

Lab 1 – P2

Cuprins

1. Crearea unui proiect în MS Visual Studio 2008.....	1
2. Adăugarea unui fișier la proiect	4
3. Scrierea programului.....	6
4. Compilarea, rularea și detectarea erorilor.....	6
5. Scrierea / citirea cu <i>cin</i> și <i>cout</i>	7
6. Operatorii new și delete	9
7. Un program mai complex.....	9
8. Depanare.....	11
9. Documentație	17
10. Exerciții	17

1. Crearea unui proiect în MS Visual Studio 2008

Visual Studio 2008 este mediul de dezvoltare integrat (**IDE – Integrated Development Environment**) ce se va folosi pentru scrierea programelor C++. Mediul permite dezvoltarea de aplicații în mai multe limbaje de programare, de diverse tipuri. În cadrul laboratorului se vor scrie programe C++ de tip consolă. Pentru a scrie un program în VS2008, la început trebuie să creăm un proiect.

Pentru a crea un proiect, se vor parcurge următorii pași :

1. Se lansează în execuție mediul de dezvoltare VS2008
2. În meniul principal, *File* → *New* → *Project...* figura 1.1:

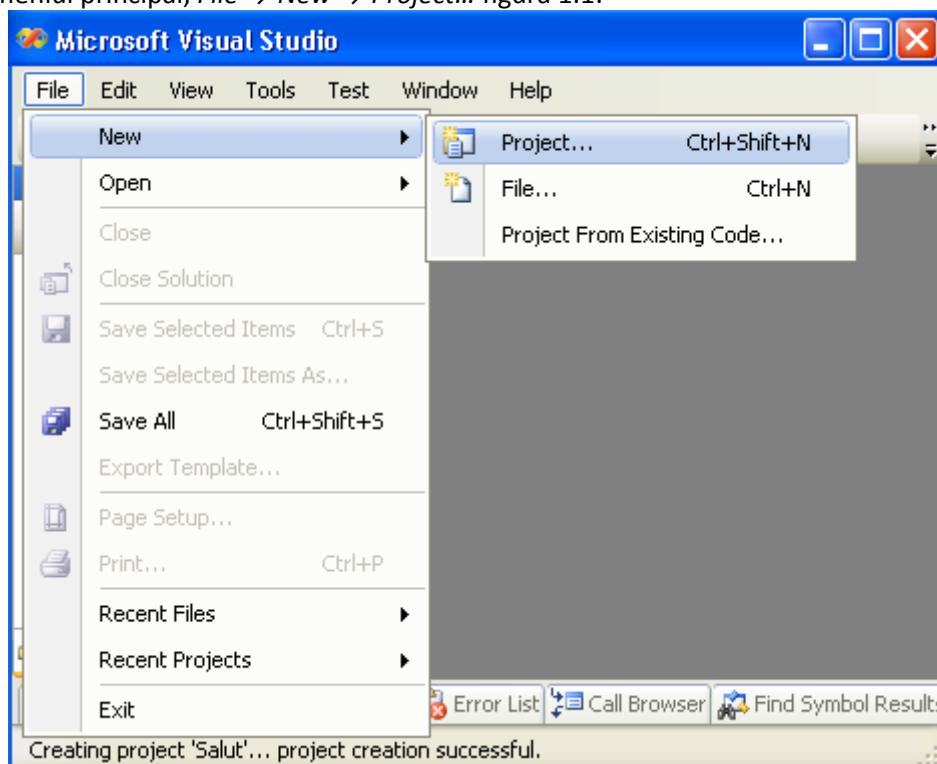


Fig. 1.1 Crearea unui nou proiect

3. În fereastra *New Project*, în stânga se observă caseta *Project types* (figura 1.2). Se alege *Other languages* → *Visual C++*

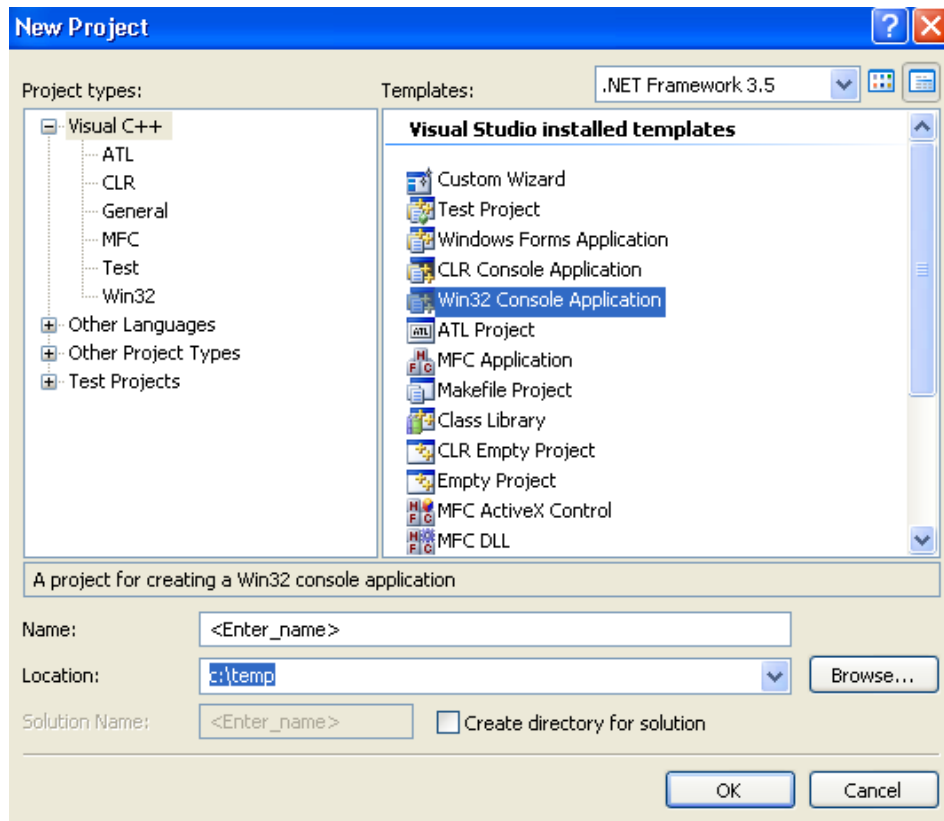


Fig. 1.2 Fereastra New Project

4. În partea dreaptă se observă caseta *Templates* din care se alege *Win32 Console Application*.
5. În partea de jos se observă casetele *Name* și *Location*. Se alege un nume pentru proiect. La *Location* se selectează directorul de lucru: *C:\temp* sau *D:\temp* după caz. (figura. 1.2)
6. Se apasă *OK*.
7. În continuare, apare fereastra *Welcome to the Win32 Application Wizard* (figura 1.3). Se apasă *Next*, **NU** *Finish*!

