exemplu:

```
void *mem;
//[...]
*mem = 10;        // Operatie ILEGALA
((int *)mem) = 10; // Operatie legala, dar riscanta
```

Tipuri de pointeri

Pointeri la date

Stocarea datelor la o anumită adresă; Citirea datelor de la o anumită adresă

```
*p = y;         // Ia valoarea y si pune-o la adresa indicata de p
x = *p;         // Ia valoarea de la adresa indicata de p si pune-o in variabila x
*s1++ = *s2++;
```

Atribuirea unei adrese unui pointer

```
p1 = p2;
p = NULL;
p = malloc(n); // Veti studia malloc() intr-un laborator ulterior
```

Interpretarea diferită a datelor din memorie

```
int n;
short s1, s2;
s1 = *((short*)&n);     // Extrage primul cuvant din intregul n
s2 = *((short*)&n + 1); // Extrage cel de-al doilea cuvant din intregul n
```

Aritmetică cu pointeri

Adunarea sau scăderea unui întreg la un pointer, incrementarea sau decrementarea unui pointer. Aceste operaţii lucrează în multipli de dimensiunea tipului de date la care pointerii se referă, pentru a permite accesul la memorie ca într-un vector (a se vedea laboratorul de tablouri). De exemplu:

```
int *num;

/* Aduna la adresa initiala dimensiunea tipului de date referit
   de pointer (pe sizeof(int)), dand acces la urmatorul intreg
   care ar fi stocat daca zona aceea de memorie ar fi organizata
   sub forma unui vector
  */
num++;

/* Incrementeaza adresa cu 5 * sizeof(int) */
num = num + 5;
```

Pointeri la tablouri

După cum a fost prezentat în laboratorul de vectori, o variabilă vector conţine adresa de început a vectorului (adresa primei componente a vectorului), şi de aceea este echivalentă cu un pointer la tipul elementelor din vector. Această echivalenţă este exploatată, de obicei, în argumentele de tip vector şi în lucrul cu vectori alocaţi dinamic. De exemplu, pentru declararea unei funcţii care primeşte un vector de întregi şi dimensiunea lui, avem două posibilităţi:

```
void printVec(int a[], int n);
```

sau

```
void printVec(int *a, int n);
```

În interiorul funcţiei ne putem referi la elementele vectorului a fie prin indici, fie prin indirectare, indiferent de felul cum a fost declarat parametrul vector a:

```
void printVec (int a[], int n)
{
  int i;
  for (i = 0; i < n; i++)
    printf("%6d", a[i]); // Indexare
}
```

sau

```
void printVec (int *a, int n)
{
  int i;
  for (i = 0; i < n; i++)
    printf("%6d", *a++); // Indirectare
}
```

Astfel, există următoarele echivalenţe de notaţii pentru un vector a:

```
a[0] <==> *a
a[1] <==> *(a + 1)
a[k] <==> *(a + k)
&a[0] <==> a
&a[1] <==> a + 1
&a[k] <==> a + k
```

Diferenţa dintre o variabilă pointer şi un nume de vector este aceea că un nume de vector este un pointer constant (adresa sa este alocată de către compilatorul C şi nu mai poate fi modificată la execuţie), deci nu poate apărea în stânga unei atribuiri, în timp ce o variabilă pointer are un conţinut modificabil prin atribuire sau prin operaţii aritmetice. De exemplu:

```
int a[100], *p;
p = a; ++p; //corect
a = p; ++a; //EROARE
```

De asemenea, o variabilă de tip vector conţine şi informaţii legate de lungimea vectorului şi dimensiunea totală ocupată în memorie, în timp ce un pointer doar descrie o poziţie în memorie (e o valoarea punctuală). Operatorul `sizeof(v)` pentru un vector `v[N]` de tipul `T` va fi `N * sizeof(T)`, în timp ce `sizeof(v)` pentru o variabila `v` de tipul `T *` va fi `sizeof(T *)`, adică dimensiunea unui pointer.

