Programarea calculatoarelor
Limbajul C

CURS 11

Argumente în linia de comandă
Pointeri la funcții
Directive preprocesare
Tipuri generice

Argumentele liniei de comandă

- La lansarea în execuție a unui fişier executabil, în linia de comandă, pe lângă numele fişierului se pot specifica argumente, care se transmit ca parametrii funcției main.
- Argumentele = date inițiale pentru program: nume de fişiere folosite de program, opțiuni diverse de lucru.
- Antetul funcției main va fi:

  int main(int argc, char ** argv)

- argc - numărul de argumente
  argv - adresa vectorului de pointeri la şiruri, reprezentând argumentele

Argumentele liniei de comandă

- Sistemul de operare
  - analizează linia de comandă
  - extrage cuvintele din linie (siruri separate prin spații albe)
  - alocă memorie pentru aceste cuvinte
  - introduce adresele lor într-un vector de pointeri (alocat dinamic).
- Argumentele liniei de comandă vor fi şirurile:
  argv[0] - numele fişierului executabil
  argv[1]
  ...
  argv[argc-1]

Exemplu: tipărirea argumentelor liniei de comandă

```
#include<stdio.h>
int main( int argc, char** argv){
        int i;
        puts ("Argumente:");
        for (i=0; i<argc; i++)
          puts (argv[i]);
        return 0;
}
```

- O linie de comandă de forma:
  listare iata patru argumente
- va lansa în execuție listare.exe, care va tipări pe ecran:
  listare.exe
  iata
  patru
  argumente

Exemplu: copierea conținutului unui fişier în alt fişier

- numele sursei şi destinației transmise în linia de comandă.

```
# include <stdio.h>
int main(int argc, char** argv){
        FILE *fisi, *fise;
        char c;
        if ( !(fisi = fopen(argv[1],"r") || !(fise=fopen(argv[2],"w") ) {
                puts("Fisierele nu pot fi deschise");
                return 1;
        }  //eroare daca fişierele nu pot fi deschise
        while((c=fgetc(fisi))!=EOF) //copiere caracter cu caracter
                fputc(fise);
        fclose(fisi);
        fclose(fise);
        return 0;
}
```

- Lansarea în execuție a programului:

  copiere fişier_sursa.dat fişier_dest.dat

Exemplu: copierea conținutului unui fişier în alt fişier utilizand redirectarea

- Folosind redirectarea fişierelor standard, se va lansa printr-o linie de comandă de forma:
  copiere1 <fişier_sursa.dat >fişier_dest.dat

- următorul program va avea acelaşi rezultat:

```
# include <stdio.h>
int main(void){
        char c;
        while ( (c=getchar()) != EOF )
                putchar(c);
        return 0;
}
```

Pointeri la funcții

- Problemă: funcție care să poată apela o funcție cu nume necunoscut, dar cu prototip şi efect cunoscut.
- Exemple:
  - Funcție care să sorteze un vector ştiind funcția de comparare a două elemente ale unui vector.
  - Funcție care să determine o rădăcină reală a oricărei ecuații (neliniare).
  - Funcție "listf" care poate afişa (lista) valorile unei alte funcții cu un singur argument, într-un interval dat şi cu un pas dat.

```
int main () {
        listf (sin, 0., 2*M_PI, M_PI/10.);
        listf (exp,1., 20., 1.);
        return 0;
}
```

Observații

- **Prin convenție, în limbajul C, numele unei funcții neînsoțit de o listă de argumente (chiar vidă) este interpretat ca un pointer către funcția respectivă (fără a se folosi operatorul de adresare '&')!**
- "sin" este adresa funcției "sin(x)" în apelul funcției "listf".
- O eroare de programare care trece de compilare şi se manifestă la execuție este apelarea unei funcții fără paranteze; compilatorul nu apelează funcția şi consideră că programatorul vrea să folosească adresa funcției!
- Exemplu:

```
if ( test ) break;          // gresit, echiv. cu if (1) break;
if ( test() ) break;
// iesire din ciclu daca funcția test intoarce true
```