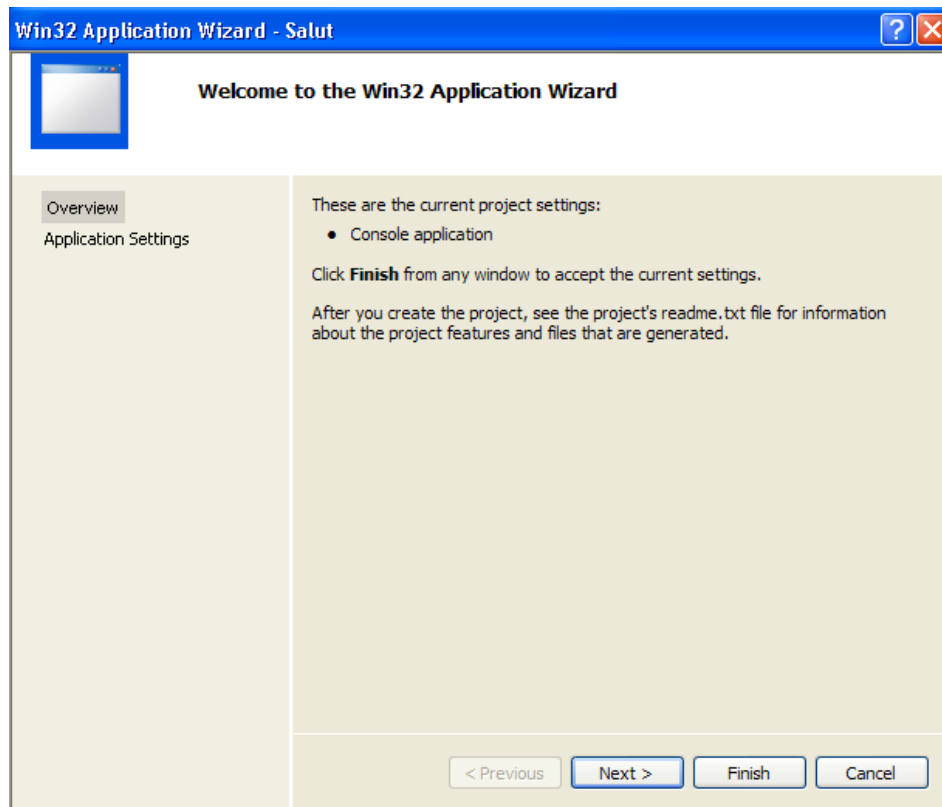


Fig. 1.3 Fereastra Welcome...

8. În noua fereastră apărută, la rubrica *Additional options*, se bifează *Empty project*. Se lasă celelalte setări neschimbate (figura 1.4):

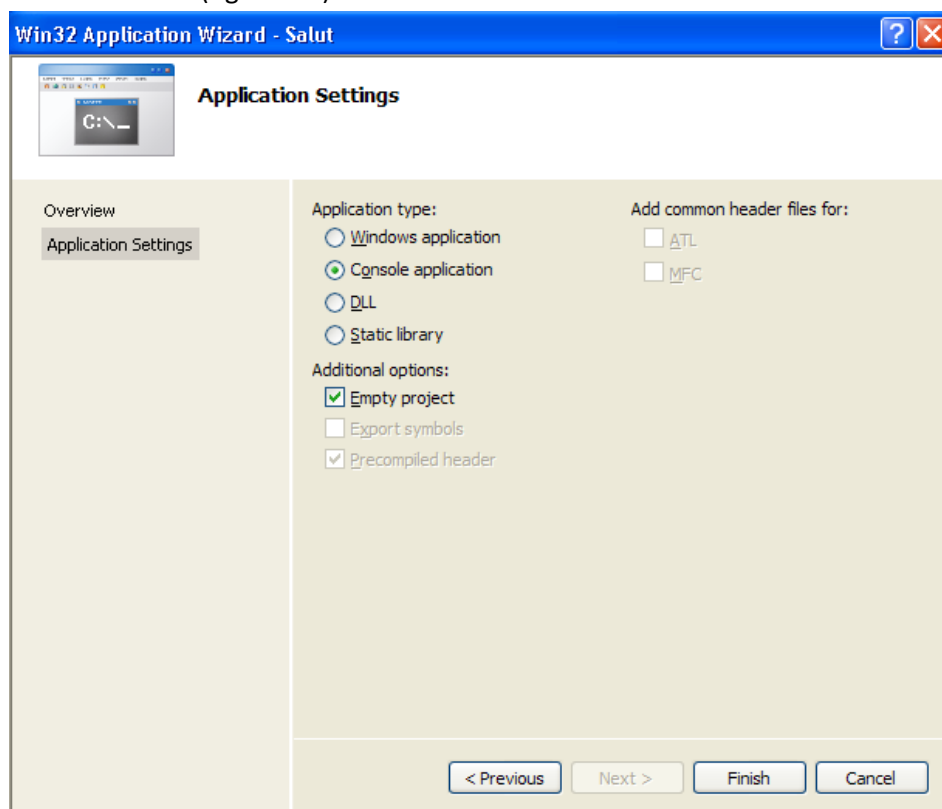


Fig.1.4 Fereastra Application Settings

9. Se apasă *Finish*.

Proiectul este creat și deschis în mediul de dezvoltare (figura 1.5).

Se pot observa următoarele ferestre:

- *Solution explorer* – în partea stânga. De aici se pot crea sau deschide fișierele proiectului. Inițial proiectul nu conține nici un fișier.
- *Start Page* – în restul ecranului. Această fereastră nu este utilă, ea poate fi închisă.

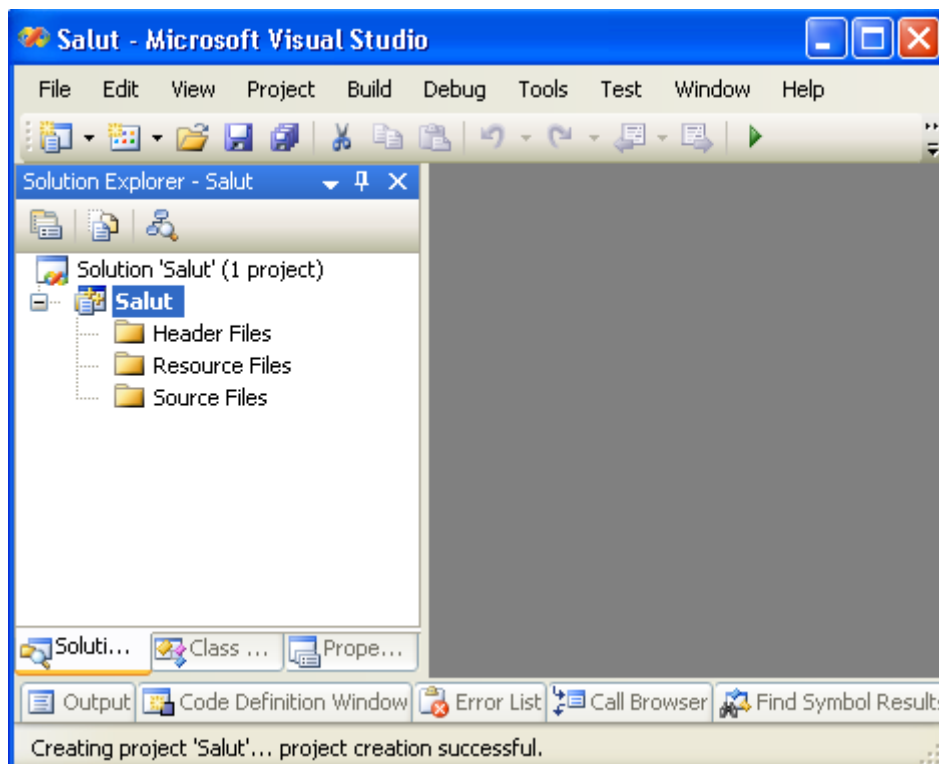


Fig. 1.5 Proiectul nou creat

2. Adăugarea unui fișier la proiect

Pentru a scrie un program în VS2008, trebuie adăugat un fișier sursă la proiect. Pentru aceasta se vor efectua următorii pași:

1. În *Solution Explorer*, click dreapta pe grupul *Source Files* → *Add* → *New Item...* (figura 1.6)