Ca o ultimă notă, este importat de remarcat că o funcţie poate avea ca rezultat un pointer, dar nu poate avea ca rezultat un vector.

Pointeri în funcţii

În cadrul funcţiilor, pointerii pot fi folosiţi, printre altele, pentru:

- Transmiterea de rezultate prin argumente
- Transmiterea unei adrese prin rezultatul funcţiei
- Utilizarea unor funcţii cu nume diferite (date prin adresele acestora)

O funcţie care trebuie să modifice mai multe valori primite prin argumente sau care trebuie să transmită mai multe rezultate calculate în cadrul funcţiei trebuie să folosească argumente de tip pointer.

De exemplu, o funcţie care primeşte ca parametru un număr, pe care il modifica:

```
// Functie care incrementeaza un intreg n modulo m
int incmod (int *n, int m) {
  return ++(*n) % m;
}

// Utilizarea functiei
int main() {
  int n = 10;
  int m = 15;

  incmod(&n, m);
  // Afisam noua valoare a lui n
  printf("n: %d", n);

  return 0;
```

O funcţie care trebuie să modifice două sau mai multe argumente, le va specifica pe acestea individual, prin câte un pointer, sau într-un mod unificat, printr-un vector, ca în exemplul următor:

```
void inctime (int *h, int *m, int *s);
// sau
void inctime (int t[3]); // t[0]=h, t[1]=m, t[2]=s
```

O funcţie poate avea ca rezultat un pointer, dar acest pointer nu trebuie să conţină adresa unei variabile locale. De obicei, rezultatul pointer este egal cu unul din argumente, eventual modificat în funcţie. De exemplu:

```
// Incrementare pointer p
char *incptr(char *p) {
  return ++p;
}
```

O variabila locală are o existenţă temporară, garantată numai pe durata execuţiei funcţiei în care este definită (cu excepţia variabilelor locale statice), şi de aceea adresa unei astfel de variabile nu trebuie transmisă în afara funcţiei, pentru a fi folosită ulterior. De exemplu, următoarea secvenţă de cod este `greşită`:

```
// Vector cu cifrele unui nr intreg
int * cifre (int n) {
  int k, c[5]; // Vector local

  for (k = 4; k >= 0; k--) {
    c[k] = n % 10;
    n = n / 10;
  }

  return c; // Aici este eroarea !
}
```

Astfel, o funcţie care trebuie să transmită ca rezultat un vector poate fi scrisă corect în două feluri:

- Primeşte ca argument adresa vectorului (definit şi alocat în altă funcţie) şi depune rezultatele la adresa primită (soluţia recomandată!);
- Alocă dinamic memoria pentru vector (folosind `malloc`), iar această alocare se menţine şi la ieşirea din funcţie.

Studiu de caz

Anumite aplicaţii numerice necesită scrierea unei funcţii care să poată apela o funcţie cu nume necunoscut, dar cu prototip şi efect cunoscut.

De exemplu, o funcţie care să calculeze integrala definită a oricărei funcţii cu un singur argument sau care să determine o radăcină reala a oricărei ecuaţii (neliniare).

Aici vom lua ca exemplu o funcţie `listf` care poate afişa (lista) valorile unei alte funcţii cu un singur argument, într-un interval dat şi cu un pas dat. Exemple de utilizare a funcţiei `listf` pentru afişarea valorilor unor funcţii de bibliotecă:

```
int main(void) {
  listf(sin, 0.0, 2.0 * M_PI, M_PI / 10.0);
  listf(exp, 1.0, 20.0, 1.0);

  return 0;
}
```

Problemele apar la definirea unei astfel de funcţii, care primeşte ca argument numele (adresa) unei funcţii. Prin convenţie, în limbajul C, numele unei funcţii neînsoţit de o listă de argumente şi de parantezele `()` specifice unui apel este interpretat ca un pointer către funcţia respectivă (fără a se folosi operatorul de adresare `&`). Deci `sin` este adresa funcţiei sin(x) în apelul funcţiei `listf`. Declararea unui argument formal (sau a unei variabile) de tip pointer la o funcţie are forma următoare:

```
tip (*pf) (lista_arg_formale)
```

unde:

