UNIVERSITATEA TEHNICĂ A MOLDOVEI

PROGRAMAREA CALCULATOARELOR
VARIABILE. CLASE DE MEMORARE. COMPILARE CONDIȚIONATĂ. ARGUMENTE ÎN LINIA DE COMANDĂ

Prelegere

Kulev Mihail, dr., conf. univ.

Stimați studenți și stimată audiență!

Mă numesc Kulev Mihail sunt doctor, conferențiar universitar la Universitatea Tehnică a Moldovei. Din cadrul cursului PROGRAMAREA CALCULATOARELOR Vă propun spre atenția Dumneavoastră prelegerea cu tema:

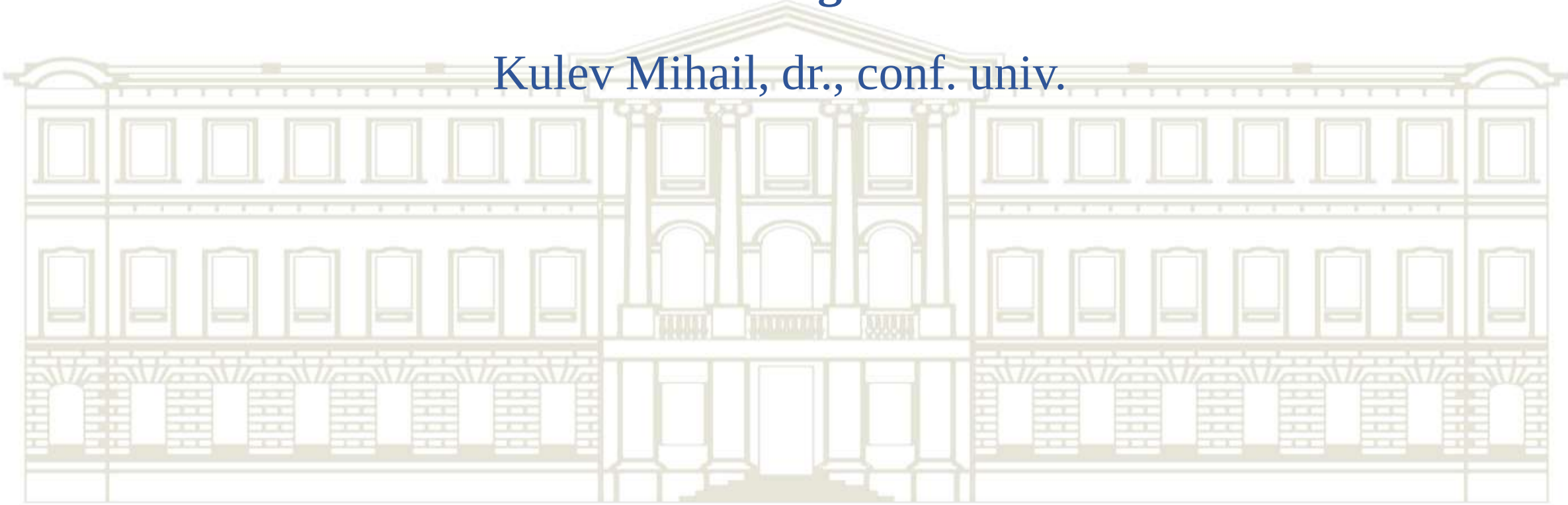
”VARIABILE. CLASE DE MEMORARE. COMPILARE CONDIȚIONATĂ. ARGUMENTE ÎN LINIA DE COMANDĂ”

UNIVERSITATEA TEHNICĂ A MOLDOVEI

PROGRAMAREA CALCULATOARELOR
VARIABILE. CLASE DE MEMORARE. COMPILARE CONDIȚIONATĂ. ARGUMENTE ÎN LINIA DE COMANDĂ

Prelegere

Kulev Mihail, dr., conf. univ.



UNIVERSITATEA TEHNICĂ A MOLDOVEI

Variabile. Clase de memorare. Compilare condiționată. Argumente în linia de comandă

Conținutul prelegerii

1. Atribute ale variabilelor în limbajul C.

Vom defini care sunt atributele unei variabile ce se referă la modul de alocare, domeniul de vizibilitate si durata de viață a variabilei.

2. Variabile globale și locale.

Vom afla ce reprezintă și prin ce se diferă variabilele globale și locale și care sunt atributele lor.

3. Clase de memorare.

Vom determina care sunt clasele de memorare ale variabilelor cu nume și moduri de alocare pentru diferite variabile.

4. Compilare condiționată.

Vom studia care sunt directivele preprocesorului pentru compilarea condiționată și modalități de utilizare a directivelor respective.