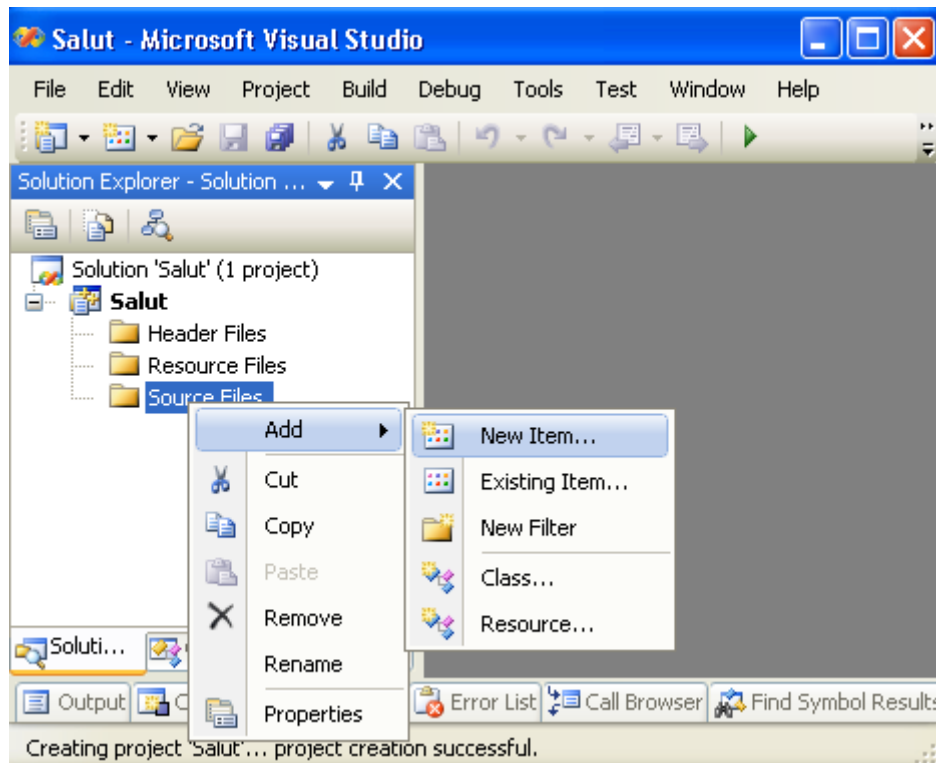


Fig. 1.6 Adăugare fișier sursă

2. Apare fereastra *Add New Item* (figura 1.7).
3. În caseta *Templates* se alege *C++ File (.cpp)*
4. La *Name* se introduce numele fișierului. Ca regulă ne scrisă, se va denumi fișierul care conține funcția `main()` **<numeProiect>Main**; în cazul de față – **salutMain**.

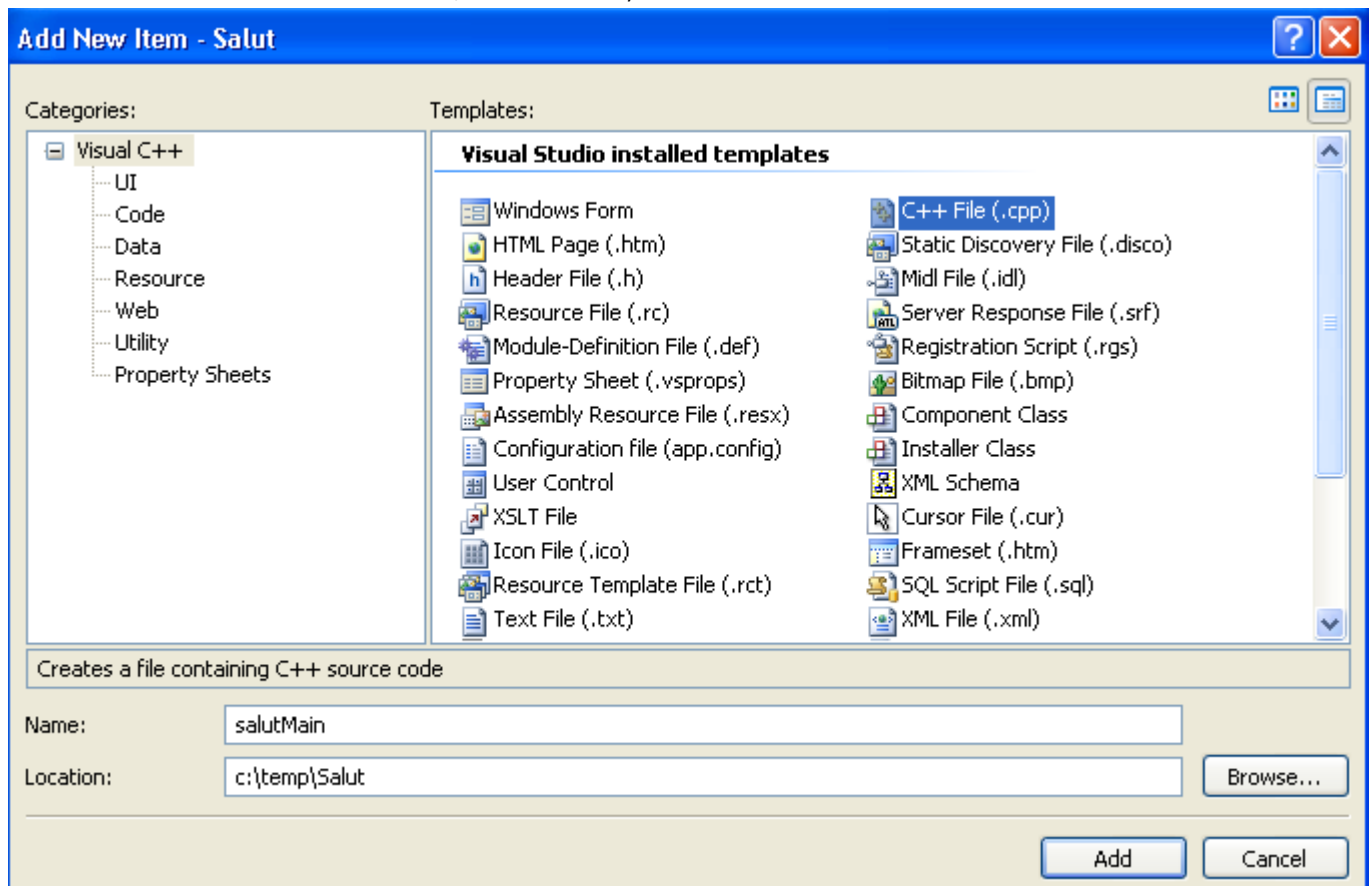


Fig. 1.7 Adăugare fișier sursă – partea 2

5. Se apasă *Add*. Noul fișier va fi creat și deschis în editor.

3. Scrierea programului

Se va scrie un program simplu care va afișa un mesaj la consolă (vezi figura 1.8):

