Pointeri

BREVIAR TEORETIC: FUNCȚII CU NUMĂR VARIABIL DE PARAMETRI

Prin intermediul acestui tip de funcție avem posibilitatea de a transmite un număr variabil de parametri reali, în momentul apelului funcției.

Acest număr de parametru reali poate varia de la un apel la altul, în cadrul aceluia și program. O funcție de acest gen va conține în lista parametrilor formali și simbolul *elipsis* (...). Declarația va fi asemănătoare cu cea prezentată în continuare.

DECLARARE:

Declararea unei astfel de funcții va arăta astfel:

```
#include <stdarg.h>

[tip_fct] <id_functie>( [lista_par_fix], ... );
```

DEFINIREA FUNCȚIEI

Macrodefinițiile utilizate în operarea funcțiilor cu număr variabil de argumente:

```
void va_start( va_list ap, lastfix );

<type> va_arg( va_list ap, <type>  );

void va_end  ( va_list ap );
```

Aceste macrouri dau posibilitatea utilizatorului de a accesa argumentele unei funcții ce nu are un număr constant (fix) de argumente după cum urmează:

- **`va_start()`** - setează variabila *ap* de tip *va_list* la primul argument ce este transmis funcției;
- **`va_args()`** - transformă argumentul curent din lista la tipul *type* și îl întoarce, după care trece la următorul argument din listă;
- **`va_end()`** - ajută funcția să efectueze un return normal (să golească stiva înainte de întoarcere).

Parametrul lastfix al funcției va_start() este de fapt numele ultimului parametru fix transmis funcției.

```
void modelFunctie( int a, ... )
{
	int x;

	/*
	 * Tipul de data "va_list"
	 */
	va_list va;
	/*
	 * Macrodefinitia "va_start()" va initializa lista de parametri suplimentari
	 * cu următorul parametru de după ultimul parametru FIX.
	 */

	va_start( va, a );
	/*
	 * Macrodefinitia "va_arg()" va extrage următorul parametru de tip "int"
	 * din lista.
	 */

	x = va_arg( va, int );
	/*
	 * Eliberează lista de parametri suplimentari
	 */

	va_end( va );
}
```

APELUL FUNCȚIEI

```
int main( void )
{
	int varA;

	/* ...
	 * apeluri functie */
	modelFunctie( varA );
	modelFunctie( varA. 2, 3, 1, 0 );
	/* ...
	 */
}
```

BREVIAR TEORETIC: POINTERI

Simplist privind un pointer este o variabilă întreagă care stochează o adresă de memorie, adresă care poate indica o altă variabilă sau chiar un alt pointer.

Toate operațiunile pe pointeri sunt efectuate prin intermediul a doi operatori noi: „ *” (*star*) și „&”(*ampersand*):

- „&” este un operator unar care returnează adresa de memorie a unei variabile;
- „*” este complementar operatorului „&” și întoarce valoarea depozitată la o locație de memorie, adresă stocata într-un pointer și nu numai.

„*” poate fi interpretat - ca motto - „la adresa” în timp ce „&” poate fi interpretat prin „adresa”.

DECLARAREA POINTERILOR

Sintaxa declarației unui pointer de date este:

```
<tip> *<identificator_pointer>;
```

unde:

- *identificator_pointer* este numele unei variabile pointer;
- *tip* este tipul variabilei a cărei adresă o va conține.

UTILIZAREA POINTERILOR

Pointerii de tipuri similare pot fi utilizați în expresii de atribuire, după cum se arată mai jos, dar pentru pointerii de tipuri diferite va trebui să se specifice prin operatorul cast conversia dorită.

```
#include <stdio.h>

int main ()
{
   char ch = a;
   char* p1, *p2;

   p1 = &ch; // Atribuirea unei "adrese"

   p2 = p1; // Atribuire de pointeri
   // PImprima 'a' de doua ori
   printf (" *p1 = %c And *p2 = %c", *p1, *p2);

   return 0;
}
```