Declarare pointeri la funcții

- Declararea unui argument formal (sau unei variabile) de tip pointer la o funcție are forma următoare:

  tip (*pf) (lista_arg_formale)

unde:

  - pf este numele argumentului (variabilei) pointer la funcție
  - tip este tipul rezultatului funcției

- Definirea funcției "listf":

```
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);      // sau: y=(*fp)(x);
    printf ("\n%20.10lf %20.10lf", x, y);
  }
}
```

Observații

- Parantezele sunt importante, deoarece absența lor modifică interpretarea declarației:

  Declarație *funcție cu rezultat pointer, nu pointer la funcție!!*

  tip * f (lista_arg_formale)
- Pentru a face programele mai explicite se pot defini nume de tipuri pentru tipuri pointeri la funcții, folosind declarația ***typedef***.

```
typedef double (* ftype) (double);
void listf (ftype fp, double min, double max, double pas) {
    double x, y;
    for (x=min; x<=max; x=x+pas) {
      y = fp(x);
      printf ("\n%20.10lf %20.10lf", x,y);
    }
  }
```

Funcții callback

- O funcție C transmisă, printr-un pointer, ca argument unei alte funcții F se numeşte şi funcție “callback”, pentru că ea va fi apelată “înapoi” de funcția F.
- De obicei, funcția F este o funcție de bibliotecă, iar funcția C este parte din aplicație.
- Funcția F poate apela o diversitate de funcții, dar toate cu acelaşi prototip, al funcției C.

Exemplu

- program cu meniu de opțiuni; operatorul alege una din funcțiile realizate de programul respectiv:

```
#include<stdio.h>
#include<stdio.h>
typedef void (*funPtr) ();

// funcții ptr. operatii realizate de program
void  unu () {
  printf ("unu\n");
}
void  doi () {
  printf ("doi\n");
}
void  trei () {
  printf ("trei\n");
}
```

Exemplu - continuare

```
// selectare şi apel funcție
int main () {
  funPtr tp[ ]= {unu,doi,trei};        // vector de pointeri la funcții
  short option=0;
  do {
     printf(“Optiune (1/2/3):“);
     scanf ("%hd", &option);
     if (option >=1 && option <=3)
            tp[option-1]();   // apel funcție (unu/doi/trei)
  } while (1);
  getchar();
  return 0;
}
```

Exemplu - observație

Secvența echivalentă (cu *switch*) este :

```
do {
   printf(“Optiune (1/2/3):“);
   scanf ("%hd", &option);
   switch (option) {
     case 1: unu(); break;
     case 2: doi(); break;
     case 3: trei(); break;
     default: continue;
   }
} while (1);
```

Directive preprocesor

- sunt interpretate într-o etapă preliminară compilării (traducerii) textului C, de un preprocesor
- nu se foloseste caracterul ‘;’ pentru terminarea unei directive!
- #define ident text

  inlocuieşte toate aparițiile identificatorului “ident” prin şirul “text”
- #define ident (a1,a2,...) text

  defineşte o macroinstructiune cu argumente
- #include “fişier”

  include în compilare conținutul fişierului sursa “fişier”
- #if expr

  compilare condiționată de valoarea expresiei “expr”
- #if defined ident

  compilare condiționată de definirea unui identificator (cu #define)
- #endif

  terminarea unui bloc introdus prin directiva #if

Directive de preprocesare

- Substituția lexicală – constante simbolice

```
#define identificator [text]
#define void
#define then
#define begin {
#define end }
#define N 100
```

- Includerea fişierelor

```
#include <nume_fişier>
#include ”nume_fişier”
```

Macrouri

- Au aspect de funcție
- Pentru compilarea mai eficientă a unor funcții mici, apelate în mod repetat.
- Pot conține şi declaratii şi se pot extinde pe mai multe linii
- Pot fi utile în reducerea lungimii programelor sursă şi a efortului de programare.