- `pf` este numele argumentului (variabilei) pointer la funcţie
- `tip` este tipul rezultatului funcţiei

Parantezele sunt importante, deoarece absenţa lor modifică interpretarea declaraţiei. De exemplu, putem avea:

```
tip * f(lista_arg_formale) // functie cu rezultat pointer, si NU pointer
```

În concluzie, definirea funcţiei `listf` este:

```
void listf (double (*fp)(double), double min, double max, double pas) {
  double x,y;

  for (x = min; x <= max; x = x + pas) {
    y=(*fp)(x); // apel functie de la adresa din "fp"
    printf("\n%20.10lf %20.10lf", x, y);
  }
}
```

O eroare de programare care trece de compilare şi se manifestă la execuţie este apelarea unei funcţii fără paranteze; compilatorul nu apelează funcţia şi consideră că programatorul vrea să folosească adresa funcţiei. De exemplu:

```
if (kbhit)
  break; // echivalent cu if(1) break;

if (kbhit())
  break; // iesire din ciclu la apasarea unei taste
```

Deşi sunt întâlnite mai rar în practică, limbajul C permite declararea unor tipuri de date complexe, precum:

```
char *(*(*x)())[10];
```

În interpretarea acestor expresii, operatorii `()` şi `[]` au precedenţa în faţa * şi modul de interpretare al acestor expresii este pornind `din interior spre exterior`. Astfel expresia dată ca exemplu mai sus

este (numerele de sub expresie reprezintă ordinea de interpretare):

1. o variabila x
2. care este un pointer la o funcţie
3. fără nici un parametru
4. şi care întoarce un pointer
5. la un vector de 10 elemente
6. de tip pointer
7. către tipul `char`

Folosind acest procedeu, se pot rezolva şi alte situaţii aparent extrem de complexe:

```
    unsigned int *(* const *name[5][10] ) ( void );
```

care semnifică o matrice de 5×10 de pointeri către pointeri constanţi la o funcţie, care nu ia nici un parametru, şi care întoarce un pointer către tipul `unsigned int`.

Exerciții Laborator CB/CD

Primul exercițiu presupune rularea unei secvente de cod cu scopul de a clarifica diverse aspecte legate de pointeri. Analizați fiecare intrebare si incercati sa intuiti ce ar trebui sa se afiseze in continuare. După aceea verificați

ex.c

```
#include <stdio.h>

void next(void)
{
        fflush(stdout);
        getchar();
}

int main(void)
{
        int a[10];

        printf("a[0] = ?"); next();
        printf("a[0] = %d\n", a[0]); next();

        printf("*a = ?"); next();
        printf("*a = %d\n", *a); next();

        printf("a = ?"); next();
        printf("a = %p\n", a); next();

        printf("&a = ?"); next();
        printf("&a = %p\n", &a); next();

        printf("sizeof(a[0]) = ?"); next();
        printf("sizeof(a[0]) = %ld\n", sizeof(a[0])); next();

        printf("sizeof(*a) = ?"); next();
        printf("sizeof(*a) = %ld\n", sizeof(*a)); next();

        printf("sizeof(a) = ?"); next();
        printf("sizeof(a) = %ld\n", sizeof(a)); next();

        printf("sizeof(&a) = ?"); next();
        printf("sizeof(&a) = %ld\n", sizeof(&a)); next();

        printf("*a + 1 = ?"); next();
        printf("*a + 1 = %d\n", *a + 1); next();

        printf("*a + 2 = ?"); next();
        printf("*a + 2 = %d\n", *a + 2); next();
```

```
        printf("a + 1 = ?"); next();
        printf("a + 1 = %p\n", a + 1); next();

        printf("a + 2 = ?"); next();
        printf("a + 2 = %p\n", a + 2); next();

        printf("&a + 1 = ?"); next();
        printf("&a + 1 = %p\n", &a + 1); next();

        printf("&a + 2 = ?"); next();
        printf("&a + 2 = %p\n", &a + 2); next();

        return 0;
}
```

Următoarele două probleme vă vor fi date de asistent în cadrul laboratorului.