TRANSMITEREA PARAMETRILOR PRIN „REFERINȚĂ”

Pentru ca o funcție C să poată modifica valoarea unei variabile indicate ca parametru efectiv, trebuie declarat un parametru formal de tip pointer, iar la apelare trebuie să i se ofere explicit adresa variabilei. Declarația unei funcții de acest tip este următoarea:

```
void Inversare( int *a, int *b );
```

iar prototipul este:

```
void Inversare( int *a, int *b )
{
     int t;

     t = *a; *a = *b; *b = t;
}
```

apelul fiind de forma:

```
int x=2, y=3;
Inversare( &x, &y );
```

Observaţie: Un parametru de tip pointer este echivalent cu un parametru de tip şir pentru o funcție.

POINTERI LA FUNCȚII

Declararea unui parametru de tip pointer la funcție se face astfel:

```
int (*p_functie)( int , float );
```

Acest pointer nu poate conține decât adresa unei funcții ce are un prototip identic cu cel al pointerului:

```
int functie( int a, float b )
{
     /*
      * corpul functiei
      */
}

void main( void )
{
     p_functie = functie;
}
```

Se face observația ca numele funcției reprezintă adresa funcției. Apelul unei funcții prin intermediul unui pointer la funcție se face:

```
p_functie( 23, 45.6 );
```

APLICAȚII PROIECT ÎN C

Această tehnică va fi utilizată atunci când suntem în situația de a construi un fișier executabil pe baza a mai multor fișiere sursă C sau biblioteci.

Pentru a crea un proiect C, pe baza mai multor fișiere-sursă C va trebui să respectăm următoarele reguli:

- Aplicația poate fi divizată doar la nivel de funcție; nu putem împărți o funcție între doua fișiere-sursă C.
- Declarațiile de variabile globale, cât și de funcții, vor fi făcute în fișiere cu extensia *.h* (fișiere *header*), fișiere ce vor fi incluse, prin directiva #include în orice fișier-sursă C ce folosește cel puțin una din declarații.
- În proiect vor fi incluse doar fișiere-sursă C, ASM, fișiere-bibliotecă (*.lib*) sau fișiere cod-obiect (*.obj*).
- Fișiere header (*.h*) nu vor fi incluse în proiect.

CREAREA UNUI PROIECT ÎN BORLAND C 3.1

Paragraful următor arată modul în care construim un proiect C, format din doua fișiere-sursa C (fișierul ce va conține funcția main(), cat și fișierul ce va conține restul funcțiilor din proiect), împreună cu un fișier header (de declarații). Problema propusă are un caracter didactic.

1. Crearea fișierului „*main.c*”:

```
#include <stdio.h>
#include <stdlib.h>
#include "functii.h"

//Variabilă globală - externă
int  sir[ 100 ];

int main( void )
{
     int N; //Variabilă locală

     clrscr( );
     printf( "Introdu numarul de elemente: " );
     scanf ( "%d", &N );

     //Citirea elementelor din șir
     fCitire( N );

     //Afișarea sumei elementelor șirului
     printf( "\nSuma elementelor: %d", fSuma( N ) );

     return 0;
}
```

2. Crearea fișierului „*funcții.c*”:

```
#include <stdio.h>
#include "functii.h"

void fCitire( int n )
{
     int i;
     printf( "\nIntrodu cele %d elemente ale sirului!\n", n );
     for( i = 0; i < n; i++ )
     {
          printf( "S(%d) = ", i );
          scanf ( "%d", &sir[ i ] );
     }
}

int fSuma( int n )
{
     int i, suma = 0;
     for( i = 0; i < n; i++ )
          suma += sir[ i ];
```

```
    return suma;
}
```

