POO – C++ - Laborator 1
Recapitularea unor elemente ale limbajului C

1. Structuri

Definire

Se foloseste cuvantul cheie *struct* urmat de numele structurii si enumerarea membrilor acesteia:

```
struct X
{
        int i;
        char c;
};
```

Utilizare

Elementele membre ale variabilelor de tip structura pot fi accesate folosind operatorul **.**:

```
struct X x1, x2[4];
x1.i = 4;
x2[2].c=getch();
```

2. Pointeri

Declarare

```
int *pi;  /*  pointer la o variabila de tip int                */
char *pc;       /*  pointer la o variabila de tip char              */
struct X *px;   /*  pointer la o variabila de tip struct X  */
struct X x1, x2[4];
```

Utilizare

```
pi=&x1.i;       /*  atribuirea adresei unei variabile de        */
px=&x2[2];      /*  acelasi tip cu al pointerului               */
pc=&px->c;
*pi=4;          /*  referirea zonei de memorie adresate de      */
*pc=getch();    /*  pointer cu ajutorul operatorului *          */
```

3. Alocare dinamica

Exista aplicatii în care necesarul de memorie nu este cunoscut din faza de compilare si rezervarea unor zone de memorie de dimensiuni acoperitoare pentru toate datele ar depasi capacitatea memoriei disponibile. Singura solutie în acest caz o reprezinta alocarea/eliberarea interactiva de zone de memorie chiar în timpul executiei programului - **alocarea dinamica** de memorie. În acest scop, în programele C se folosesc functii de biblioteca ale caror prototipuri (descrieri ale tipului si argumentelor functiilor) se gasesc în fisierul header **alloc.h**. Cele mai utilizate functii pentru alocarea de memorie sunt:

```
void* malloc(unsigned size);
void* calloc(unsigned nelem, unsigned size);
```

Prima functie primeste ca unic argument numarul de octeti ce trebuie alocati si returneaza adresa de început a zonei de memorie alocate în caz de succes sau *NULL* (0) în caz de esec. A doua functie se comporta identic, încercând însa alocarea a *nelem* blocuri succesive de *size* octeti.

Toate alocarile dinamice se fac într-o zona de memorie destinata special acestui scop, zona numita **heap** (gramada). În functie de modelul de memorie folosit, dimensiunea heap-ului variaza de la dimensiunea

unui segment (64KO) minus dimensiunea programului pâna la dimensiunea întregii memorii disponibile minus aceeasi dimensiune a programului.

Pentru eliberarea unei zone de memorie alocate dinamic se foloseste functia complementara

void free(void addr);*

unde *addr* reprezinta un pointer ce contine adresa de început a unei zone de memorie alocata dinamic.

```
int *pi;
struct X *px;
…
/* alocare memorie pentru o variabila de tip int */
pi = (int *)malloc(sizeof(int));
*pi=3;

/* alocare pentru un vector de 4 elemente de tip struct X */
px=(struct X *)calloc(4, sizeof(struct X));
px[2]->i=4;
…
/* eliberare memorie */
free(pi);
free(px);
```

4. Crearea unui proiect în MS Visual Studio 2013

Visual Studio 2013 este mediul de dezvoltare integrat (***IDE – Integrated Development Environment***) ce se va folosi pentru scrierea programelor C++. Mediul permite dezvoltarea de aplicaţii 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 VS2013, la început trebuie să creem un proiect.