Exemple:

```
# define max(A,B)  ( (A)>(B) ? (A):(B) )
#define randomize()   srand ((unsigned)time(NULL))
#define abs(a) (a)<0 ? -(a) : (a)
#define random(num) (int)(((long) rand() * (num)) /
    (RAND_MAX+1))
// generează un număr aleator între 0 şi num !
```

Exemplu

```
#include<stdio.h>
#define PAR(a) a%2==0 ? 1 : 0
int main(void)
{
    if (PAR(9+1)) printf("este par\n");
    else printf("este impar\n");
    getchar();
    return 0;
}
```

Atenție!

9+1%2==0 va conduce la 9+0 ==0 F ->”este impar”

Ar trebui:

#define PAR(a) (a)%2==0 ? 1 : 0

Programare generică în C. Colecții de date generice

- Colecția poate conține:
  - valori numerice de diferite tipuri şi lungimi sau
  - şiruri de caractere sau
  - alte tipuri de date agregat (structuri), sau
  - pointeri (adrese).
- Problemă: operațiile cu un anumit tip de colecție să poată fi scrise ca funcții generale, adaptabile pentru fiecare tip de date ce va face parte din colecție!
- Rezolvare:

1. utilizarea de tipuri generice (neprecizate) pentru elementele colecției în subprogramele ce realizează operații cu colecția.
2. utilizarea unor colecții de pointeri la un tip neprecizat (*void* * în C); înlocuirea cu un alt tip de pointer (la date specifice aplicației) se face la execuție.

1. Utilizarea de tipuri neprecizate

```
typedef  int T;                    // tip componente multime
typedef  struct {
    T m[M];                        // multime de intregi
    int n;                         // dimensiune multime
} Set;
     // operatii cu o multime
int findS ( Set a, T x) {          // cauta pe x în multimea a
  int j=0;
  while ( j < a.n  &&  x != a.m[j] )
      ++j;
   if ( j == a.n) return 0;        // negasit
   return 1;                       // gasit
}
int addS ( Set* pa, T x) {         // adauga pe x la multimea a
  if ( findS (*pa,x) )
      return 0;                    // nu s-a modificat multimea a
  pa® m[pa®n] = x;
  pa®n++;
  return 1;                        // s-a modificat multimea a
}
```

Observație

Operații valabile pentru orice tip:

a) Definirea unor operatori generalizați, modificați prin macro-substituție :

```
#define EQ(a,b)  (a == b)      // equals
#define LT(a,b)  (a < b)       // less than
#define AT(a,b)  (a = b)       // assign to
int findS ( Set a, T x) {      // cauta pe x în multimea a
   int j=0;
   while ( j < a.n  &&  ! EQ(x,a.m[j]) )
      ++j;
   if ( j==a.n) return 0;      // negasit
   return 1;                   // gasit
}
int addS (Set* pa, T x) {       // adauga pe x la o multime
  if ( findS (*pa,x) )
     return 0;                 // nu s-a modificat multimea
  AT(pa®m[pa®n++],x);          // adaugare x la multime
  return 1;                    // s-a modificat multimea
}
```

Observație

- Pentru o mulțime de şiruri de caractere trebuie operate următoarele modificări în secvențele anterioare :

```
#define EQ(a,b)  ( strcmp(a,b)==0) // equals
#define LT(a,b)  (strcmp(a,b) < 0)  // less than
#define AT(a,b)  ( strcpy(a,b) ) // assign to
typedef char * T;
```

Observație

b) Utilizarea unor funcții de comparație cu nume predefinite, care vor fi rescrise în funcție de tipul T al elementelor mulțimii:

```
typedef  char * T;
     // comparare la egalitate şiruri de caractere
int  comp  (T a, T b ) {
   return strcmp (a, b);
}
int findS ( Set a, T x) {      // cauta pe x în multimea a
  int j=0;
  while ( j < a.n  &&  comp(x,a.m[j]) ==0 )
     ++j;
  if ( j==a.n) return 0;
  return 1;
}
```

