Tehnici de programare

pointeri la funcții; qsort; bsearch

Pointeri la funcții

În limbajul C putem declara pointeri la funcții, dacă dorim să stocăm adresa funcției într-o variabilă sau să o transmitem ca argument unei alte funcții. Pointerii la funcții se declară astfel:

*tip_bază (*ptr_fn)(argumente_funcție);*

Parantezele din jurul lui *"*ptr_fn"* sunt necesare deoarece, așa cum s-a discutat mai sus, modificatorii de tip din dreapta au precedența mai mare decât cei din stânga. Dacă ar fi lipsit parantezele, deci declarația ar fi fost *"tip_bază *ptr_fn(argumente_funcție)"*, *ptr_fn* ar fi fost o funcție care ar fi returnat un pointer la *tip_bază*.

La argumentele funcției se poate da doar tipul lor, fără a se mai scrie și numele, acesta nefiind relevant. Numele se poate da, dacă dorim ca declarația să fie mai descriptivă.

Când se preia adresa unei funcții, se folosește operatorul **&** (adresă), iar când se folosește pointerul la funcție, acesta se dereferențiază pentru a obține funcția pointată, ca în exemplul de mai jos:

```c
#include <stdio.h>

int f1(int a,int b)
{
   return a+b;
}

int f2(int a,int b)
{
   return a-b;
}

int main(void)
{
   int a=7,b=5;
   int (*pf)(int,int);       // pf - pointer la o functie cu doi parametri de tip int, care returneaza o valoare de tip int
   pf=&f1;                   // se seteaza pf cu adresa functiei f1
   printf("op(%d,%d)=>%d\n",a,b,(*pf)(a,b));          // se apelează functia pointata de pf cu argumentele a si b
   pf=&f2;
   printf("op(%d,%d)=>%d\n",a,b,(*pf)(a,b));
   return 0;
}
```

La apelul funcției pointată de pointerul *pf* s-au folosit paranteze în jurul pointerului, *"(*pf)"*, deoarece apelul de funcției are precedență mai mare decât dereferențierea. Dacă nu s-ar fi pus parantezele, ar fi însemnat că se apelează funcția, iar apoi se dereferențiază rezultatul returnat de aceasta (un *int*).

Pointerii la funcții sunt compatibili doar cu funcții care au exact același tip de parametri și valoare returnată ca cele declarate în pointer. De exemplu, nu putem folosi un pointer la o funcție cu 2 parametri pentru a apela o funcție cu 3 parametri sau un pointer la o funcție care returnează *int* ca să returnăm o valoare de tip *double*.

Limbajul C permite preluarea directă a adresei unei funcții, fără folosirea operatorului & și, analogic, apelul funcției pointate fără a mai fi necesară dereferențierea pointerului. Funcția *main* din exemplul de mai sus devine:

```
int main(void)
{
   int a=7,b=5;
   int (*pf)(int,int);  // pf este un pointer la o functie cu doi parametrii de tip int, care returneaza o valoare de tip int
   pf=f1;               // se seteaza pf cu adresa functiei f1
   printf("op(%d,%d)=>%d\n",a,b,pf(a,b));   // se apelează functia pointata de pf cu argumentele a si b
   pf=f2;
   printf("op(%d,%d)=>%d\n",a,b,pf(a,b));
   return 0;
}
```

Pointerii la funcții se folosesc în multe situații, cum ar fi: algoritmi generici, colecții eterogene, implementarea acțiunilor din interfețele grafice utilizator (GUI - Graphics User Interface), etc. Vom discuta în continuare doar primele două dintre aceste situații.

Algoritmi generici

Una dintre aplicațiile pointerilor la funcții o constituie implementarea *algoritmilor generici*. De exemplu, testarea dacă toate elementele unui vector îndeplinesc o anumită condiție, presupune iterarea vectorului și testarea fiecărui element conform condiției date. Dacă dorim să testăm diverse condiții, partea de iterare rămâne identică și se modifică doar condiția de testat. Am putea defini un algoritm generic de testare, care să accepte diferite condiții, ca în exemplul următor:

```
#include <stdio.h>

int pozitiv(int e)
{
   return e>=0;
}

int par(int e)
{
   return e%2==0;
}

// testeaza daca toate elementele din vectorul v, de dimensiune n, indeplinesc conditia cond
int testare(int *v,int n,int(*cond)(int))
{
   int i;
   for(i=0;i<n;i++){
      if(!cond(v[i]))return 0;
      }
   return 1;
}

int main(void)
{
   int v[5]={4,8,1,2,0};
   printf("toate elementele sunt pozitive: %d\n",testare(v,5,pozitiv));
   printf("toate elementele sunt pare: %d\n",testare(v,5,par));
   return 0;
}
```