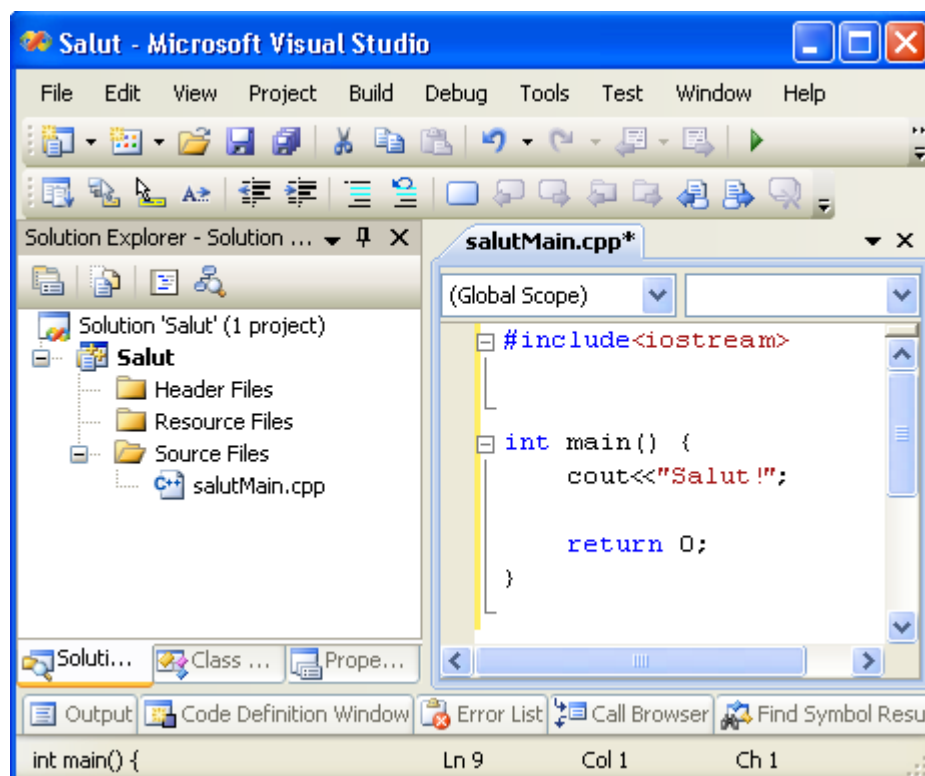


Fig. 1.8 Scrierea codului sursă în editor

În Visual Studio codul este formatat în mod automat în timp ce este scris. Poate fi formatat și ulterior apăsând **Ctrl+A**, iar apoi **Ctrl+K**, **Ctrl+F**.

4. Compilarea, rularea și detectarea erorilor

Pentru a compila proiectul se apasă **Ctrl+Alt+F7**, sau din meniul principal se selectează *Build* → *Rebuild Solution*.

Dacă sunt detectate erori de compilare acestea vor fi afișate în fereastra *Error List*. Se va introduce intenționat o eroare pentru a vedea facilitățile acestei ferestre. Astfel va înlocui linia `cout<<"Salut!";` cu `cout<<"Salut!"<<x;`. La compilare, deoarece variabila `x` nu este declarată se va genera o eroare și se va afișa în fereastra *Error List* următorul mesaj (figura 1.10):

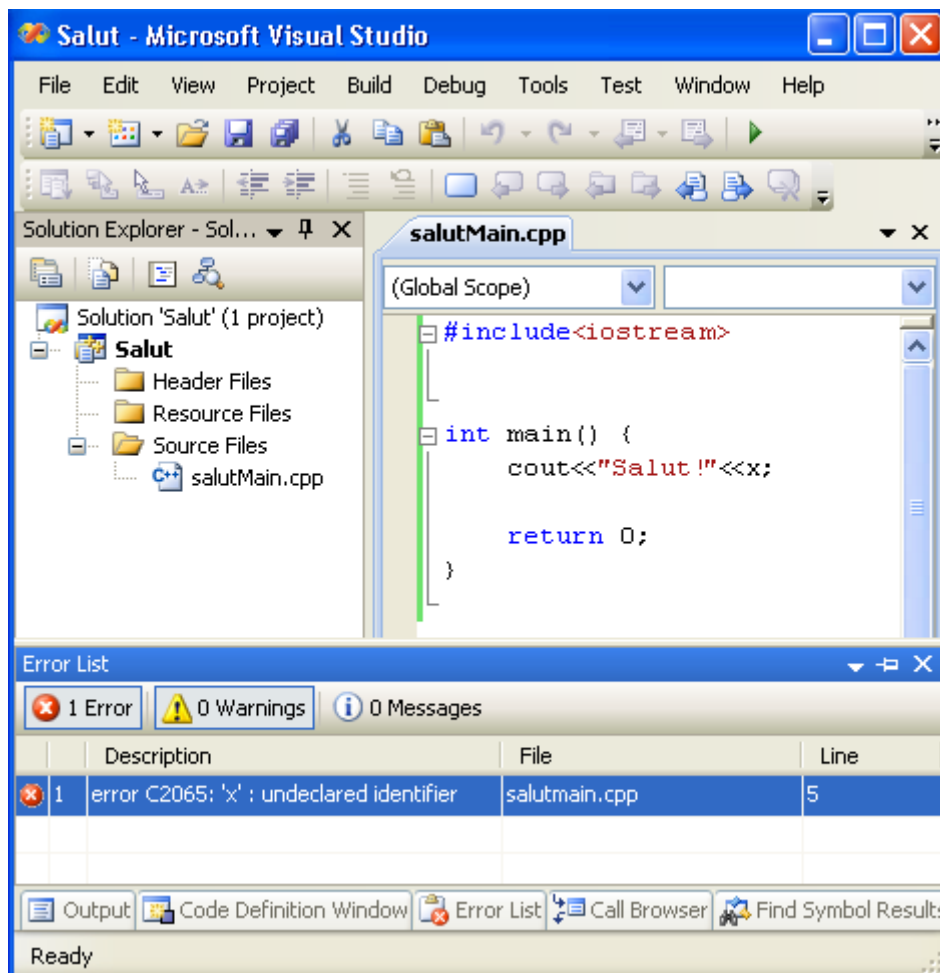


Fig. 1.10 Mesaj de eroare

Prin dublu-click peste mesajul de eroare, cursorul se va deplasa la linia de cod care conține eroarea respectivă. Alte tipuri de erori pot duce la mesaje de eroare neclare și chiar linia de cod a erorii poate fi indicată greșit.

În general se aplică următoarea regulă: eroarea trebuie căutată ori pe linia afișată în *Error List*, ori cu o linie mai sus. De exemplu, dacă în program, se șterge caracterul „;” de la sfârșitul uneia dintre linii, se observă că eroarea este localizată cu o linie mai jos.

Chiar dacă, după compilare, în fereastra *Error List* nu apar erori, dar apar warning-uri, acestea trebuie eliminate din codul sursă. De cele mai multe ori warning-urile duc la erori de execuție și se recomandă eliminarea acestora înainte de rularea proiectului.

Pentru a executa programul se apasă tasta **F5**. Se deschide o fereastră de tip consolă în care rulează programul.

5. Scrierea / citirea cu *cin* și *cout*

În C++ s-a elaborat o modalitate mai simplă de scriere/citire la/de la consolă având același efect ca și funcțiile de bibliotecă `scanf()`/`printf()` din C. La începutul execuției fiecărui program sunt instanțiate automat două variabile globale speciale *cin* și *cout*. Ele sunt folosite pentru citirea, respectiv scrierea la/de la consolă.

Pentru a citi o variabilă de la consolă, se va scrie următoarea sintaxă:

```
int a;
cin >> a;
```

Operatorul >> are un rol special pentru variabila `cin`. Expresia:

```
cin >> a;
```

semnifică faptul că de la consolă este citită o valoare și depozitată în variabila `a`. Tipul variabilei din dreapta poate fi oricare din tipurile fundamentale: `int`, `char`, `float`, `double` sau `char*` care reprezintă un șir de caractere. Pentru fiecare tip enumerat mai sus, implicit citirea se va face corect.