5. Argumente în linia de comanda.

Vom afla ce reprezintă parametrii funcției main(), cum ei sunt legați cu argumentele liniei de comandă și modalități de utilizare a lor. **Exemple de cod.** Pe parcursul studierii întrebărilor menționate vom prezenta exemple de cod în limbajul C.

1. Atribute ale variabilelor în limbajul C

În limbajul C orice variabilă are 4 atribute:

1. *Modul (clasa) de alocare a memoriei* - caracterizează locul în care este memorată variabila:

- într-un segment (zonă) de date
- în zona stivă
- în registre de memorie ale procesorului
- în zona heap (variabile dinamice).

2. *Domeniul de vizibilitate* reprezintă partea din program în care variabila poate fi folosită (accesată):

- *la nivel de bloc (funcție)* - variabila este cunoscuta într-un bloc (funcție) din momentul declarării până la sfârşitul blocului
- *la nivel de fisier* - variabila este cunoscuta într-un fisier.

Atribute ale variabilelor în limbajul C

3. Durata (timpul) de viaţă - reprezintă timpul în care variabila are alocat spaţiu în memoria internă:

- durată statică - variabila are alocat spaţiu pe tot timpul execuţiei programului. Variabilele statice sunt memorate într-un segment (zonă) de date și sunt în mod automat iniţializate cu 0.
- durată locală - variabila are alocat spaţiu cât timp se execută instrucţiunile unui bloc (funcții). Alocarea de spaţiu se face în stivă sau într-un registru.
- durată dinamică - variabila este alocată şi dezalocată în timpul execuţiei programului prin funcţii speciale. Alocarea de spaţiu se face în heap.

4. Tipul variabilei - determină numărul de octeţi rezervaţi pentru variabilă.

Atributele unei variabile se stabilesc:

- implicit - în locul unde se face declaraţia
- explicit - prin anumiţi calificatori.

2. Variabile globale și locale

În limbajul C variabilele cu nume pot fi clasificate în:

- variabile globale (externe)
- variabile locale (interne)

Variabilele globale:

- se definesc în afara corpurilor funcţiilor
- li se rezervă spaţiu într-un segment (zonă) de date a memoriei înainte de execuţia programului. Alocarea de memorie se face o singură dată şi anume în fişierul sursă, în care variabila externă este definită. Tot în acel loc se face şi iniţializarea variabilei globale
- au durata statică de viață și reţin valori între apeluri la diferite funcţii
- sunt iniţializate implicit cu 0
- au vizibilitate la nivel de fișier - sunt văzute şi pot fi accesate din întregul fişier sursă în care sunt definite din punctul declaraţiei până la sfârşitul fişierului
- vizibilitatea lor poate fi extinsă la nivel de program.

Variabile globale și locale

O variabilă globală (externă) poate fi referită şi dintr-un alt fişier sursă al programului, dacă este declarată în acest fişier cu atributul **extern**. O asemenea declaraţie anunţă proprietăţile variabilei (tipul) fără a aloca memoria pentru variabilă. De exemplu: **extern int x;**

```
// FISIER1.C
int x = 5;      // aici se defineste variabila x
void F();       // prototipul functiei F, definita in FISIER2.C
void main(){
F(); }   // apelul functiei F
// FISIER2.C
extern int x;// declararea variabilei definite in alt fisier
void F()      // aici se defineste functia F
{ ... }
```

Variabile globale și locale

Spre deosebire de *variabilele globale* (*externe*), *variabilele locale* (*interne)* au un domeniu mult mai restrâns - ele sunt cunoscute numai în interiorul funcţiei (blocului de instrucțiuni), în care sunt definite.

Durata de viaţă a variabilelor interne este mai scurtă decât a variabilelor externe – alocarea de memorie se face la intrare în funcţie (bloc) și, după părăsirea funcţiei, memoria ocupată de variabilă să fie eliberată.

Într-o funcţie sunt vizibili:

- parametrii formali (sunt variabile locale ale funcției)
- variabilele locale declarate în interiorul corpului al funcţiei (blocului de instrucțiuni)
- variabilele globale ale programului (modulului, fișierului) care declară funcţia (variabilele declarate în afara funcţiei în aceeaşi unitate de compilare);
- variabilele globale exportate din alte module şi importate de modulul care declară funcţia.

3. Clase de memorare

Clasa de memorare arată când, cum si unde se alocă memoria pentru o variabilă cu nume. În limbajul C orice variabilă cu nume are o clasă de memorare, care rezultă fie dintr-o declarație explicită cu cuvântul cheie respectiv, fie implicit din locul unde este definită variabila.