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

1. Se lansează în execuţie mediul de dezvoltare VS2013
2. În meniul principal, *File → New → Project…* figura 1.1:

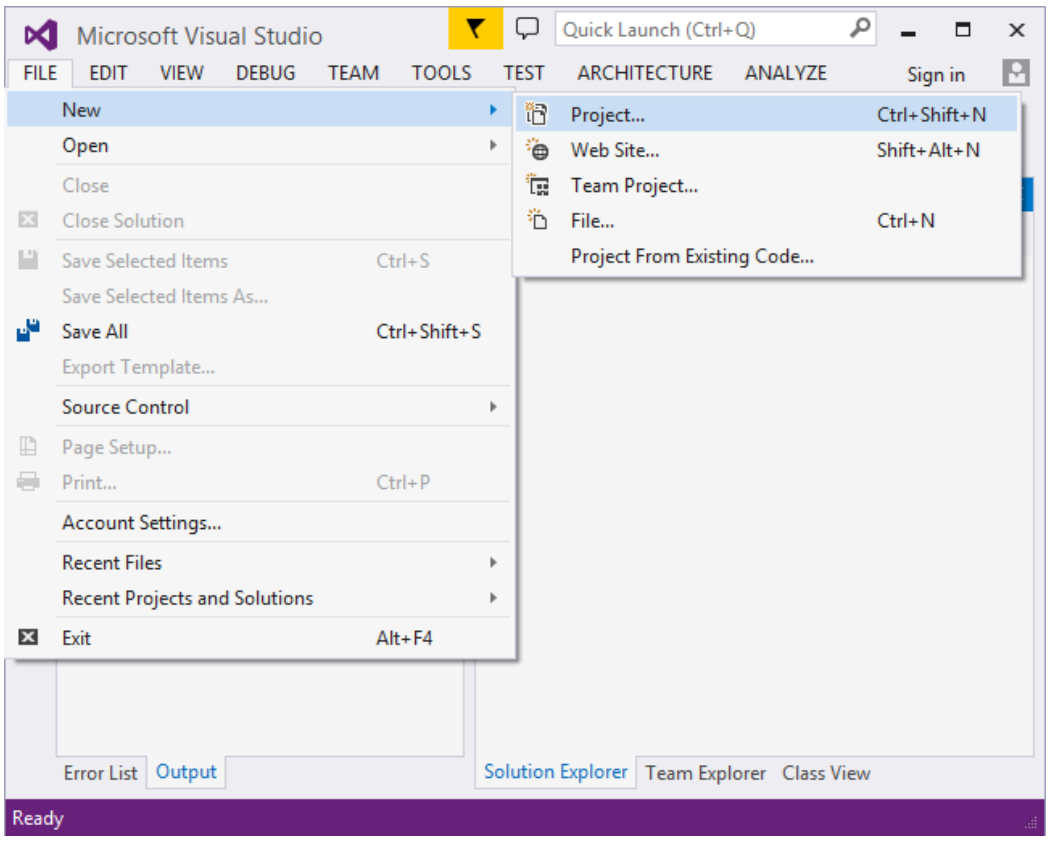

Fig. 1.1 Crearea unui nou proiect

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

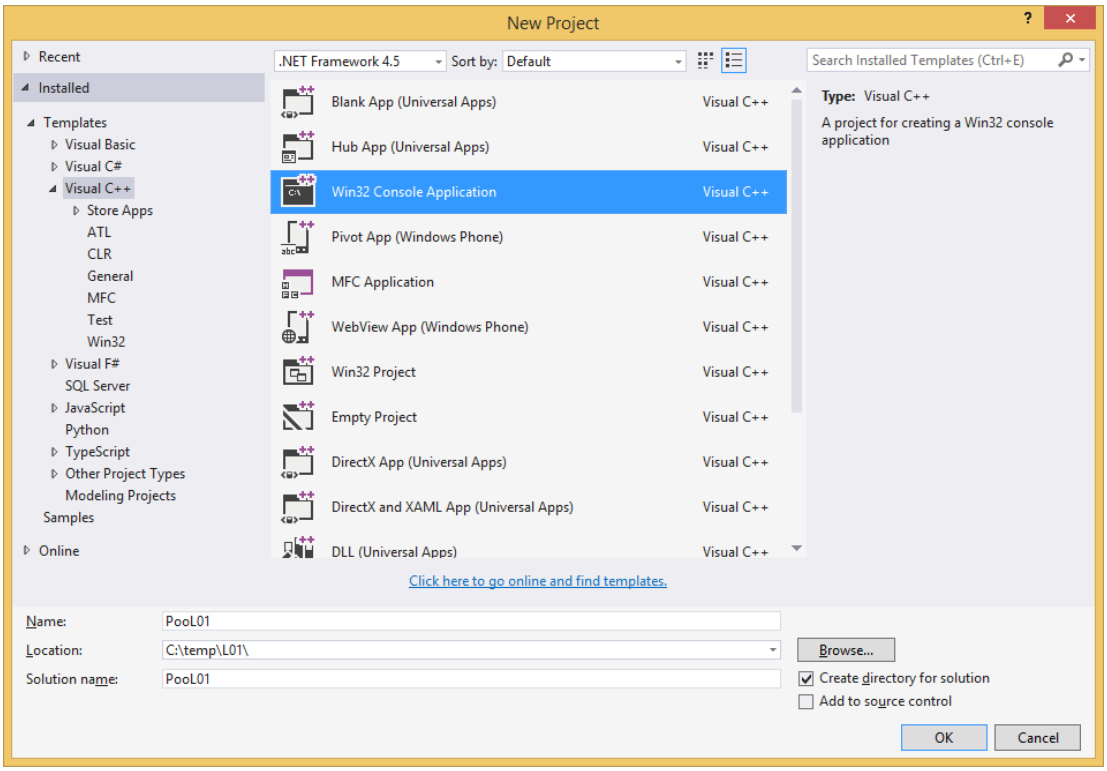

Fig. 1.2 Fereastra New Project

4. În partea dreaptă se observă caseta *Templates* din care se alege *Win32 Console Application*.

5. În partea de jos se observă casetele *Name* şi *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*!

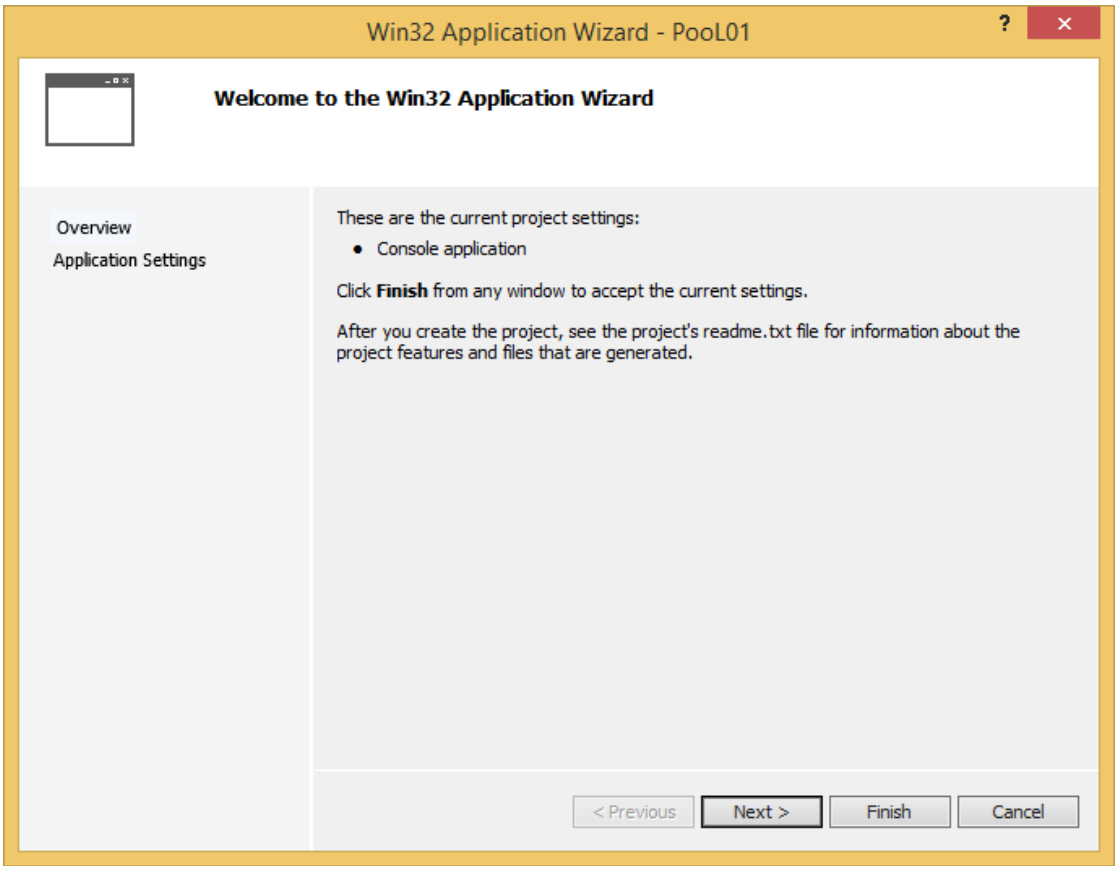

Fig. 1.3 Fereastra Welcome…

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

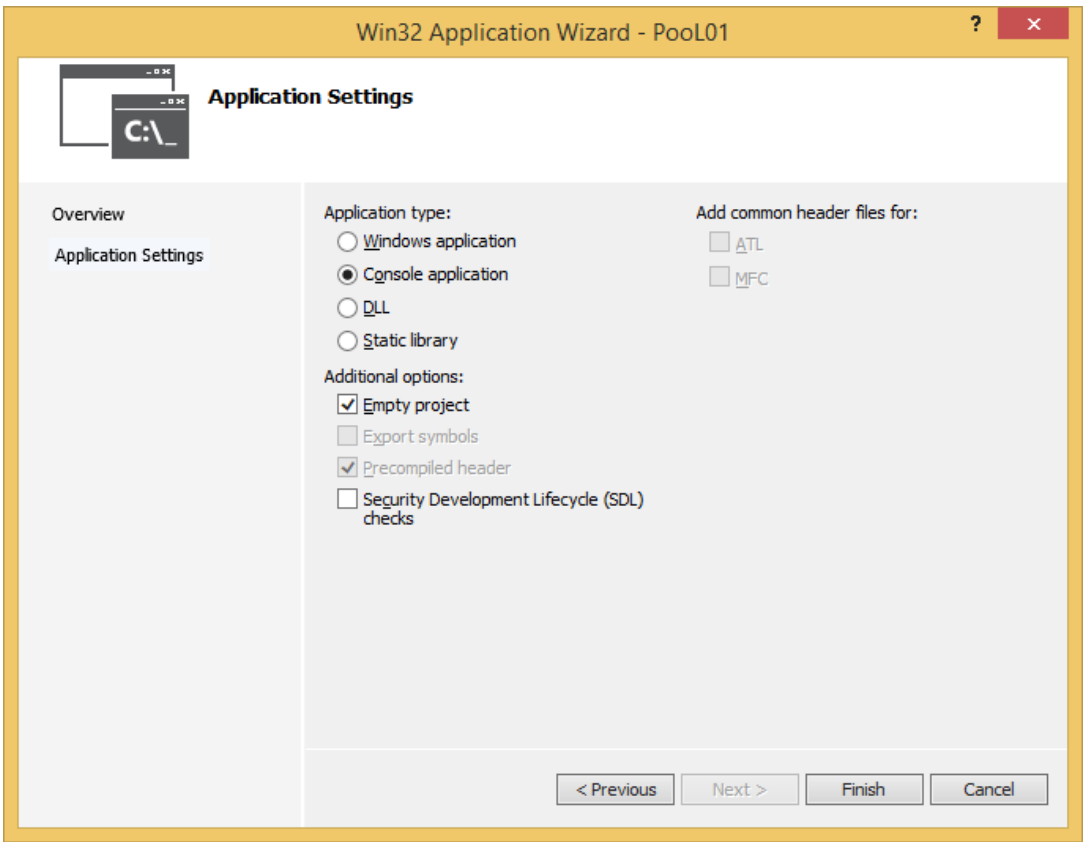

Fig.1.4 Fereastra Application Settings

9. Se apasă *Finish*.

Proiectul este creat şi 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 fişierele proiectului. Iniţial proiectul nu conţine nici un fişier.
- *Start Page* – în restul ecranului. Această fereastră nu este utilă, ea poate fi închisă.