Pentru a scrie o variabilă la consolă, folosim sintaxa:

```
char str[] = "abc";  
cout << str;
```

În mod similar, operatorul << are o semnificație specială pentru variabila `cout`. Expresia:

```
cout << str;
```

semnifică faptul că variabila `str` este scrisă la consolă. Variabilele scrise pot avea aceleași tipuri ca și cele citite cu `cin`. Aceste tipuri au fost enumerate mai sus pentru variabila `cin`.

Se observă că în exemplul de mai sus a fost scrisă la consolă o variabilă de tip `char[]`, tip care nu a fost menționat în lista de tipuri suportate pentru operandul dreapta. Totuși, utilizarea lui a fost posibilă într-o expresie cu `cout`. De ce?

Variabilele `cin` și `cout` sunt definite în header-ul `<iostream>`. Pentru a fi utilizate trebuiesc adăugate la începutul programului următoarele linii:

```
#include <iostream>  
using namespace std;
```

Aceste variabile speciale(`cin` și `cout`) sunt **obiecte**. Obiectele vor fi studiate în detaliu ulterior.

Iată un exemplu complet folosind noile facilități de scriere/citire din C++:

```
// exemplu: cin si cout  
#include <iostream>  
using namespace std;  
  
int main() {  
    int iVal;  
    char sVal[30];  
  
    cout << "Introduceti un numar: ";  
    cin >> iVal;  
    cout << "Si un sir de caractere: ";  
    cin >> sVal;  
    cout << "Numarul este: " << iVal << "\n"  
        << "Sirul este: " << sVal << endl;  
    return 0;  
}
```

Exemplu de rulare:
Introduceti un numar: 12 Si un sir de caractere: abc Numarul este: 12 Sirul este: abc

Un element nou este cuvântul `endl`. Acesta este o funcție specială numită și manipulator care trimite o nouă linie (echivalent cu `"\n"`) la ieșirea standard, golind și bufferul acesteia.

Atât expresiile cu `cin` cât și cele cu `cout` pot fi înlănțuite.

Expresia

```
cout << a << " " << b;
```

este echivalentă cu

```
cout << a;
cout << " ";
cout << b;
```

Comparativ cu funcțiile `printf()` / `scanf()` din C, expresiile cu `cin` și `cout` sunt mai simple și mai ușor de înțeles. Nu mai sunt necesari specificatorii de format. Dezavantajul constă în faptul că nu se pot face afișări formatare pe un anumit număr de caractere. O afișare de genul:

```
printf("%7.2f", f);
```

nu are echivalent folosind `cout`, decât dacă se folosesc manipulatori speciali pentru formatare.

6. Operatorii `new` și `delete`

În C++, lucrul cu memoria dinamică este facilitată de doi operatori speciali: `new` și `delete`.

Alocarea dinamică de memorie se face cu operatorul `new`. El returnează un pointer către începutul blocului de memorie alocat. Sintaxa operatorului este următoarea:

```
TIP *p, *pvector;
p = new TIP;
pvector = new TIP[nr_elemente];
```

De exemplu, pentru pointeri la tipurile fundamentale, alocarea se va face ca în exemplul următor :

```
int *pi;
pi = new int;
char *pch;
pch = new char;
```

Pentru pointeri la vectori alocarea se face după cum urmează:

```
int *vi;
vi = new int[10];
char *psir = new char[80];
```

Atunci când nu mai avem nevoie de o variabilă alocată dinamic, aceasta trebuie dealocată. Memoria dealocată devine disponibilă pentru noi cereri de alocare. Pentru dealocări se folosesc operatorii `delete` – pentru variabile de tip fundamental și `delete[]` – pentru vectori. Exemplu:

```
delete pi;
delete pch;
delete[] vi;
delete[] psir;
```

7. Un program mai complex

Fie următorul program: să se citească de la consolă un vector de `n` șiruri de caractere, se alocă spațiu de memorie strict necesar și sortează șirurile în ordine crescătoare.

În Visual Studio, se poate crea o soluție nouă (o soluție reprezintă o mulțime de proiecte deschise simultan), sau se poate adăuga un proiect nou în cadrul soluției existente. Se va crea un proiect nou și se vor adăuga trei fișiere: *SortareSiruri.h*, *SortareSiruri.cpp*, *SortareSiruriMain.cpp*. Codul sursă este prezentat mai jos.

SortareSiruri.h

```
#ifndef _SortareSiruri_
#define _SortareSiruri_
```

```

char **citireVSiruri(int n);
void sortareVSiruri(char ** vsiruri, int n);
void afisareVSiruri(char **vsiruri, int n);
void dealocareVSiruri(char **vsiruri, int n);

#endif

```

SortareSiruri.cpp

```

#include <iostream>
#include<string.h>
#include"SortareSiruri.h"
using namespace std;

char **citireVSiruri(int n) {
    char buffer[100];
    char **vsiruri = new char*[n];
    cin.ignore(100, '\n');
    for(int i=0; i<n; i++) {
        int len;
        cin.getline(buffer, 100);
        len = strlen(buffer);
        vsiruri[i] = new char[len + 1];
        strcpy(vsiruri[i], buffer);
    }
    return vsiruri;
}

void sortareVSiruri(char ** vsiruri, int n) {
    int suntPerm = 1;
    while(suntPerm) {
        suntPerm = 0;
        for(int i=0; i<n-1; i++) {
            if(strcmp(vsiruri[i],
                vsiruri[i+1]) > 0) {
                char *aux = vsiruri[i];
                vsiruri[i] = vsiruri[i+1];
                vsiruri[i+1] = aux;
                suntPerm = 1;
            }
        }
    }
}

void afisareVSiruri(char **vsiruri, int n) {
    cout << "Sirurile sortate sunt:" << endl;
    for(int i=0; i<n; i++) {
        cout << vsiruri[i] << endl;
    }
}

void dealocareVSiruri(char **vsiruri, int n) {
    for(int i=0; i<n; i++) {
        delete[] vsiruri[i];
    }
    delete[] vsiruri;
}

```

SortareSiruriMain.cpp

```
#include<iostream>
#include"SortareSiruri.h"
using namespace std;

int main() {
    int n;
    char** vsiruri;

    cout << "n=";
    cin >> n;
    vsiruri = citireVSiruri(n);
    sortareVSiruri(vsiruri, n);
    afisareVSiruri(vsiruri, n);
    dealocareVSiruri(vsiruri, n);
    return 0;
}
```

Acest program conține câteva elemente noi:

1. În fișierul *SortareSiruri.cpp*, avem următoarea linie în funcția *citireVSiruri()*:

```
cin.ignore(100, '\n');
```

Acest apel este necesar înainte de apelarea funcției *cin.getline()*, care citește o linie de caractere de la consolă. Rolul apelului este să ignore caracterele din bufferul de intrare rămase de la citirea anterioară. Pentru detalii, se poate studia documentația acestei funcții. Ea are același rol ca și funcția *fflush(stdin)* în C, cu observația că *fflush(stdin)* golește bufferul de intrare.

2. În aceeași fișier *SortareSiruri.cpp*, este apelul:

```
cin.getline(buffer, 100);
```

Funcția *cin.getline()* citește o linie de la consolă, sub forma unui șir de caractere și o depune în buffer. Al doilea-lea parametru – 100 – este numărul maxim de caractere care pot fi citite. Funcția este echivalentă cu *fgets()* din C.

A fost necesar utilizarea acestor funcții noi, în loc de *fflush()*, *fgets()* din C, deoarece nu este bine combinarea funcțiilor de lucru cu consola din C cu cele din C++. La utilizarea lor mixtă pot apărea incompatibilități.