! Există 4 moduri de alocare a memoriei pentru variabilele limbajului C și există doar 3 clase de memorare pentru variabilele cu nume:

- Static (clasa statică - cuvântul cheie **static**): memoria este alocată la compilare in zona (segmentul) de date din cadrul programului și nu se mai poate modifica în cursul execuției. Variabilele externe (globale), definite în afara funcțiilor, sunt implicit statice,

dar pot fi declarate **static** și variabile locale, definite in cadrul funcțiilor.

Clase de memorare

- Automat (clasa auto - cuvăntul cheie **auto**): memoria este alocată automat, la activarea unei funcții (unui bloc de instucțiuni), in zona stivă alocată unui program si este eliberată automat la terminarea funcției (blocului). Variabilele locale ale unui bloc (unei funcții) si parametrii formali sunt implicit din clasa auto.

- Register (clasa register – cuvântul cheie **register**): variabile implementate în registre de memorie ale procesorului pentru un timp de acces mai scurt.

- Dinamic (pentru variabilele dinamice): memoria se alocă la execuție in zona “heap” alocată programului, dar numai la cererea explicită a programatorului prin apelarea unor funcții de bibliotecă. Memoria este eliberată numai la cerere prin apelarea funcției “free”. Variabilele dinamice nu au nume,

deci, nu se pune problema clasei de memorare (atribut al variabilelor cu nume).

Clase de memorare

Variabilele alocate static pot fi inițializate numai cu valori constante (la compilare), dar variabilele auto (automatice) pot fi inițializate cu rezultatul unor expresii (la execuție). Toate variabilele externe si/sau statice sunt automat inițializate cu valori zero.

Cantitatea de memorie alocată pentru variabilele cu nume, rezultă din tipul variabilei sau din dimensiunea declarată pentru vectori.

Memoria alocată dinamic pentru variabilele dinamice este specificată explicit ca parametrul funcțiilor de alocare.

Variabile și funcții statice

Declararea unei variabile externe (globale) sau funcţii cu atributul **static** limitează domeniul obiectului la fişierul sursă compilat. Folosirea cuvântului extern nu asigură, în acest caz, accesul din alt fişier la obiectul declarat static.

```
#include <stdio.h>
static double a;
int main(){
  ++a;
  printf("a=%.2lf", a);
return 0;}
```

a - variabila globală (externă) statică este iniţializată implicit la 0.

Variabile statice

Variabilele interne (locale) pot fi şi ele declarate statice. Prin aceasta ele vor rămâne alocate şi după ieşirea din funcţie, asigurând memorarea de valori între apelurile funcţiei. Variabilele interne (locale) statice :

- au durata de viaţă din momentul definiţiei până la sfârşitul execuţiei programului
- sunt memorate într-un segment (zonă) de date a memoriei
- sunt iniţializate automat
- definiţia lor începe cu cuvântul **static**.

Variabile statice

```
#include <stdio.h>
void increm(void){
  static int a;
  printf("%d\n", ++a);}
int main(void){
  int k;
  for (k=0; k<5; k++)
    increm(); return 0;}
```

Variabila **a** este variabila internă (locală) statică și este inițializată la 0, astfel în primul apel se va afișa 1; următorul apel preia valoarea existentă (1) pe care o incrementează, deci, afișează 2 ș.a.m.d.

Vizibilitatea este la nivelul fișierului (modulului) și nu se poate extinde la nivelul programului.

Variabile clasei register

Pentru variabilele folosite în mod frecvent și pentru creşterea eficienţei programului, este posibilă alocarea variabilelor în regiștrii procesorului. În acest scop, variabilele se declară cu atributul **`register`**.

De exemplu: **`register int a;  register char c;`**

Pot fi declarate ca variabile regiştri numai variabilele automatice. Puţine tipuri de variabile (care ocupă un spațiu mic al memoriei) permit declararea ca variabile regiştri (de exemplu, tipurile **`char`** şi **`int`** dar nu **`double`**!).

Întrucât numărul de regiştri este limitat, declararea unui număr mare de variabile de acest tip determină, că după alocarea regiştrilor disponibili să se ignore această declarație.

Cu variabilele alocate în regiştri nu poate fi utilizat operatorul de adresare &.

Variabile definite în interiorul blocurilor

O variabilă definită în interiorul unei instrucţiuni compuse ascunde o variabilă cu acelaş nume definită într-un bloc exterior şi durează până la terminarea blocului.

O variabilă definită şi iniţializată într-un bloc, va fi reiniţializată la fiecare intrare în acel bloc.

O variabilă statică definită şi iniţializată într-un bloc, va fi iniţializată o singură dată, la prima intrare în acel bloc.