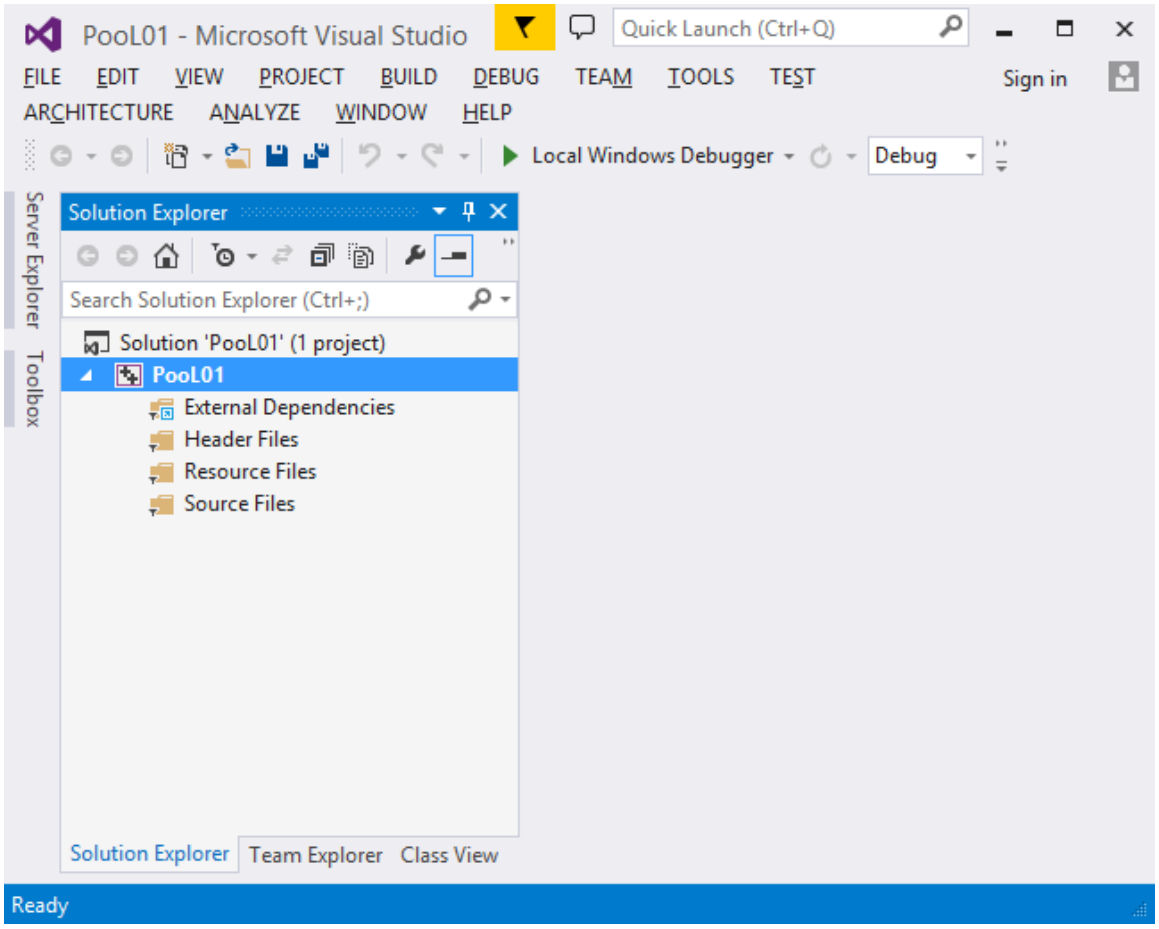

Fig. 1.5 Proiectul nou creat

5. Adăugarea unui fișier la proiect

Pentru a scrie un program în VS2013, trebuie adăugat un fişier sursă la proiect. Pentru aceasta se vor efectua următorii paşi:

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

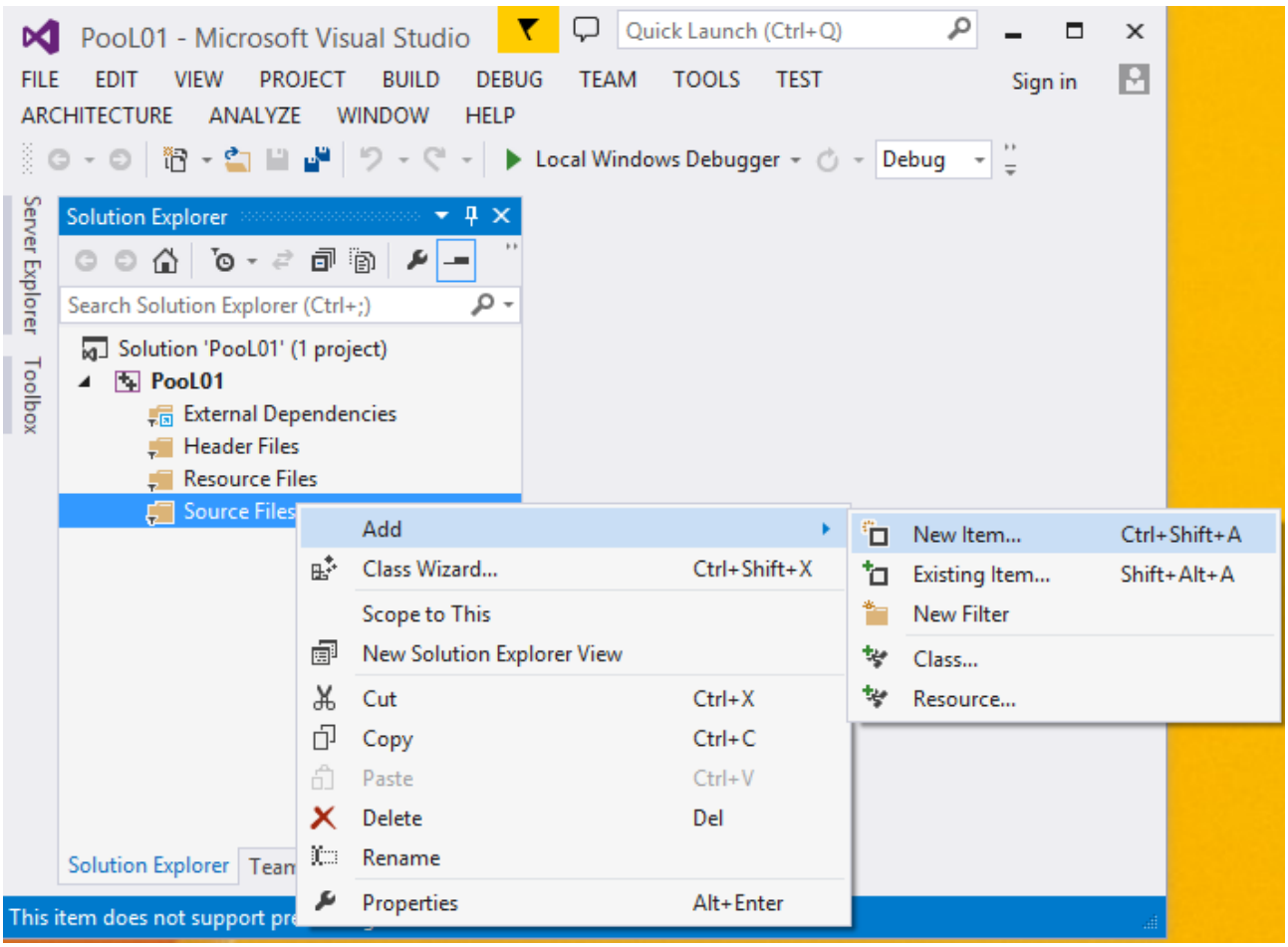

Fig. 1.6 Adăugare fişier sursă

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

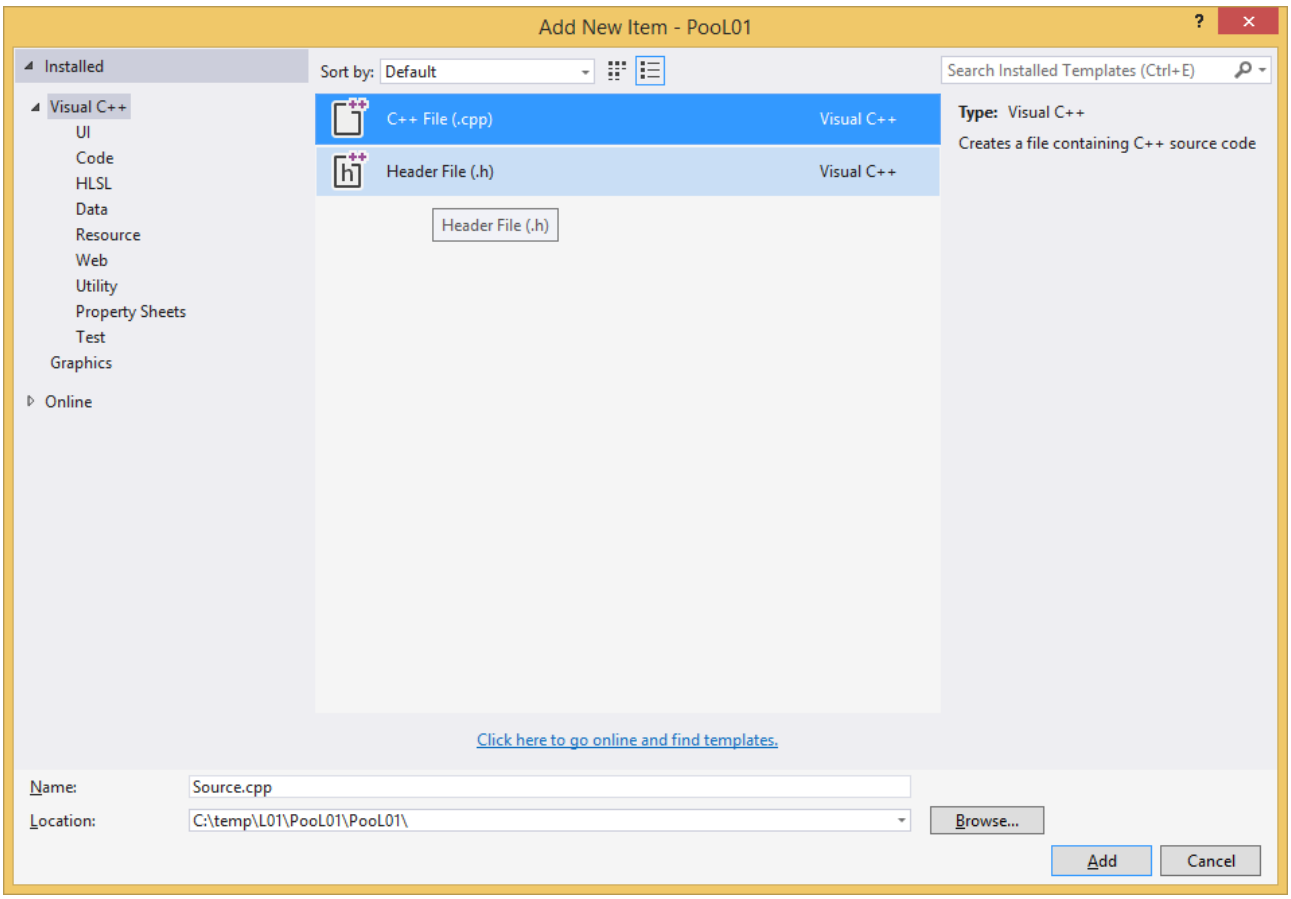

Fig. 1.7 Adăugare fişier sursă – partea 2

5. Se apasă *Add*. Noul fişier va fi creat şi deschis în editor.

6. Scrierea programului

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

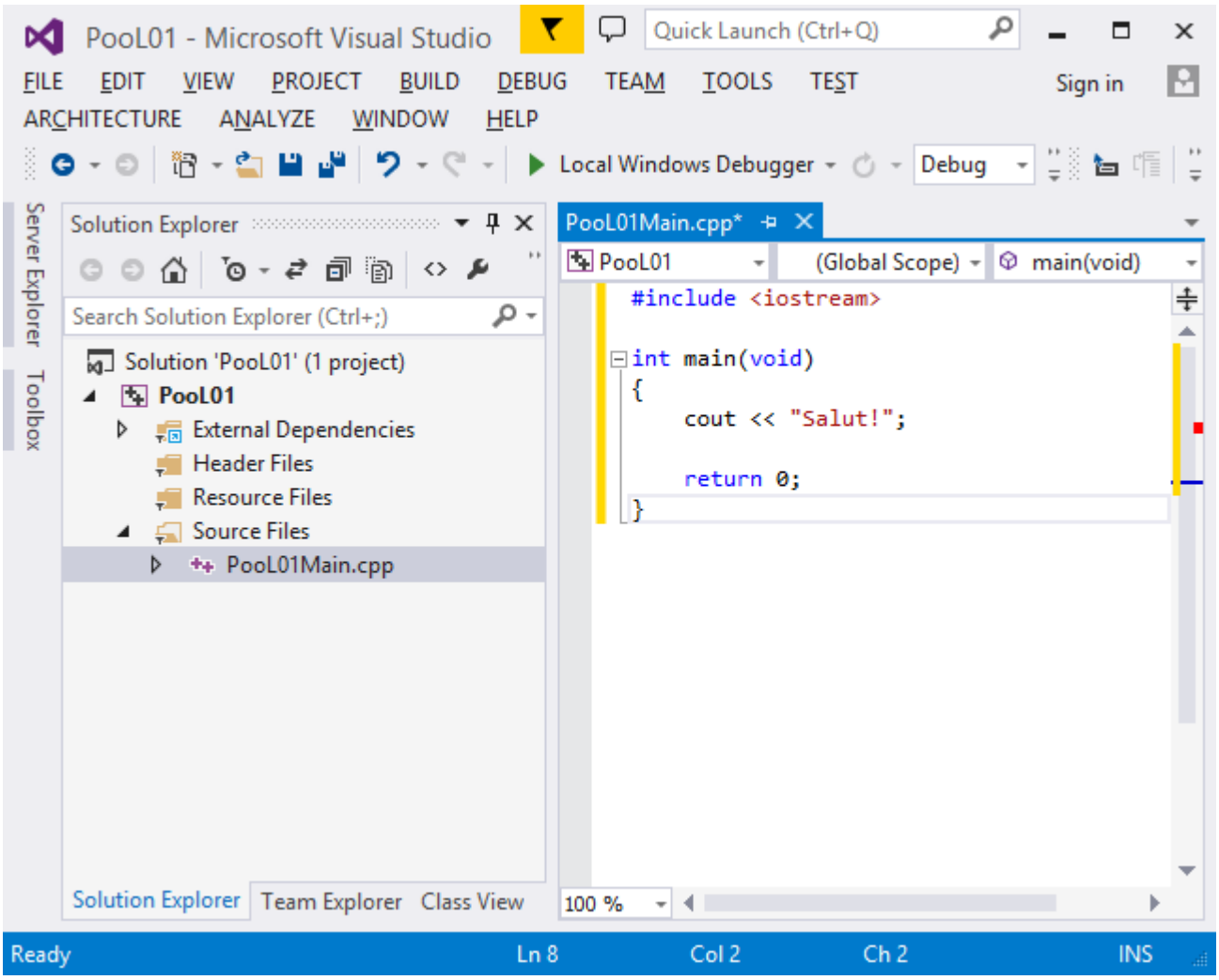

Fig. 1.8 Scrierea codului sursă în editor

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

7. Compilarea, rularea şi 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 afişate în fereastra *Error List*. Se va introduce intenţionat o eroare pentru a vedea facilităţile 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 şi se va afişa în fereastra *Error List* următorul mesaj (figura 1.10):

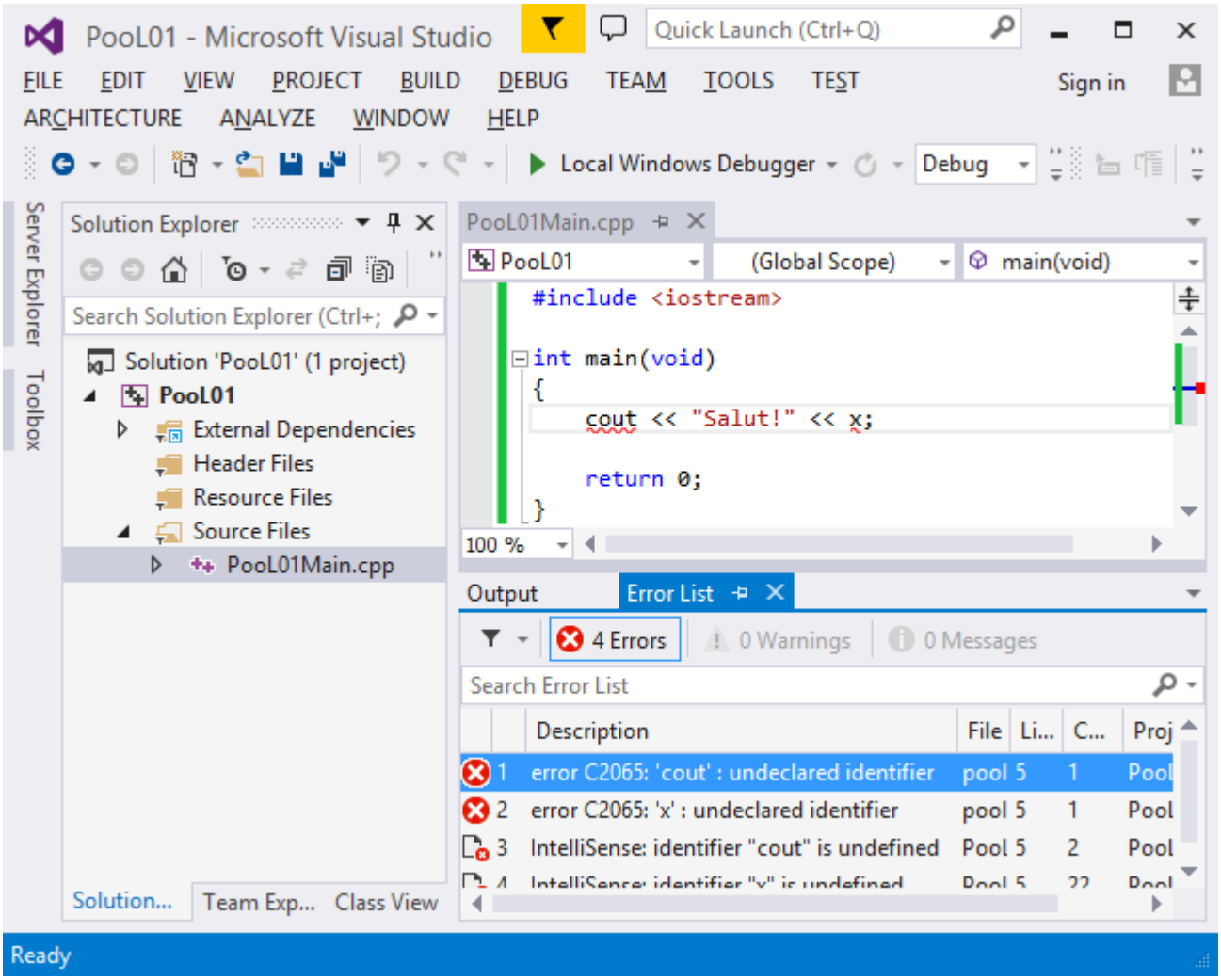

Fig. 1.10 Mesaj de eroare

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

În general se aplică următoarea regulă: eroarea trebuie căutată ori pe linia afişată în *Error List*, ori cu o linie mai sus. De exemplu, dacă în program, se şterge caracterul „;” de la sfârşitul 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 execuţie şi se recomandă eliminarea acestora înainte de rularea proiectului.

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

8. Scrierea / citirea cu cin şi cout

În C++ s-a elaborat o modalitate mai simplă de scriere/citire la/de la consolă având acelaşi efect ca şi funcţiile de bibliotecă `scanf()/printf()` din C. La începutul execuţiei fiecărui program sunt instanţiate automat două variabile globale speciale `cin` şi `cout`. Ele sunt folosite pentru citirea, respectiv scrierea la/de la consola.