3. Crearea fișierului „*funcții.h*”:

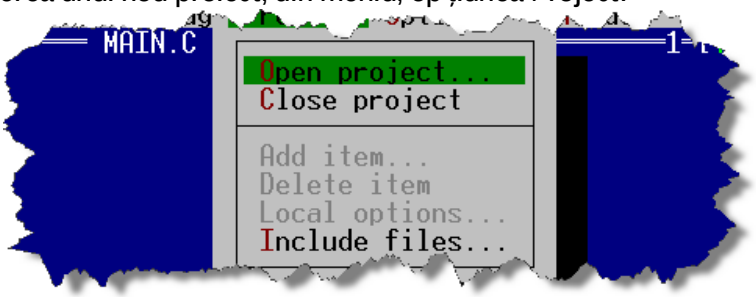
```
//Declarații variabile externe
extern int sir[];

// Declarații funcții
void fCitire( int n );
int fSuma( int n );
```

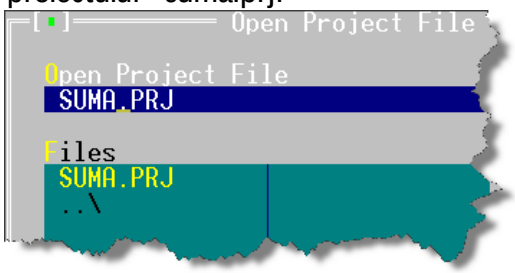
4. Crearea fișierului proiect „*suma.prj*”:

Pentru a realiza etapa finală, proiectul în Borland C 3.1, se vor parcurge următorii pași:

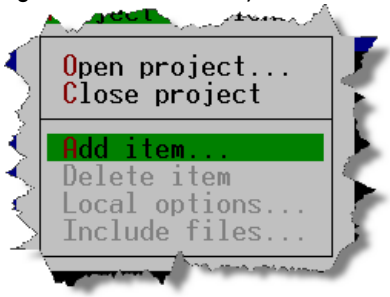
A) Deschiderea unui nou proiect, din meniu, opțiunea Project:

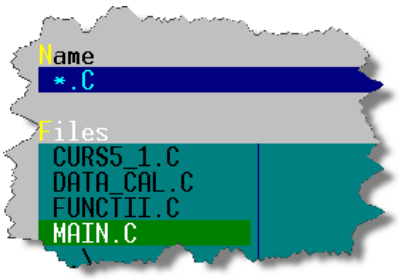


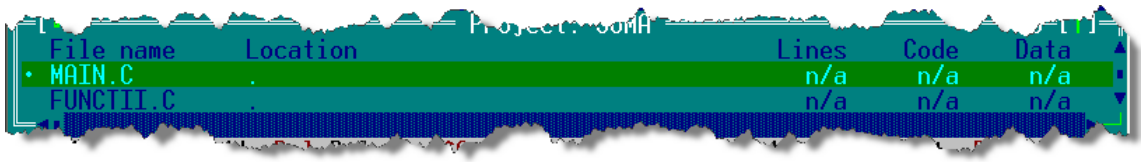
B) Stabilirea numelui proiectului - suma.prj:



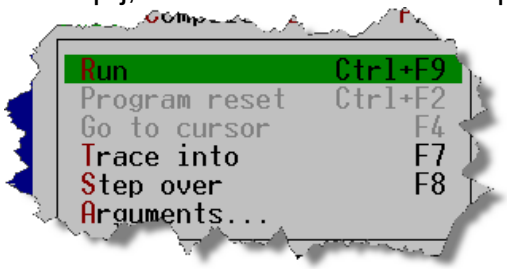
C) Adăugarea celor două fișiere sursa C în proiectul suma.prj:



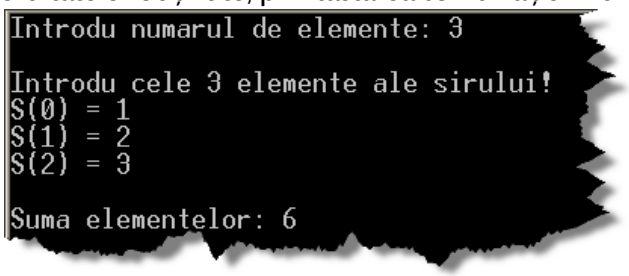




D) Rularea proiectului suma.prj, asemănător cu rularea unei aplicații simple:



E) Vizualizarea rezultatelor obținute, prin tastarea combinației Alt + F5:



PROBLEME REZOLVATE

1. Sortarea matricilor utilizând indecși (pointeri):

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

#define DEBUG 1

/*
 * { 2, 0, 17 } */
int  *idx[100];
/*
 * "Vasile POPESCU\", " ... "  */
char tbl[100][100];

/*
* Citeste matricea - textul  */
int fCitesteTabela( void )
{
     int i = 0;

     printf( "\nIntrodu inregistrarile...\n" );
     do {
          printf( ">" );      // afișează un prompter
          fflush( stdin );    // golește bufferul de tastatură
```

```
          gets( tbl[ i++ ] );
     } while( strlen( tbl[ i-1 ] ) != 0 );

     return i;
}

/*
 * Initializeaza sistemul de index */
void fStabilireIndex( int n )
{
     int i, flag;
     char *tmp;

     /*
      * INITIALIZAREA */
     for( i = 0; i < n; i++ )
          idx[ i ] = tbl[ i ];
          /*
           * Sortarea Bubble-Sort */
     do
     {    flag = 0;
          for( i = 0; i < n-1; i++ )
               if( strcmp( idx[ i ], idx[ i+1 ] ) > 0 )
               {    tmp = idx[ i ];
                    idx[ i ] = idx[ i+1 ];
                    idx[ i+1 ] = tmp;
                    flag = 1;
               }
     } while( flag );
}

/*
 * Afisarea tabelei INDEXATE */
void fAfisareIdx( char *mat[], int n )
{
     int i;

     for( i = 0; i < n; i++ )
          printf( "\n%s", mat[ i ] );
}

/*
 * Afisarea tabelei ORIGINALE */
void fAfisareTbl( char mat[][100], int n )
{
     int i;

     for( i = 0; i < n; i++ )
          printf( "\n%s", mat[ i ] );
}

/*
 * Functia MAIN */
int main( void )
{
     int N;
```

```
    system( "cls" );          // clrscr();
    printf("[START]\n");

    if( DEBUG )
        printf( "\n[CITESTE] Matricea (tabela)" );
    /*
     * Citește "tabela virtuala" */
    N = fCitesteTabela();
    /*
     * Indexează matricea */
    if( DEBUG )
        printf( "\n[INDEX] Creare pe (tabela)" );
    fStabilireIndex( N );
    /*
     * Afișează conținutul "tabelei" */
    if( DEBUG )
        printf( "\n[AFISARE] (tabela) originala" );
    fAfisareTbl( tbl, N );
    /*
     * Afișarea conținutului "tabelei" după index */
    if( DEBUG )
        printf( "\n[AFISARE] (tabela) indexata" );
    fAfisareIdx( idx, N );
    getch();

    return 0;
}
```

```
START.
CALCUL COMBINARI:
        - CombinariF( 7, 3 ) = 35
        - CombinariR( 7, 3 ) = 35
END.

Process returned 0 (0x0)   execution ti
```

PROBLEME PROPUSE SPRE REZOLVARE

1. Dezvoltați funcția „myprintf(char *, ...)”, prezentată în cadrul cursului, astfel încât să poată opera cu toate tipurile fundamentale din C. În acela și timp, funcția va fi capabilă să accepte un minim de formatare (%4d, %5.1f, ...).
2. Creați o funcție ce va primi ca și parametru adresa unei funcții, pentru care va calcula integrala pe un domeniu specificat. Se va folosi noțiunea de pointer la funcție.
3. Aplicația rezolvată în cadrul laboratorului va fi construită sub forma de proiect, în mediul Borland C++ 3.1.
4. Modificați aplicația din cadrul laboratorului, astfel încât în matrice să putem memora, separat, numele și prenumele unei persoane, iar indexarea să se poată realiza după ambele câmpuri.