Se poate utiliza cuvântul cheie **auto** în declararea variabileor locale:

```
auto int a, b, c;
auto double f;
```

O variabilă (globală) sau o funcţie declarată cu cuvântul cheie **extern** este vizibilă şi în alt fişier decât cel în care a fost declarată.

Funcţiile au clasa de alocare extern; cuvântul cheie **extern** poate fi utilizat la declararea/

definirea funcţiilor: `extern double sinus(double);`

4. Compilare condiționată

Directivele preprocesorului în limbajul C sunt utilizate pentru:

- Includerea fişierelor (**`#include`**): introducerea conţinutului unui alt fişier în fişierul sursă.
- Macrosubstituţii (**`#define`**): înlocuirea unei secvenţe de text cu alta.
- **Compilare condiţionată**: în funcţie de unele circumstanţe când unele părţi ale codului sursă sunt (sau nu sunt) luate în considerare de compilator.

! Comenzile pentru preprocesor se scriu pe o singură linie. Preprocesorul

nu cunoaşte sintaxa limbajului C - funcţii, instrucţiuni sau expresii.

Compilare condiţionată

Dacă avem următoarea secvenţă:

```
#ifdef nume // sau #ifndef nume sau #if defined nume sau #if expresie
    textul programului
#else
    textul programuluif
#endif
```

Codul ce este compilat depinde de existenţa macroului ***nume***. Dacă acest macro este definit (adică am folosit linia **`#define nume`**), atunci se compilează textul programului cuprins între **`#ifdef`** şi **`#else,`** iar textul dintre **`#else`** şi **`#endif`** este ignorat.

Dacă macroul nu este definit atunci se va compila textul sursă cuprins între **`#else`** şi **`#endif`**, textul cuprins între **`#ifdef`** şi **`#else`** fiind ignorat.

Aceasta seamănă cu o instrucţiune **if**, dar se comportă total diferit: o instrucţiune **if** controlează care instrucţiune se execută în momentul rulării programului, pe când **`#ifdef`** controlează care parte a programului este compilată.

Compilare condiționată

Exemplu de evitare a dublei incluziuni a fisierelor header:

```
#ifndef LISTE_H

#define LISTE_H

// definitii obiecte

// ...
#endif //LISTE_H
```

5. Argumente în linia de comandă

Funcția **`main()`** poate avea două argumente prin care primește date transmise prin linia de comandă, ce lansează programul în execuție. Sistemul de operare analizează linia de comandă, extrage cuvintele din linie (șiruri separate prin spații albe), alocă memoria pentru aceste cuvinte și introduce adresele lor intr-un vector de pointeri de tip **`char`** (alocat dinamic).

Primul argument al funcției **`main()`** este dimensiunea vectorului de pointeri (la tipul char), iar al doilea argument este adresa vectorului de pointeri la tipul **`char`** (deci, este un pointer la pointer la tipul **`char`**).

Exemplu:
```
int main ( int argc, char ** argv) {
        int i;
for(i=0;i<argc;i++) //sau for(i=1;argv[i];i++) nu se afiseaza argv[0]
          printf ("%s ", argv[i]); return 0;}
```

Argumente în linia de comandă

Primul argument cu adresa în argv[0] este numele programului executat (numele fisierului ce contine programul executabil), iar celelalte cuvinte din linie sunt date inițiale pentru program (nume de fisiere folosite de program, optiuni de lucru diverse, cuvinte, stringuri etc.).

Prelucrarea argumentelor din linia de comandă este, de obicei, secventială și, deci, putem renunța la indici în referire la componentele vectorului de pointeri. Exemplu:

```
int main ( int argc, char ** argv) {
  while (argc --)
    printf ("%s ", *argv++); return 0;}
```

O linie de comandă de forma: **salutare Buna ziua, dragi studenti!** va lansa în execuţie fișierul executabil salutare.exe, care va tipări pe ecran: **Buna ziua, dragi studenti!**

Exemplu: copierea conţinutului unui fişier în alt fişier

Numele sursei şi destinaţiei transmise în linia de comandă.

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

Lansarea în execuţie a programului:

```
copiere fişier_sursa.txt fişier_dest.txt
```

Tutoriale online pentru tema prelegerii:

1. http://www.euroinformatica.ro/documentation/programming/Documentatie/curs%20scoala%20C++/curs%20scoala%20C++/
2. https://https://ocw.cs.pub.ro/courses/programare/laboratoare/lab13
3. http://andrei.clubcisco.ro/cursuri/anul-1/semestrul-1/programarea-calculatoarelor.html

VĂ MULȚUMESC PENTRU ATENȚIE!

MULTĂ SĂNĂTATE ȘI SUCCESE!