

Departamentul Calculatoare



- 2



- 3



Constante simbolice și macro-uri

- Directiva utilizată este **#define**
- Definirea unei **constante simbolice** este un caz special al definirii unui macro

#define nume text

- În timpul preprocesării **nume** este înlocuit cu **text**
- **text** poate să fie mai lungă decât o linie, continuarea se poate face prin caracterul **** pus la sfârșitul liniei
- **text** poate să lipsească, caz în care se definește o constantă vidă
- Înlocuirea se continuă până în momentul în care **nume** nu mai este definit sau până la sfârșitul fișierului
 - Renunțarea la definirea unei constante simbolice se poate face cu directiva **#undef nume**



- ```
#define nume(p1, p2, ..., pn) text
```

- ```
nume(p_actual1, p_actual2, ..., p_actualln)
```



```
int main()
{
    int x=2*BETA;
    int y=2*GAMMA;
    printf("%d %d\n",x,y); //70 80
    int m=MIN(x,y);
    printf("%d\n",m); //70
    int a=ABS1(x-y);
    int b=ABS2(x-y);
    printf("%d %d\n",a,b); //-150 10
    INTER(int,a,b);
    printf("%d %d\n",a,b); //10 -150
    INTER(int,a,b);
    printf("%d %d\n",a,b); //-150 10
    return 0;
}
```



- 7



- ```
#if expr
 text
#endif

#if expr
 text1
#else
 text2
#endif
```

- 8





```
#ifdef nume
 text
#endif
```

```
#ifdef nume
 text1
#else
 text2
#endif
```

- unde **nume** este o constantă care este testată de către preprocesor dacă este definită, **text**, **text1**, **text2** sunt porțiuni de cod sursă
- Dacă **nume** este definită atunci **text** respectiv **text1** sunt compilate, altfel numai **text2** este compilat și procesarea continuă după **#endif**



```
#ifndef nume
```

# #endif

```
#ifndef nume
```

text1

```
#else
```

text2

```
#endif
```

- unde **nume** este o constantă care este testată de către preprocesor dacă nu este definită, **text**, **text1**, **text2** sunt porțiuni de cod sursă
- Dacă **nume** nu este definită atunci **text** respectiv **text1** sunt compilate, altfel numai **text2** este compilat și procesarea continuă după **#endif**



# Compilarea condiționată

- `#ifdef` și `#ifndef` sunt folosite de obicei pentru a evita incluziunea multiplă a modulelor în programarea modulară
  - La începutul fiecărui fișier *header* se practică de obicei o astfel de secvență

```
#ifndef _MODUL_H_
#define _MODUL_H_
... // conținutul fișierului header
#endif /* _MODUL_H_ */
```

- Există o serie de nume predefinite care nu trebuie re/definite. Ex:
  - `__DATE__` data compilării
  - `__CDECL__` apelul funcției urmărește convențiile C
  - `__STDC__` definit dacă trebuie respectate strict regulile ANSI C
  - `__FILE__` numele complet al fișierului curent compilat
  - `__FUNCTION__` numele funcției curente
  - `__LINE__` numărul liniei curente



- ```
#ifndef __cplusplus
#error "Acest program trebuie compilat cu \
      compilatorul de C++"
#endif
```

- ```
tip const identificator=valoare;
sau
const tip identificator=valoare;
```

```
int const alpha=10;
const double beta=20.5;
```



# Compilarea condiționată

```
#include <stdio.h>
//constante simbolice
#define DEBUG
#define X -3
#define Y 5

int main()
{
#ifdef DEBUG
 printf("Suntem in functia %s\n", __FUNCTION__); //main
#endif
 #if X+Y
 double a=3.1;
 #else
 double a=5.7;
 #endif
 a*=2;
#ifdef DEBUG
 printf("La linia %d valoarea lui a este %f\n", __LINE__, a); //18 6.2
#endif
 a+=10;
 printf("a este %f", a); //16.2
 return 0;
}
```



- 
- ```

graph TD
    A([Root]) --- B([Left Child])
    A --- C([Right Child])
    B --- D([Left-Left])
    B --- E([Left-Right])
    C --- F([Right-Left])
    C --- G([Right-Right])
    G --- H([Right-Right-Left])
    G --- I([Right-Right-Right])
  
```



- 15



Modulul

- O colecție de funcții care rezolvă task-uri înrudite pentru rezolvarea unei probleme
- Exemple:
 - Modul care implementează funcții pentru lucrul cu numere complexe
 - Modul care implementează funcții pentru lucrul cu matrice
 - Modul care implementează funcții pentru lucrul cu informații din fișiere
 - etc...