Funcția *testare* iterează vectorul dat și returnează 1 dacă toate elementele îndeplinesc condiția dată, altfel returnează 0. Condiția este dată prin intermediul unui pointer la o funcție, care primește un element și îl testează, returnând 1 sau 0, dacă acel element îndeplinește sau nu condiția dată. Se constată în *main* că, apelând *testare* cu diverse condiții, putem refolosi această funcție pentru diverse situații, trebuind să rescriem doar condiția de testat. Funcțiile care returnează valori logice se mai numesc și *predicate*, astfel încât în exemplul de mai sus avem predicatele *pozitiv* și *par*.

Funcția qsort

Pentru algoritmi simpli nu este neapărat o îmbunătățire folosirea pointerilor la funcții dar, dacă algoritmul este foarte complex, conținând sute sau chiar mii de linii de cod, atunci implementarea unei variante generice a sa, care permite refolosirea unei mari părți din algoritm, reprezintă o certă îmbunătățire. În biblioteca standard C există doi algoritmi generici, *qsort (*quick sort) și *bsearch* (binary search), ambii declarați în antetul *stdlib.h*.

Funcția *qsort* este o implementare performantă a algoritmului quick sort și poate fi folosită pentru sortarea oricăror tipuri de vectori. *qsort* este declarat astfel:

```
void qsort(void *vector, size_t nElemente, size_t dimensiuneElement,
           int (*compar)(const void *ptrElement1,const void *ptrElement2));
```

qsort are următorii parametri:

- *vector* - un vector de elemente
- *nElemente* - numărul elementelor din vector
- *dimensiuneElement* - dimensiunea unui element din vector, exprimată în octeți
- *compar* - o funcție care primește pointeri la două elemente din vector (transfer prin adresă). Parametrii sunt de tipul "*const void **", adică pointeri generici la valori constante. Funcția *compar* va fi apelată de *qsort* cu perechi de elemente, primul element fiind în vector în stânga celui de al doilea. *compar* trebuie să compare elementele și să returneze un întreg cu următoarele semnificații:
  - <0 - ordinea elementelor este corectă, deci vor fi lăsate de *qsort* la pozițiile lor prezente
  - 0 - elementele sunt egale, deci ordinea nu contează
  - >0 - ordinea este incorectă, deci elementele trebuie inversate

Pointerii la valori de tip **const** specifică faptul că nu se poate modifica valoarea pointată. Pointerul în sine poate fi modificat, de exemplu să fie folosit ca iterator. Declarațiile devin mai descriptive, iar compilatorului i se permite să facă anumite optimizări.
Câteva proprietăți ale pointerilor la valori constante sunt redate în următoarea secvență de program:

```
char v[2]={'a', 'b'};
const char *p;          // pointer la valori de tip char constante
p=v;                    // corect, se modifica pointerul
p++;                    // corect, se modifica pointerul
printf("%c",*p);        // corect, valoarea doar se citeste, fara a fi modificata
*p='7';                 // gresit, se incearca modificarea unei valori constante
```

Exemplu qsort: Se dă un vector de puncte în plan, având coordonatele (x,y) de tipul *double*. Se cere să se sorteze acest vector în ordinea distanțelor punctelor față de origine.

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

typedef struct{
   double x,y;
   }Pt;
```

```c
double dist(const Pt *pt)                    // distanta fata de origine
{
   return sqrt(pt->x*pt->x+pt->y*pt->y);
}