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 şi depozitată în variabila `a`. Tipul variabilei din dreapta poate fi oricare din tipurile fundamentale: `int`, `char`, `float`, `double` sau `char*` care reprezintă un şir 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 semnificaţie specială pentru variabila `cout`. Expresia:

```
cout << str;
```

semnifică faptul că variabila `str` este scrisă la consolă. Variabilele scrise pot avea aceleaşi tipuri ca şi 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 menţionat în lista de tipuri suportate pentru operandul dreapta. Totuşi, utilizarea lui a fost posibilă într-o expresie cu `cout`. De ce?

Variabilele *`cin`* şi *`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`* şi *`cout`*) sunt **obiecte**. Obiectele vor fi studiate în detaliu ulterior.

Iată un exemplu complet folosind noile facilităţi 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 funcţie specială numită şi manipulator care trimite o nouă linie (echivalent cu `"\n"`) la ieşirea standard, golind şi bufferul acesteia.

Atât expresiile cu `cin` cât şi cele cu `cout` pot fi înlănţuite.

Expresia

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

este echivalentă cu

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

Comparativ cu funcţiile `printf() / scanf()` din C, expresiile cu `cin` şi `cout` sunt mai simple şi mai uşor de înţeles. Nu mai sunt necesari specificatorii de format. Dezavantajul constă în faptul că nu se pot face afişări formatate pe un anumit număr de caractere. O afişare de genul:

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

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

9. Un program mai complex

Fie următorul program: să se citească de la consolă un vector de `n` şiruri de caractere, se alocă spaţiu de memorie strict necesar şi sortează şirurile în ordine crescătoare.