Observație

c) Transmiterea funcțiilor de comparare, atribuire, ş.a ca argumente la funcțiile care le folosesc (fără a impune nume fixe acestor funcții)

```
typedef  char * T;      // definire tip T

  // tip funcție de comparare
typedef  (int *) Fcmp ( T a, T b) ;

   // cauta pe x în multimea a
int findS ( Set a, T x, Fcmp cmp ) {
  int j=0;
  while ( j < a.n  &&  cmp(x,a.m[j]) ==0 )
      ++j;
  if ( j==a.n) return 0;
  return 1;
}
```

2. Utilizarea de pointeri la “void”

- Colecție generică = colecție de pointeri la orice tip (***void ***),** care vor fi înlocuiți cu pointeri la datele folosite în fiecare aplicație
- Funcția de comparare trebuie transmisă ca argument funcțiilor de prelucrare (inserare, căutare, etc) a colecției

- Avantaj: funcțiile pentru operatii cu colecții pot fi compilate şi puse într-o bibliotecă şi nu este necesar codul sursă
- Dezavantajul unor colecții de pointeri apare în aplicațiile numerice: pentru fiecare număr trebuie alocată memorie la execuție ca să obținem o adresă distinctă ce se memorează în colecție!

Exemplu: Mulțime ca vector de pointeri

```
#define M 100
typedef void* Ptr;
typedef int (*Fcmp) (Ptr,Ptr) ;          // tip funcție de comparare
typedef void (*Fprint)  (Ptr);           // tip funcție de afisare
typedef struct  {
     Ptr v[M];          // un vector de pointeri la elementele multimii
     int n;             // nr elem în multime
} * Set;

// afisare date din multime
void printS ( Set a, Fprint print) {
  int i;
  for (i=0;i<a→n;i++)
       print ( a→v[i] );  // depinde de tipul argumentului
  printf ("\n");
}
```

Exemplu

```
 // initializare multime
void initS (Set a) {
  a→n=0;
}
// cautare în multime
int findS ( Set a, Ptr p, Fcmp comp ) {
  int i;
  for (i=0; i<a→n; i++)
    if (comp(p,a→v[i]) == 0 )
      return 1;
  return 0;
}
 // adaugare la multime
int addS ( Set a, Ptr p, Fcmp comp) {
  if ( findS(a,p,comp) )
    return 0;                    // multime nemodificata
  a→v[a→n++] = p;                // adaugare la multime
  return 1;                      // multime modificata
}
```

Exemplu de creare şi afişare a unei mulțimi de întregi

```
void  print ( Ptr p) {              // afisare număr intreg
     printf ("%d ", *(int*)p );
}
int intcmp ( void* a, void* b) {            // comparare de intregi
  return *(int*)a - *(int*)b;
}
int main () {
     Set a; int x; int * p;
     initS(a);
     printf ("Elem. multime: \n");
     while ( scanf ("%d", &x) > 0) {
          p= (int*) malloc (sizeof(int));
          *p=x;
          add (a, p, intcmp);
     }
     printS (a);
     return 0;
}
```

Funcții generice predefinite

- “stdlib.h”: funcții generice pentru sortarea şi căutarea binară într-un vector cu componente de orice tip
- Ilustrează o modalitate simplă de generalizare a tipului unui vector: argumentul formal de tip vector al acestor funcții este declarat ca *void** şi este înlocuit cu un argument efectiv pointer la un tip precizat (nume de vector).
- Un alt argument al acestor funcții este adresa unei funcții de comparare a unor date de tipul celor memorate în vector, funcție furnizată de utilizator şi care depinde de datele folosite în aplicația sa.

bsearch

- **void * bsearch(const void *key, const void *base, size_t nelem, size_t width, int (*fcmp)(const void *, const void *));**
  - returnează adresa primei intrări din tablou care coincide cu parametrul căutat şi zero dacă acesta nu există în tablou (căutare binară)
  - key- adresa cheii căutate
  - base - începutul tabloului
  - nelem - nr.elemente din tablou
  - width - dim. unui elem. de tablou
  - fcmp - funcția de comparare definită de utilizator şi care primeşte doi parametri
- Atenție: Vectorul trebuie sa fie sortat conform funcției de comparație!