// deoarece qsort transmite functiei de comparare adresele elementelor, functia va primi pointeri la elemente
// in acest caz, deoarece vectorul are elemente de tipul Pt, functia va primi parametri de tipul Pt*
int cmpDist(const void *elem1,const void *elem2)
{
   const Pt *p1=(const Pt*)elem1;
   const Pt *p2=(const Pt*)elem2;
   double d1=dist(p1);
   double d2=dist(p2);
   if(d1<d2)return -1;
   if(d1>d2)return 1;
   return 0;
}

int main(void)
{
   Pt puncte[5]={{1,3},{7,5},{0,0},{-2,4},{1,1}};
   int i,n=5;
   qsort(puncte,n,sizeof(Pt),cmpDist);
   for(i=0;i<n;i++){
      printf("%g, %g\n",puncte[i].x,puncte[i].y);
      }
   return 0;
}
```

Notă: în Linux, la compilarea programelor care conțin funcții din *math.h* (*sqrt*, *cos*, …), trebuie specifică în linia de comandă opțiunea ”-lm”, ceea ce înseamnă adăugarea (link) a funcțiilor matematice. Comanda va fi de forma:

gcc -lm -Wall -o prg prg.c

Aplicații propuse

Aplicația 2.1: Se cere un număr *n* și apoi un vector de *n* rezultate ale studenților la un examen. Fiecare rezultat este definit prin *(nume,nota)*. Se cere să se sorteze folosind *qsort* vectorul în ordinea notelor, notele cele mai mari fiind primele. Dacă două note sunt identice, rezultatele respective se vor sorta în ordinea alfabetică a numelor.

Aplicația 2.2: Să se implementeze o funcție care primește ca parametri un vector de întregi, numărul elementelor din vector transmis prin adresă și un predicat care testează dacă un întreg îndeplinește o anumită condiție. Funcția va sterge din vector toate elementele care nu îndeplinesc condiția dată și va actualiza numărul elementelor cu numărul de elemente rămas în vector. Să se testeze funcția cu un predicat care testează dacă un număr este negativ și să se afișeze vectorul rezultat.

Aplicația 2.3: Se introduce un număr întreg *n*<10 și apoi *n* numere de tip *float*. Să se folosească funcția *qsort* pentru a sorta aceste numere în mod crescător. Se va introduce apoi un număr *x* de tip *float*. Folosind funcția *bsearch,* să se determine dacă *x* există în vectorul sortat. Se pot consulta paginile de manual pentru funcțiile *qsort* și *bsearch*.

Aplicația 2.4: Să se implementeze o funcție care tabelează o funcție matematică de un parametru, primită ca argument. Funcția tabelată primește un parametru de tip *double* și returnează o valoare tot de tip *double*. Funcția de tabelare va primi ca parametri: *a* și *b* - capetele de tip *double* ale intervalului închis de tabelat, *n* - un număr întreg care arată în câte segmente egale se împarte intervalul *[a,b]*, incluzând capetele acestuia și *f* - funcția de tabelat. Să se testeze funcția de tabelare cu valori *a,b* și *n* citite de la tastatură, tabelând funcțiile *cos*, *sqrt* și *fabs* din *math.h*

Exemplu: *tabelare(-10,10,20,fabs)* va afișa pe câte o linie valori de forma *f(-10)=10 f(-9)=9 … f(10)=10*

Exemple de probleme date la examen

Subiectul 2.1 Să se scrie un program pentru tabelarea unor funcţii matematice. Programul trebuie să tabeleze funcţiile (`x*x*x+4)/(x*x+5)`, `x+10` şi `2*x*x-1`.
Tabelarea se va realiza pe intervalul [`A`,B] cu pasul P. Valorile A, B şi P se vor citi de la tastatură. Programul va afişa valorile funcţiilor sub formă de tabel.
Trebuie să se folosească pointeri la funcţii. Scrieţi o funcţie generică pentru tabelarea unei funcţii matematice oarecare. Această funcţie generică va avea un parametru de tip pointer la funcţie.
Folosind funcţia generică, tabelaţi cele patru funcţii matematice amintite mai sus.

Subiectul 2.2 Se citesc mai multe cuvinte din linia de comandă. Cu ajutorul pointerilor la funcții (un vector de pointeri la funcții) să se apeleze într-o structură repetitivă următoarele funcții. Să se afișeze rezultatele pentru fiecare cuvânt pe ecran și în fișierul text CUVINTE.TXT.

a) Primește parametru un șir de caractere și returnează lungimea șirului.
b) Primește parametru un șir de caractere și returnează numărul de vocale din text.
c) Primește parametru un șir de caractere și returnează numărul de litere mari din text.

Primește parametru un șir de caractere și returnează diferența codurilor ASCII ale primului caracter și al ultimului caracter.