În Visual Studio, se poate crea o soluţie nouă (o soluţie reprezintă o mulţime de proiecte deschise simultan), sau se poate adăuga un proiect nou în cadrul soluţiei existente. Se va crea un proiect nou şi se vor adăuga trei fişiere: *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 = (char**)calloc(n, sizeof(char*));
	cin.ignore(100, '\n');
	for(int i=0; i<n; i++) {
		int len;
		cin.getline(buffer, 100);
		len = strlen(buffer);
		vsiruri[i]=(char*)calloc(len+1,sizeof(char));
		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++) {
			free(vsiruri[i]);
		}
		free(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 conţine câteva elemente noi:

1. În fişierul *SortareSiruri.cpp*, avem următoarea linie în funcţia `citireVSiruri()`:

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

Acest apel este necesar înainte de apelarea funcţiei `cin.getline()`, care citeşte 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 documentaţia acestei funcţii. Ea are acelaşi rol ca şi funcţia `fflush(stdin)` în C, cu observaţia că `fflush(stdin)` goleşte bufferul de intrare.

2. În aceeaşi fişier *SortareSiruri.cpp*, este apelul:

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

Funcţia `cin.getline()` citeşte o linie de la consolă, sub forma unui şir de caractere şi o depune în buffer. Al doilea-lea parametru – 100 – este numărul maxim de caractere care pot fi citite. Funcţia este echivalentă cu `fgets()` din C.

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

Recomandare.

Toate programele realizate în cadrul laboratoarelor de POO vor fi concepute ca proiecte, respectând aceleaşi standarte ca şi la obiectul Programarea Calculatoarelor. Adică unul sau mai multe fişiere header, unul sau mai multe fişiere cu funcţii şi un fişier cu funcţia main.

10. Depanare

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

De exemplu, se va considera cazul unei erori des întâlnite şi se presupune că în programul *SortareSiruri.cpp* afişarea este greşită. Se va verifica în primul rând dacă alocarea memoriei şi citirea a fost efectuată corect iar apoi, dacă sortarea a fost corectă. Pentru aceasta, se vor plasa două breakpoint-uri în funcţia `main`, unul după funcţia de citire şi altul după funcţia de sortare. Pentru plasarea unui breakpoint, se va poziţiona cursorul în dreptul liniei de cod la care se doreşte oprirea execuţiei programului şi se apasă tasta ***F9***. Se poate scoate un breakpoint tot prin apăsarea tastei ***F9***. În partea dreaptă a editorului va apare o bulină roşie pentru fiecare breakpoint astfel plasat(figura 1.11):

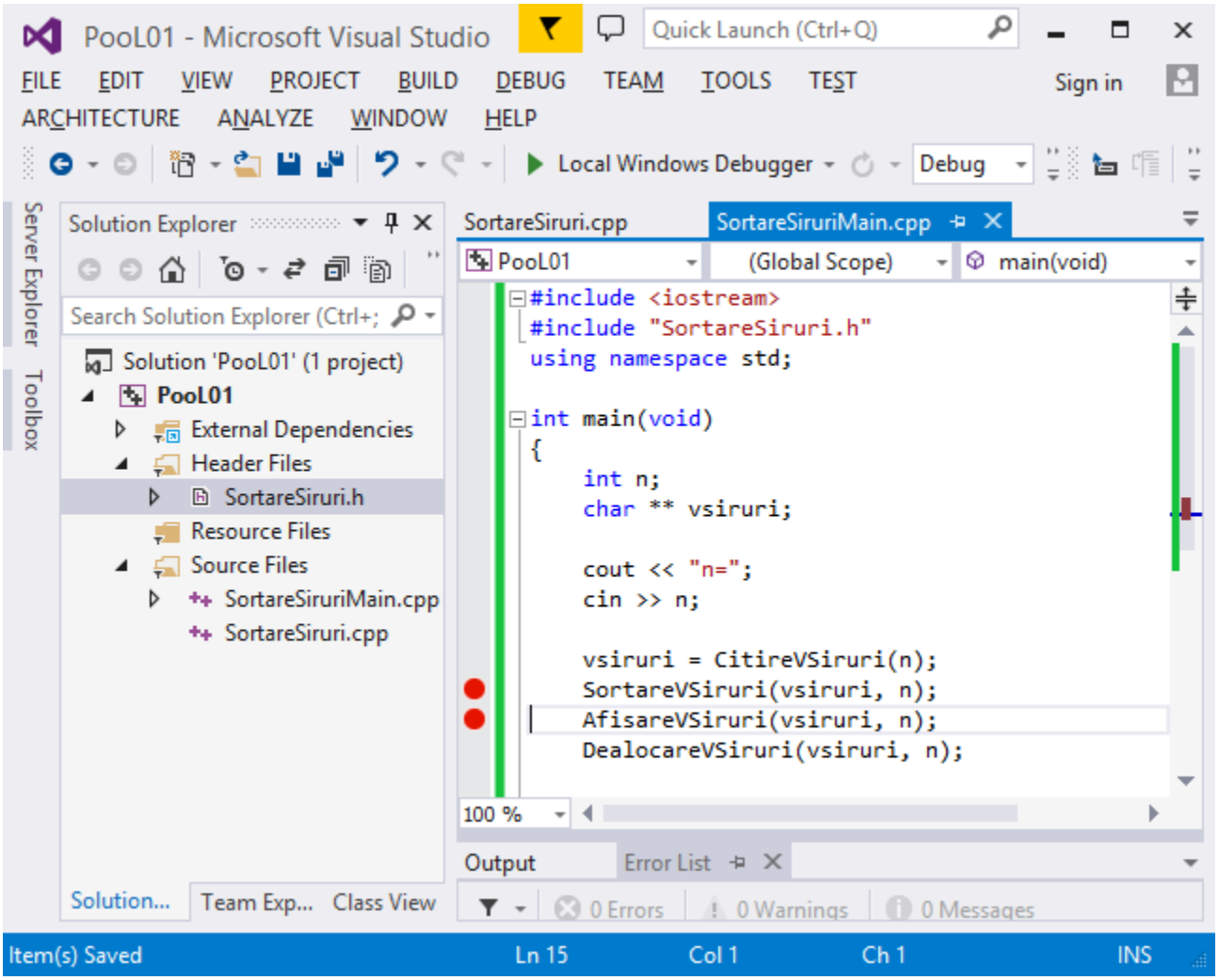

Fig. 1.11 Breakpoint-uri

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

Atenţie! *Instrucţiunea la care se află săgeata încă nu s-a executat, dar instrucţiunea 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` şi `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 1.12). În această fereastră, se dă click pe coloana *Name* şi se introduc numele variabilelor ce se doresc a fi vizualizate – întâi `n`, se apasă ENTER, pe urmă `vsiruri`, se apasă din nou ENTER:

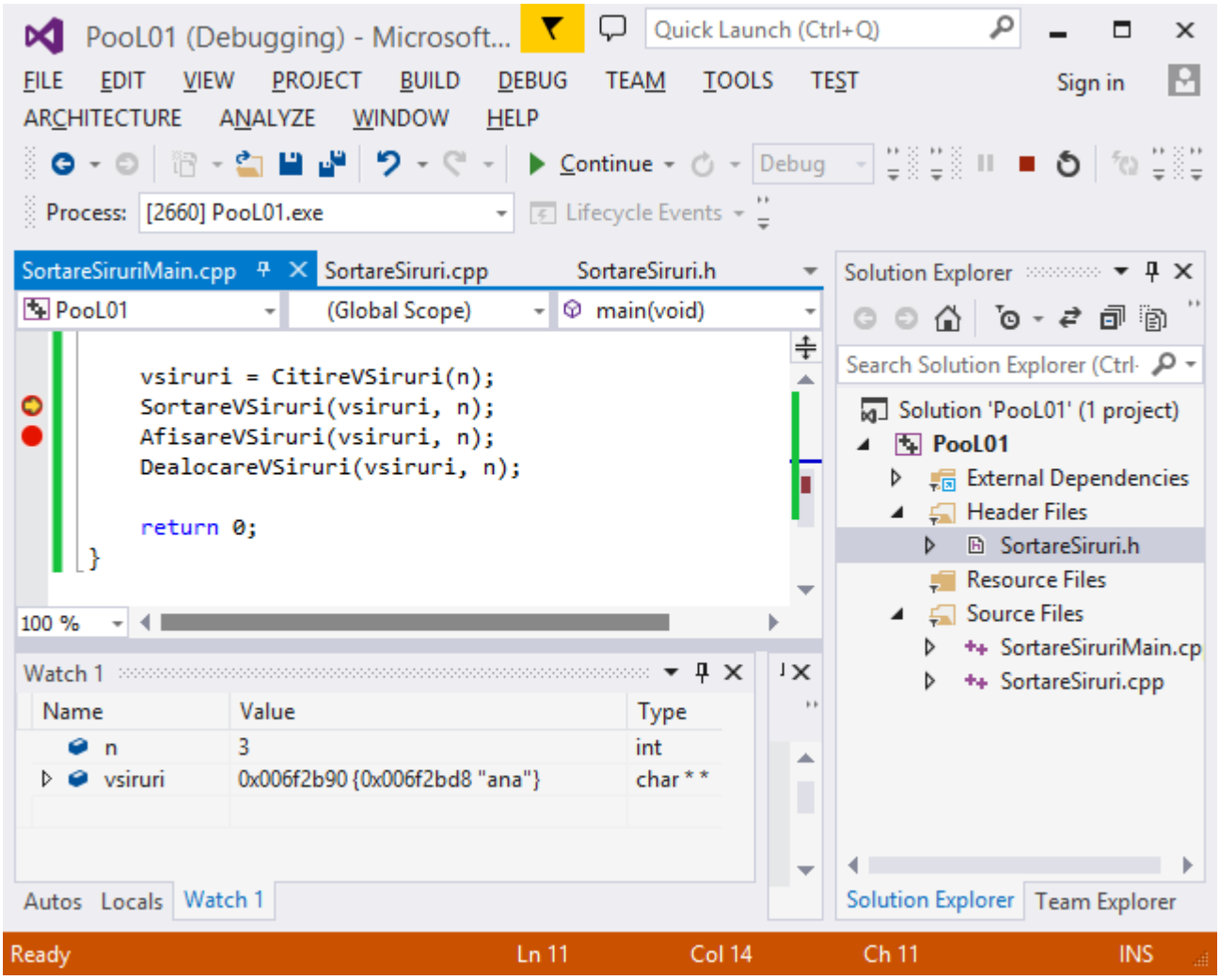

Fig. 1.12 Fereastra Watch

În a doua coloană (*Value)*, va fi afişată valoarea variabilelor la acel moment al execuţiei 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 afişează o valoare în hexazecimal ce reprezintă adresa de memorie a primului element din şir şi care nu este aşa de utilă.

Fereastra *Watch* permite vizualizarea nu doar a variabilelor, ci şi a expresiilor. De exemplu, se poat vizualiza elementele vectorului `vsiruri`(figura 1.13):

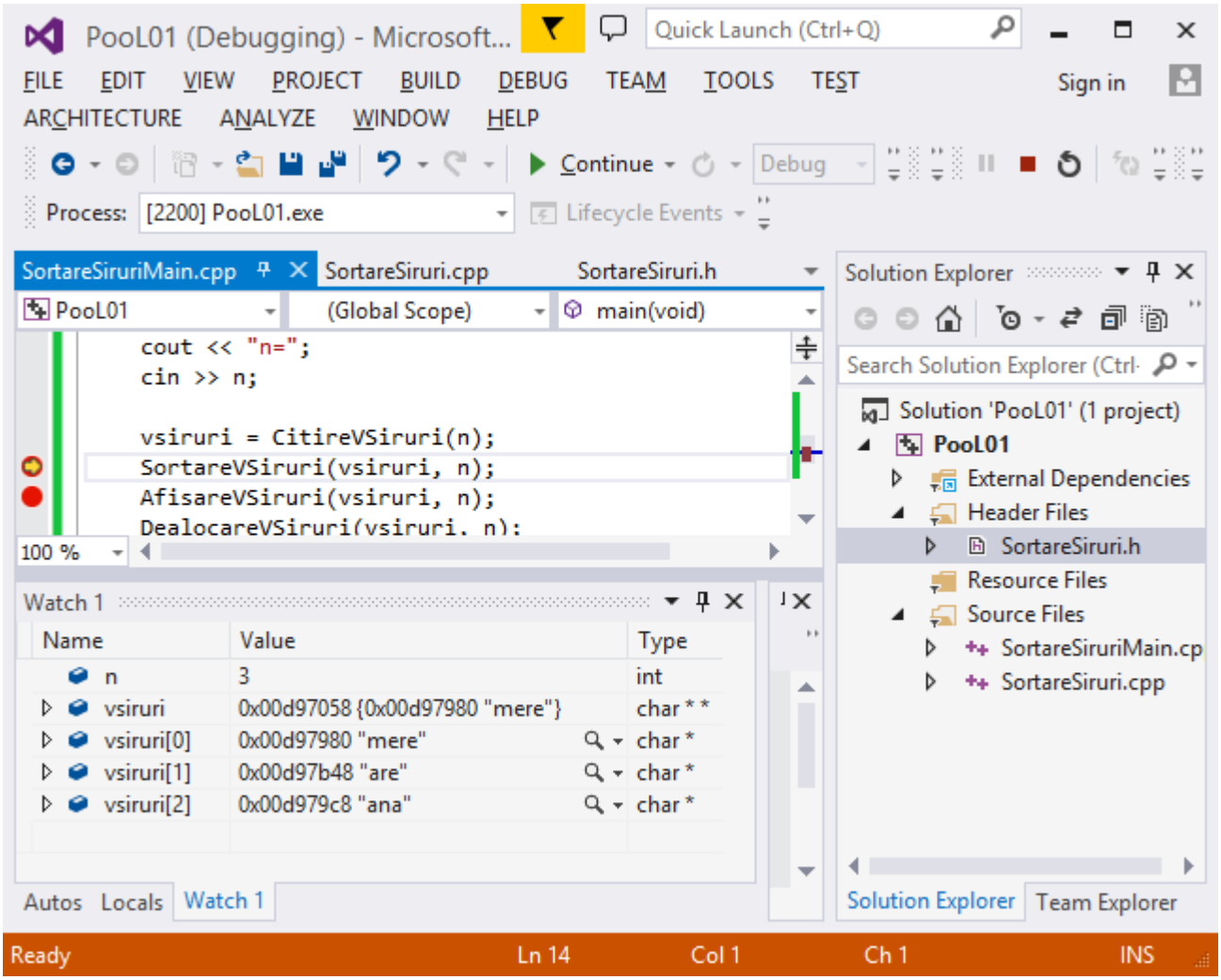

Fig. 1.13 Vizualizarea elementelor unui vector

Se observă că de data aceasta fiecare şir de caractere este afişat corect.