Recomandare.

Toate programele realizate în cadrul laboratoarelor de POO vor fi concepute ca proiecte, respectând aceleași standarte ca și la obiectul Programarea Calculatoarelor. Adică unul sau mai multe fișiere header, unul sau mai multe fișiere cu funcții și un fișier cu funcția *main*.

8. Depanare

Depanarea (debug) este facilitatea oferită de mediile de dezvoltare de a analiza procesul de execuție a unui program în scopul de a detecta erorile. În modul de lucru *Debug*, se poate rula programul instrucțiune cu instrucțiune, urmărind valoarea unor variabile după fiecare instrucțiune executată. De asemenea, se pot stabili anumite linii de cod la care dorim ca programul să se oprească și astfel, să vizualizăm valoarea variabilelor alese de noi doar în acele puncte din program. Aceste linii la care se dorește întreruperea execuției programului se numesc puncte de oprire (breakpoint-uri).

De exemplu, se va considera cazul unei erori des întâlnite și se presupune că în programul *SortareSiruri.cpp* afișarea este greșită. Se va verifica în primul rând dacă alocarea memoriei și citirea a fost efectuată corect iar apoi, dacă sortarea a fost corectă. Pentru aceasta, se vor plasa două breakpoint-uri în funcția *main*, unul după funcția de citire și altul după funcția de sortare. Pentru plasarea unui breakpoint, se

va poziționa cursorul în dreptul liniei de cod la care se dorește oprirea execuției programului și se apasă tasta **F9**. Se poate scoate un breakpoint tot prin apăsarea tastei **F9**. În partea dreaptă a editorului va apare o bulină roșie pentru fiecare breakpoint astfel plasat(figura 1.11):

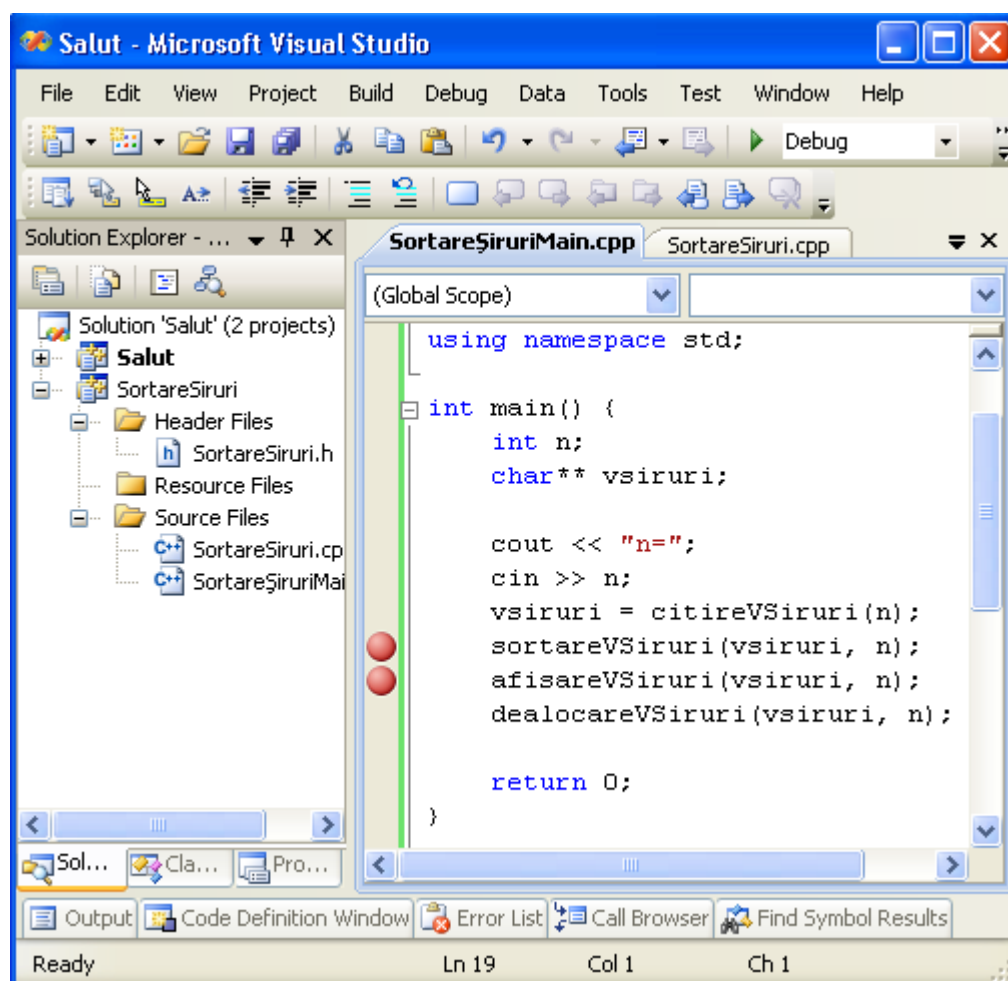


Fig. 1.11 Breakpoint-uri

În continuare, se apasă tasta **F5** pentru a rula programul în mod Debug. Se observă că programul își începe execuția normal, sunt cerute datele de intrare, iar după ce le introducem consola se blochează. În acel moment se revine la Visual Studio și se observă că aranjamentul ferestrelor s-a schimbat, iar în dreptul primului breakpoint a apărut o săgeată (figura 1.12):