- 17



- Tot ce se află în interiorul unui modul este privat
- Tot ce nu trebuie folosit direct din exteriorul unui modul trebuie să fie păstrat privat
- Exemplu de modul și utilizarea lui într-un program

18



- 19



Schelet de fișier *header*

```
/*
module1.h - Exemplu de fisier header pentru modul "module1"
Informatii generale despre modul
*/

#ifndef _MODULE1_H_
#define _MODULE1_H_
/* Includerea bibliotecilor necesare pentru declaratiile ce
urmeaza */
#include <stdlib.h>
#include <math.h>
#include "altFisierHeader.h"
/* Declaratii de constante */
/* Declaratii de tipuri */
/* Declaratii de variabile globale */
/* Prototipuri de functii */
#endif
```



- 21



22



- Definite la începutul unui fișier sursă
- Vizibile din punctul definirii lor până la sfârșitul fișierului sursă respectiv

- Trebuie utilizate cu atenție deoarece:
 - Introduc dependențe între diferitele părți ale aceluiași program
 - Fac programul mai greu de citit
 - Fac programul mai greu de întreținut
 - Pot să genereze coliziuni de nume
- Sunt inițializate automat
 - Numerele cu 0
 - Tablourile cu numere cu elemente de 0
 - Pointerii cu adresa NULL (0)



- Variabile externe

- Vizibile din alte fișiere sursă altele decât cel care conține definiția lor
- Acolo unde se dorește să fie vizibile se specifică cu ajutorul **extern**
extern tip identificador {, identificador};
- Pot fi declarate
 - După antetul unei funcții – vizibilitate doar în interiorul funcției
 - La începutul unui fișier sursă – vizibilitate în toate funcțiile din acel fișier sursă

- Variabile locale

- Se declară în interiorul unei funcții sau în interiorul unui bloc de instrucțiuni
- Vizibile doar în interiorul acelei funcții sau respectiv bloc de instrucțiuni
- Sunt neinitializate după declarație (au o valoare nedeterministă)



- Automate (de stivă)

- ```
int a, b, c;
```

- Alocate de compilator într-o zonă specială
- Persistă de-a lungul execuției programului
- Nu pot fi declarate externe în alt modul (sunt private modulului)

```
static int x, y, z;
```

- Alocate în regiștrii procesorului

```
register float f;
```

- 25



- 26



## Rezultate afișate:

- 1: a=10
- 2: a=10
- 3: a=20
- 4: a=20
- 5: a=30
- 6: a=20
- 7: a=10
- 8: a=60



- 28



# intregi.c

```
#include <math.h>
#include <stdio.h>
#include "intregi.h"
//functie care nu poate fi vizibila din
//exteriorul modului
static int divizibil(int x, int d) {
 return (x%d==0);
}

int prim(int x){
 static int nr_apeluri=0; //variabila
 //locala statica
 nr_apeluri++;
 printf("Apelul numarul %d al functiei
 prim pentru numarul %d!\n",
 nr_apeluri,x);
 if (x<2) return 0;
 int limita; //variabila locala
 //automata, neinitializata
 limita=sqrt(x)+0.001;
 for (int i=2; i<=limita; i++)
 if (divizibil(x,i)) return 0;
 return 1;
}
```



**src\_intregi.h**

```
#ifndef SIR_INTREGI_H_INCLUDED
#define SIR_INTREGI_H_INCLUDED

#define CAPACITATE 100
void adauga_prim_in_sir(int x);
void afiseaza_sir();

#endif
// SIR_INTREGI_H_INCLUDED
```

# sir\_intregi.c

```
#include <stdio.h>
#include "intregi.h"
#include "sir_intregi.h"
static int v[CAPACITATE]; //variabila
globala statica, sir initializat cu
'CAPACITATE' elemente de 0, nu poate
fi accesat din alte module
static int nr_elemente; //variabila
globala statica, initializata cu 0, nu
poate fi accesata din alte module
int nr; //variabila globala,
initializata cu 0, poate fi accesata
din alte module
void adauga_prim_in_sir(int x) {
 if (prim(x)) {
 v[nr_elemente++]=x;
 nr=nr_elemente;
 }
}
void afiseaza_sir() {
 printf("Sirul este: ");
 for (int i=0; i<nr_elemente; i++)
 printf("%d ",v[i]);
 printf("\n");
}
```



## Rezultate afișate

```
Apelul numarul 1 al functiei
prim pentru numarul 3 !

Apelul numarul 2 al functiei
prim pentru numarul 40 !

Apelul numarul 3 al functiei
prim pentru numarul 43 !

Apelul numarul 4 al functiei
prim pentru numarul 19 !

Exista 2 numere in sir!

Apelul numarul 5 al functiei
prim pentru numarul 2 !

Exista 3 numere in sir!

Sirul este: 43 19 2
```