Checker laborator 8 [https://drive.google.com/drive/u/0/folders/1qB6EZLGVubKbuTXMtMue06egH_8fo25M] Tutorial folosire checker laborator [https://ocw.cs.pub.ro/courses/programare/checker]

Exerciţii de Laborator

1. **[3p]** În reprezentarea unui număr întreg pe mai mulţi octeţi (de exemplu un short sau un int), se pune problema ordinii în care apar octeţii în memorie. Astfel, există două moduri de reprezentare:
   - Big-endian (în care primul octet din memorie este cel mai semnificativ)
   - Little-endian (în care primul octet din memorie este cel mai puţin semnificativ)
     Se cere să se scrie un program care să determine endianess-ul calculatorului pe care este compilat şi rulat, şi să afişeze un mesaj corespunzător pe ecran.
2. În limbajul C, şirurile de caractere sunt reprezentate în memorie ca o succesiune de valori de tip char terminate printr-un caracter special NULL ('\0'). Șirurile de caractere sunt pasate ca parametri printr-un pointer la primul caracter din şir, şi sunt prelucrate până când se întâlneşte caracterul nul, indiferent care este lungimea reală a zonei alocate. Astfel, un anumit vector de caractere de lungime N poate stoca şiruri de caractere de lungime între 0 şi N-1 (pentru că nu considerăm şi caracterul nul ca făcând parte din conţinutul şirului).
   - **[1p]** Să se scrie o funcţie care calculează lungimea unui şir de caractere dat ca parametru

     ```
     int str_length(char *s);
     ```

   - **[1p]** Să se scrie o funcţie pentru ştergerea (eliminarea) a n caractere dintr-o poziţie dată a unui şir ce returnează adresa șirului de caractere modificat

     ```
     char * strdel(char *s, int pos, int n);
     ```

   - **[1p]** Să se scrie o funcţie pentru inserarea unui şir `s2` într-o poziţie dată `pos` dintr-un şir `s1`. Se va presupune că există suficient loc în vectorul lui `s1` pentru a face loc şirului `s2`. Funcţia returnează adresa şirului `s1`.

     ```
     char * strins(char *s1, int pos, char *s2)
     ```

   - **[1p]** Scrieţi o funcţie care stabileşte dacă un şir dat (format din caractere alfanumerice) este egal cu o mască, ce poate conţine caractere alfanumerice şi caracterul special '?'. Se consideră că acest caracter înlocuieşte orice alt caracter alfanumeric. De exemplu, “abcde” este echivalent cu ”?bc?e”.

```
int eq_mask(char *sir, char *masca);
```

- **[2p]** Scrieţi o funcţie care stabileşte dacă un cuvânt dat se găseşte sau nu într-un tablou de cuvinte.

```
int eqcuv(char *cuv, char **tablou);
```

Pentru testare folosiți următoarea funcție `main`:

```
int main(void) {
        char *tablou[100] = {"curs1", "curs2", "curs3"};
        char *cuv1 = "curs2", *cuv2 = "curs5";
        printf("curs2 %s in tablou\n",(eqcuv(cuv1, tablou)) ? "este" : "nu este");
        printf("curs5 %s in tablou\n",(eqcuv(cuv2, tablou)) ? "este" : "nu este");
}
```

3. **[3p]** Scrieţi un program care afişează valorile functiilor `sqrt`, `sin`, `cos`, `tan`, `exp` şi `log`, în intervalul [1..10], cu pasul 0.1. În acest scop, se creează un tablou de pointeri la aceste funcţii şi se apelează funcţiile în mod indirect prin aceşti pointeri.

Probleme laborator 14-16 [https://drive.google.com/open?id=1xohyDQGMxXdGMvzyega5wKBi9WWk6qXW]

programare/laboratoare/lab08.txt · Last modified: 2020/10/05 00:37 by darius.neatu