qsort

- **void qsort (void *base, size_t nelem, size_t width, int (*fcmp)(const void *, const void *));**
- sortează tabloul dat (algorimtul Quicksort)
  - base - începutul tabloului
  - nelem - nr.elemente din tablou
  - width - dim. unui elem. de tablou
  - fcmp - funcția de comparare definită de utilizator şi care primeşte doi parametri

Utilizarea funcției Qsort

- ordonarea un vector de numere întregi cu funcția “qsort” :

```
  // comparare numere intregi
int intcmp (const void * a, const void * b) {
  return *(int*)a -*(int*)b;
}
void main () {
   int a[]= {5,2,9,7,1,6,3,8,4};
   int i, n=9;                                  //n=dimensiune vector
   qsort ( a,9, sizeof(int), intcmp);   // ordonare vector
   for (i=0;i<n;i++)                            // afisare rezultat
     printf("%d ",a[i]);
}
```

Exemplu

- Să se scrie un program care execută în mod repetat următoarele operații:
  - Preia informațiile (nume şi nota) pentru o grupa de studenți. Citirea se face dintr-un fişier al cărui nume se specifică de utilizator.
  - Verifică prezența unui student în grupă utilizând bsearch
  - Listează grupa în ordine alfabetică (qsort) şi afisează media grupei.
  - Termină program.

```
# include <stdio.h>
# include <stdlib.h>
# include <ctype.h>
# include <string.h>
# define MAX_S  30
# define MAX_L  20

int cmp (const void*a, const void *b) {
   return strcmp ( (char*)a, (char*)b );
}
```

Exemplu - continuare

```
float citire (FILE *fp, char tab[][MAX_L], int *nrstud) {
  float nota;
  int i;
  float medie=0;
  i=0;
  while ( fscanf (fp, "%s%f", tab[i++], &nota) != EOF)
    medie+=nota;
  *nrstud=i-1;
  medie/=(*nrstud);
  qsort (tab, *nrstud, MAX_L, cmp);
  return medie;
}

int cauta(char *s, char tab[][MAX_L], int nrstud){
  char *t;
  t = (char*) bsearch (s, tab, nrstud, MAX_L, cmp);
  return (t-tab[0])/MAX_L;
}
```

Exemplu - continuare

```
int main(){
 char stud[MAX_S][MAX_L];
 FILE *fp=NULL;
 int nrstud=0;
 float medie;
 char c,s[12];
 int i;
 while(1){
    printf("\nOptiunea:\nCitire, Prezenta_student, Listare, Iesire\n >");
    fflush(stdin);
    c=getchar();
     switch(toupper(c)){
      case 'C': printf("nume fişier:"); fflush(stdin); gets(s);
                if ( (fp=fopen(s,"r")) == NULL ) {
                     printf("fişierul %s nu exista\n",s);
                     break;
                }
                medie = citire (fp, stud, &nrstud);
                break;
```

Exemplu - continuare

```
case 'P': printf("nume student:"); fflush(stdin); gets(s);
          if ( (i=cauta(s,stud,nrstud)) <0 )
              printf("nu exista studentul %s în grupa",s);
          else
              printf("studentul %s este al %d-lea din grupa",s,i+1);
          break;
    case 'L': printf("Lista studenti:\n");
          for (i=0; i<nrstud; i++)
              printf("%d %s\n", i+1, stud[i]);
          printf("\n\n media grupei = %4.2f\n\n",medie);
          break;
    case 'I': printf("terminare program\n");
          fclose(fp); exit(0);
   } /*switch*/
  } /*while*/
  getchar();
  return 0;
}
```

Exercițiu

- Modificați exemplul anterior astfel:
  - Se va defini o structură student cu nume şi notă
  - Se va utiliza un vector de astfel de structuri
  - Funcția de listare va afişa ordonat acest vector (nume şi nota)
- Observație: se vor utiliza tot qsort şi bsearch