Citirea s-a efectuat aşa cum era de aşteptat. Până în acest punct programul se execută corect. Se apăsăm tasta ***F5*** pentru a continua execuţia programului până la următorul breakpoint. Se observă următoarele (figura 1.14):

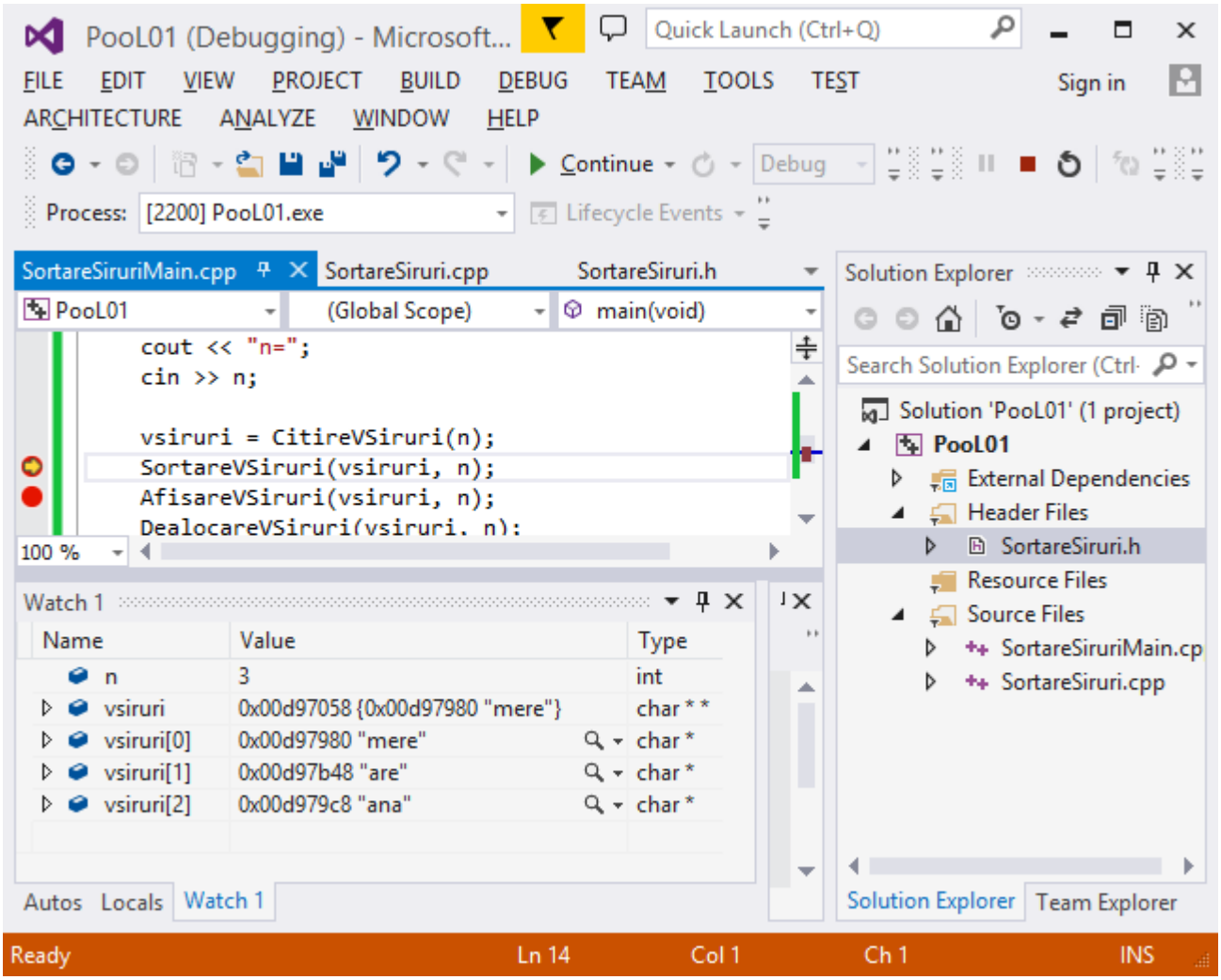

Fig. 1.14 Breakpoint după sortare

Ordinea elementelor în `vsiruri` s-a schimbat, elementele sunt sortate crescător, aşa cum era de aşteptat. Unele dintre elementele schimbate sunt afişate cu roşu. Atât citirea cât şi sortarea sunt corecte.

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

Mai jos sunt prezentate combinaţii de taste utile în Visual Studio 2013.

Combinatia de taste	**Efect**
Ctrl + C	Copy - copiere
Ctrl + V	Paste - afişare
Ctrl + A	Select all – selectare totală
Ctrl + K, F	Format selected – formatare selecţie
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 execuţiei după breakpoint
F9	Insert / remove breakpoint Adăugarea / eliminare breakpoint

11. Documentaţie

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 documentaţia completă a tuturor fişierelor de bibliotecă din C şi a funcţiilor din acestea. Documentaţia este similară cu cea din Borland C.
- http://www.cplusplus.com/reference/iostream/unde se poate găsi documentaţia claselor ce realizează operaţii de intrare/ieşire (intrare/ieşire standard – `cin`, `cout` şi fişiere).
- http://www.cplusplus.com/doc/tutorial/ unde se poate găsi un material didactic alternativ despre limbajul C++.

12. Exerciţii

1. Creaţi un nou proiect care să conţină programul prezentat în secţiunea 7.
2. Compilaţi şi rulaţi programul de la exerciţiul 1.
3. Executaţi pas cu pas programul.
4. Introduceţi în mod intenţionat o eroare în program, comentând linia:
   ```
   suntPerm = 1;
   ```
   din funcţia `sortareVSiruri()`.

Plasaţi două breakpoint-uri conform indicaţiilor din sectiunea 8. Testaţi programul cu o intrare de cinci şiruri şi verificaţi dacă la al doilea breakpoint şirurile sunt sau nu sunt sortate corect.

5. Încercăţi să detectaţi eroarea, presupunând că nu ştiţi unde este. Localizaţi prima linie de cod care se execută după ce se detectează că două şiruri trebuie inversate. Introduceţi un breakpoint pe acea linie. Vizualizaţi toate variabilele locale şi cele două şiruri care urmează să fie inversate. Ce expresii veţi introduce în fereastra *Watch* pentru cele două şiruri?

   De câte ori ar trebui să se realizeze inversarea şirurilor şi de câte ori are loc în realitate? Ce puteţi spune despre comportamentul programului?
6. Introduceţi un breakpoint pe linia `while` şi demonstraţi cu ajutorul lui că bucla `while` se execută doar o dată în programul eronat.
7. Corectaţi eroarea în program. Număraţi de câte ori se execută bucla `while` şi de câte ori are loc inversarea a două şiruri consecutive în funcţia de sortare.
8. Creati un tip de date structura numit catalog care sa contina numarul de studenti, un pointer pe siruri de caractere (vector alocat dinamic de siruri) si doi pointeri pe functii pentru sortare alfabetica si sortare dupa lungime si un pointer pe functie pentru citirea datelor. Definiti:
   a. Cele doua functii de sortare
   b. Functia de citire a sirurilor de caractere de la intrarea standard
   c. O functie de creare a unui catalog (aloca memorie pentru un tip de date catalog, initializeaza pointerii pe functii cu adresele functiilor definite mai sus, apeleaza functia de citire a datelor)
   d. O functie de distrugere a unei variabile de tip catalog (dealoca spatiul de memorie ocupat)

Scrieti un program care sa utilizeze un astfel de tip de date folosind compilatorul de C.
Modificati programul pentru a utiliza facilitatile oferite de compilatorul de C++.