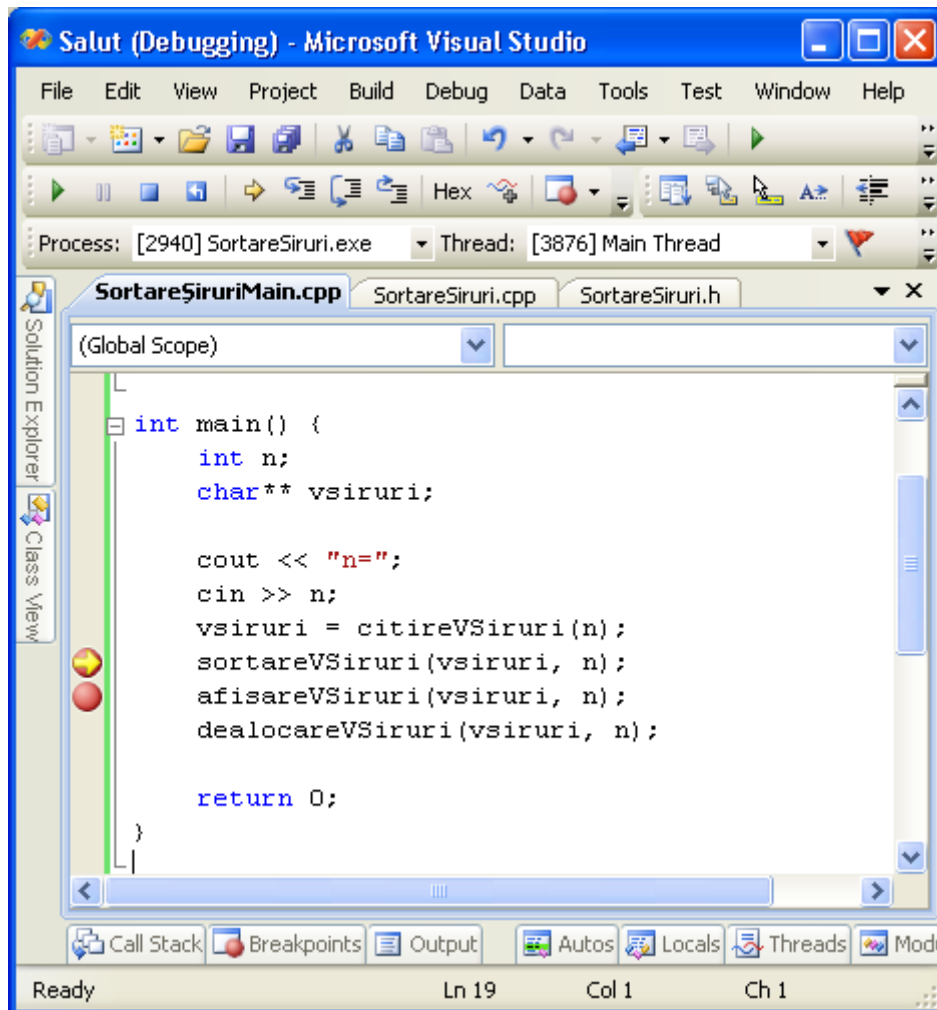


Fig. 1.12 Oprirea execuției la un breakpoint

Săgeata indică instrucțiunea la care s-a oprit execuția programului.

Atenție! Instrucțiunea la care se află săgeata încă nu s-a executat, dar instrucțiunea anterioară a fost executată!

În acest moment, pot fi vizualizate valorile unor variabile din program. În mod evident, interesează valoarea celor două variabile definite în main: `n` și `vsiruri`. Pentru a le vizualiza, alege meniul principal → *Debug* → *Windows* → *Watch* → *Watch 1*.

În partea de jos a mediului de dezvoltare, apare fereastra *Watch 1* (figura 1.13). În această fereastră, se dă click pe coloana *Name* și se introduc numele variabilelor ce se doresc a fi vizualizate – întâi `n`, se apasă ENTER, pe urmă `vsiruri`, se apasă din nou ENTER:

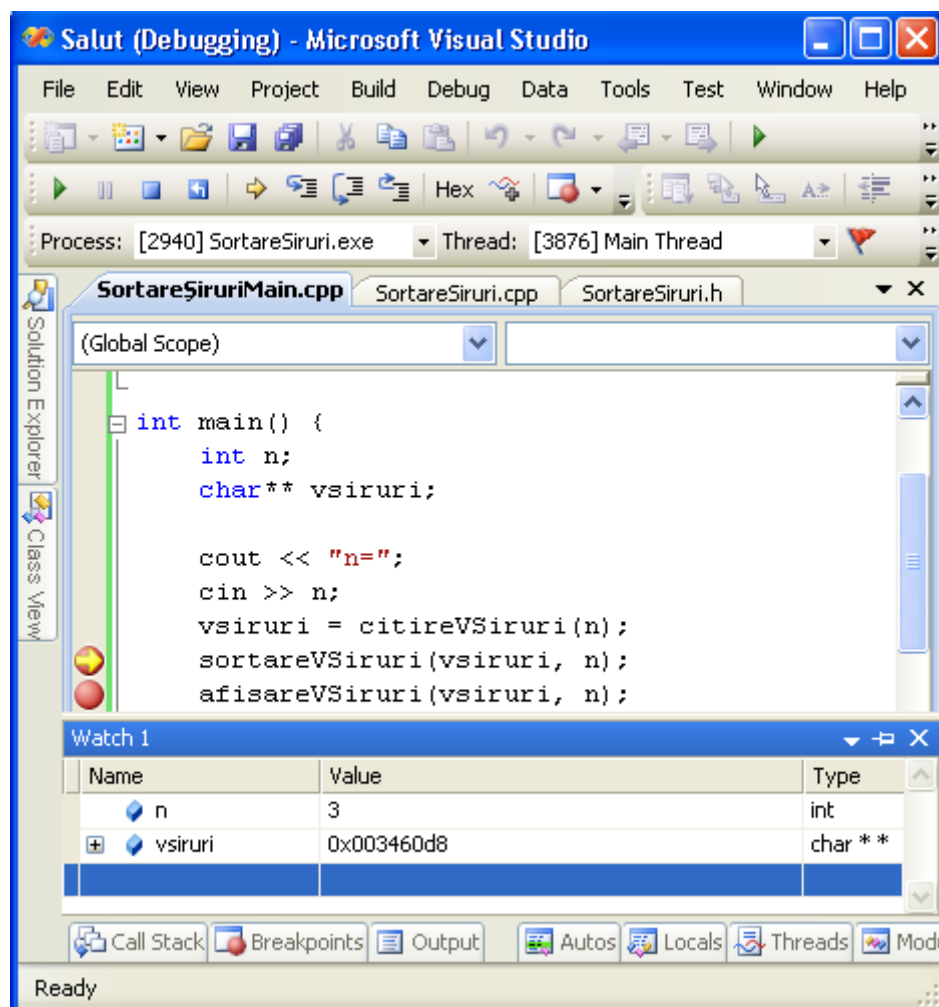


Fig. 1.13 Fereastra Watch

În a doua coloană (*Value*), va fi afișată valoarea variabilelor la acel moment al execuției programului, iar în cea de-a treia coloană (*Type*), tipul acestora. La variabila *n*, valoarea este un număr în baza zece. Însă la variabila de tip pointer la pointer, *vsiruri*, IDE-ul afișează o valoare în hexazecimal ce reprezintă adresa de memorie a primului element din șir și care nu este așa de utilă.

Fereastra *Watch* permite vizualizarea nu doar a variabilelor, ci și a expresiilor. De exemplu, se poate vizualiza elementele vectorului *vsiruri* (figura 1.14):

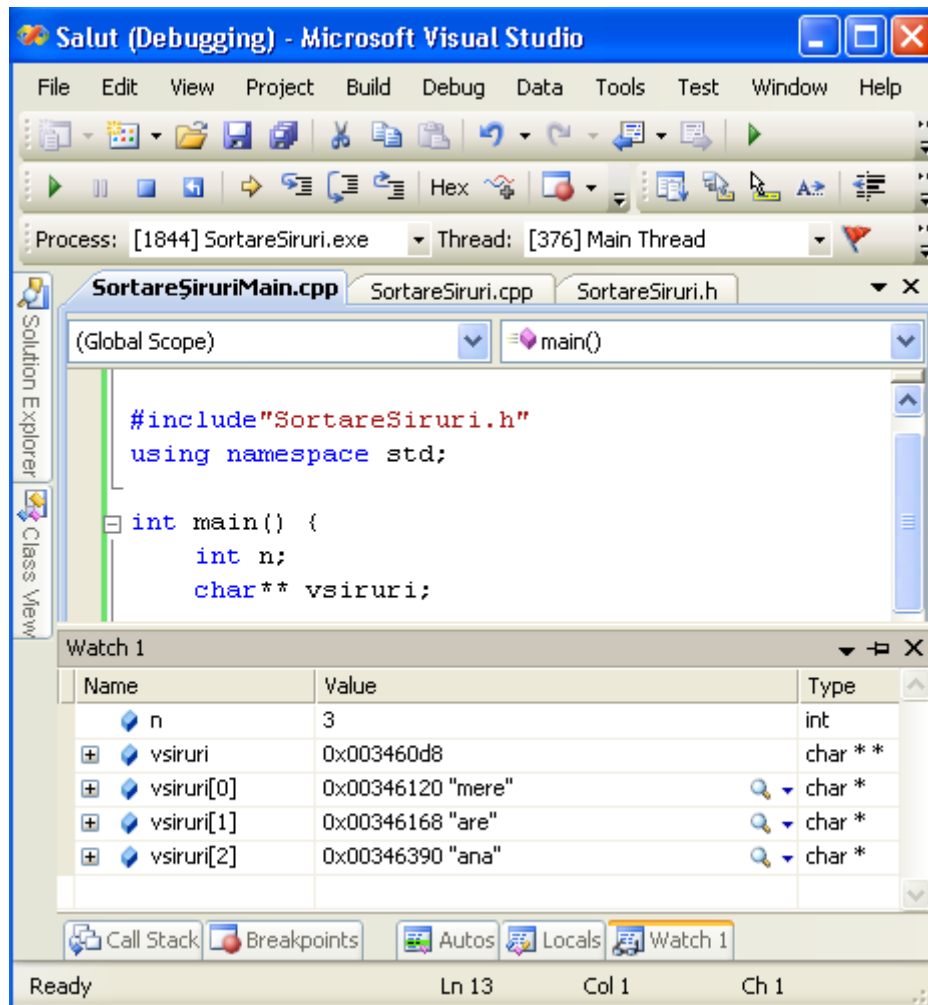


Fig. 1.14 Vizualizarea elementelor unui vector

Se observă că de data aceasta fiecare șir de caractere este afișat corect.

Citirea s-a efectuat așa cum era de așteptat. Până în acest punct programul se execută corect. Se apăsăm tasta **F5** pentru a continua execuția programului până la următorul breakpoint. Se observă următoarele (figura 1.15):

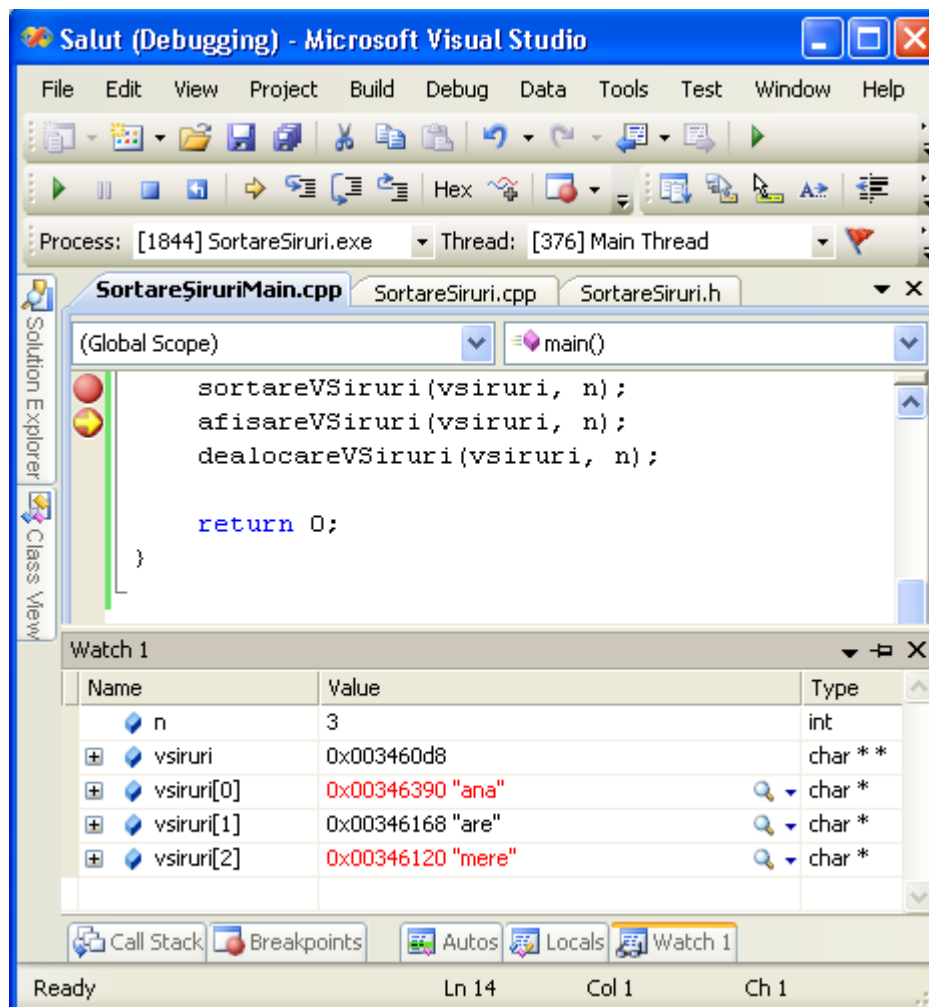


Fig. 1.15 Breakpoint după sortare

Ordinea elementelor în `vsiruri` s-a schimbat, elementele sunt sortate crescător, așa cum era de așteptat. Unele dintre elementele schimbate sunt afișate cu roșu. Atât citirea cât și sortarea sunt corecte.

Explorați meniul *Debug* în timp ce vă aflați într-un breakpoint pentru a afla și alte facilități de debug.

Mai jos sunt prezentate combinații de taste utile în Visual Studio 2008.

Combinatia de taste	Efect
Ctrl + C	Copy - copiere
Ctrl + V	Paste - afișare
Ctrl + A	Select all – selectare totală
Ctrl + K, F	Format selected – formatare selecție
Ctrl + A, K, F	Format all – formatare totală
Ctrl + Shift + B	Build all – compilare proiect
F5	Debug, continue after breakpoint – Intrare în depanare sau continuarea execuției după breakpoint
F9	Insert / remove breakpoint Adăugarea / eliminarea breakpoint

9. Documentație

Sursa recomandată pentru documentare este situl <http://www.cplusplus.com/>.

În particular, sunt recomandate următoarele link-uri:

- <http://www.cplusplus.com/reference/clibrary/> unde se poate găsi documentația completă a tuturor fișierelor de bibliotecă din C și a funcțiilor din acestea. Documentația este similară cu cea din Borland C.
- <http://www.cplusplus.com/reference/iostream/> unde se poate găsi documentația claselor ce realizează operații de intrare/ieșire (intrare/ieșire standard – `cin`, `cout` și fișiere).
- <http://www.cplusplus.com/doc/tutorial/> unde se poate găsi un material didactic alternativ despre limbajul C++.

10. Exerciții

1. Creați un nou proiect care să conțină programul prezentat în secțiunea 7.
2. Compilați și rulați programul de la exercițiul 1.
3. Executați pas cu pas programul.
4. Introduceți în mod intenționat o eroare în program, comentând linia:

```
suntPerm = 1;
```

din funcția `sortareVSiruri()`.

Plasați două breakpoint-uri conform indicațiilor din secțiunea 8. Testați programul cu o intrare de cinci șiruri și verificați dacă la al doilea breakpoint șirurile sunt sau nu sunt sortate corect.

5. Încercați să detectați eroarea, presupunând că nu știți unde este. Localizați prima linie de cod care se execută după ce se detectează că două șiruri trebuie inversate. Introduceți un breakpoint pe acea linie. Vizualizați toate variabilele locale și cele două șiruri care urmează să fie inversate. Ce expresii veți introduce în fereastra *Watch* pentru cele două șiruri?

De câte ori ar trebui să se realizeze inversarea șirurilor și de câte ori are loc în realitate? Ce puteți spune despre comportamentul programului?

6. Introduceți un breakpoint pe linia `while` și demonstrați cu ajutorul lui că bucla `while` se execută doar o dată în programul eronat.
7. Corectați eroarea în program. Numărați de câte ori se execută bucla `while` și de câte ori are loc inversarea a două șiruri consecutive în funcția de sortare.