F

Użycie debugera Visual Studio

Zagadnienia

W tym rozdziale poruszymy następujące zagadnienia:

- definiowanie punktów przerwania i uruchamianie programu w debugerze
- użycie polecenia `continue` w celu kontynuowania działania programu
- użycie okna *Locals* do wyświetlania i modyfikowania wartości zmiennych
- użycie okna *Watch* do obliczania wartości wyrażeń
- użycie poleceń *Step Into, Step Out* i *Step Over* do kontrolowania sposobu wykonywania kodu
- użycie okna *Autos* do wyświetlania zmiennych, które są zdefiniowane w poleceniach

F.1. Wprowadzenie

W rozdziale 2. poznałeś dwa rodzaje błędów — kompilacji i logiczne — a także dowiedziałeś się, jak można wyeliminować z kodu błędy kompilacji. **Błędy** logiczne nie przeszkadzają w zakończeniu kompilacji sukcesem, ale mogą prowadzić do nieprawidłowego działania programu po jego uruchomieniu. Większość producentów kompilatorów C oferuje oprogramowanie o nazwie **debuger**, które pozwala monitorować sposób wykonywania programu oraz wyszukiwać w nim błędy logiczne i je usuwać. Debuger stanie się jednym z Twoich najważniejszych narzędzi programistycznych. W tym dodatku przedstawimy kluczowe możliwości debugera Visual Studio. Z kolei w dodatku G zaprezentujemy funkcje i możliwości debugera GNU.

F.2. Punkty przerwania i polecenie continue

Analizę debugera rozpoczniemy od przyjrzenia się **punktom przerwania**, czyli znacznikom, które można zdefiniować w dowolnym wykonywalnym wierszu kodu. Gdy podczas działania programu będzie miał zostać wykonany wiersz zawierający punkt przerwania, działanie programu zostanie wstrzymanie, co pozwoli programiście sprawdzić wartości zmiennych, aby ustalić, czy program zawiera błędy logiczne. Na przykład można sprawdzić wartość zmiennej przechowującej wynik obliczeń i tym samym ustalić, czy zostały one przeprowadzone prawidłowo. Warto w tym miejscu dodać, że próba ustawienia punktu przerwania w niewykonywalnym wierszu kodu (np. komentarzu) spowoduje zdefiniowanie tego punktu przerwania w następnym wykonywalnym wierszu kodu w tej funkcji.

Omawiając funkcjonalności debugera, posłużymy się przedstawionym na listingu F.1 programem, którego działanie polega na znalezieniu największej spośród trzech podanych liczb. Wykonywanie programu rozpoczyna się od funkcji `main()` zdefiniowanej w wierszach od 8. do 22. Trzy liczby całkowite są w wierszu 15. pobierane za pomocą funkcji `scanf()`. Następnie te liczby są w wierszu 19. przekazywane funkcji `maximum()`, która znajduje największą z nich. Znaleziona wartość jest zwracana funkcji `main()` za pomocą polecenia `return` zdefiniowanego w wierszu 36. Następnie ta wartość zostanie wyświetlona przez polecenie `printf()` w wierszu 19.

LISTING F.1. Znalezienie największej wartości spośród trzech liczb całkowitych

```
1  /* Plik: figF_01.c
2  Znalezienie największej liczby całkowitej spośród trzech podanych */
3  #include <stdio.h>
4
5  int maximum( int x, int y, int z ); /* Prototyp funkcji */
6
7  /* Wykonywanie programu rozpoczyna się od funkcji main() */
8  int main( void )
9  {
10    int number1; /* Pierwsza liczba całkowita */
11    int number2; /* Druga liczba całkowita */
12    int number3; /* Trzecia liczba całkowita */
13
14    printf( "Podaj trzy liczby całkowite: " );
15    scanf( "%d%d%d", &number1, &number2, &number3 );
16
17    /* number1, number2 i number3 to argumenty
18    wywołania funkcji maximum() */
19    printf( "Największa liczba to: %d\n", maximum( number1, number2, number3 ) );
20 } /* Koniec funkcji main() */
21
22 /* Definicja funkcji maximum() */
23 /* x, y i z to parametry */
24 int maximum( int x, int y, int z )
25 {
26    int max = x; /* Przyjęcie założenia, że x to największa wartość */
```

```
27
28    if ( y > max ) {  /* Jeżeli wartość y jest większa niż max, należy przypisać wartość y do max */
29       max = y;
30    }  /* Koniec pętli if */
31
32    if ( z > max ) {  /* Jeżeli wartość z jest większa nie max, należy przypisać wartość z do max  */
33       max = z;
34    }  /* Koniec pętli if */
35
36    return max;  /* max to największa wartość */
37 }  /* Koniec funkcji maximum() */
```

Utworzenie projektu

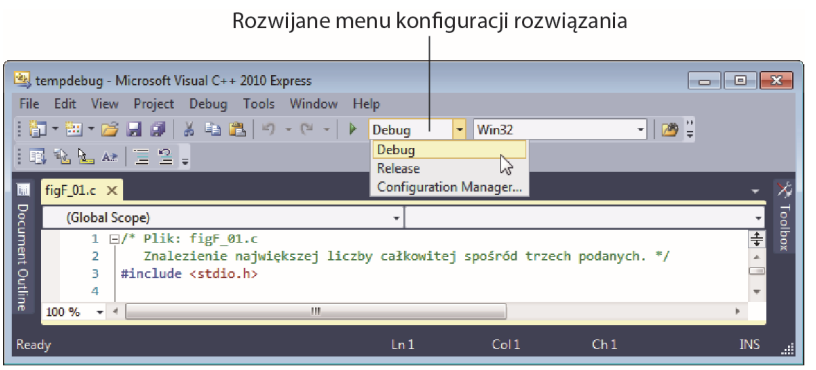
Oto kroki umożliwiające utworzenie projektu, który będzie zawierał kod zamieszczony na listingu F.1:

1. W Visual Studio wybierz opcję *File/New/Project…*, co spowoduje wyświetlanie okna dialogowego *New Project*.
2. Na liście *Installed Templates* w sekcji *Visual C++* wybierz grupę *Win32*, a następnie w części środkowej okna dialogowego wybierz *Win32 Console Application*.
3. W polu *Name:* wpisz nazwę projektu, natomiast w polu *Location:* wskaż katalog, w którym projekt ma zostać zapisany. Następnie kliknij przycisk *OK*.
4. W wyświetlonym na ekranie oknie dialogowym *Win32 Application Wizard* kliknij przycisk *Next >*.
5. W sekcji *Application type* wybierz *Console application*, natomiast w sekcji *Additional options* wybierz *Empty project* i kliknij przycisk *Finish*.
6. W oknie *Solution Explorer* prawym przyciskiem myszy kliknij katalog *Source Files* projektu, a następnie z menu kontekstowego wybierz opcję *Add/Existing Item…*. Na ekranie zostanie wyświetlone okno dialogowe *Add Existing Item*.
7. Odszukaj katalog zawierający przykłady dla dodatku F, zaznacz znajdujący się w nim plik kodu źródłowego o nazwie *figF_01.c* i kliknij przycisk *Add*.

Włączenie trybu debugowania i wstawienie punktu przerwania

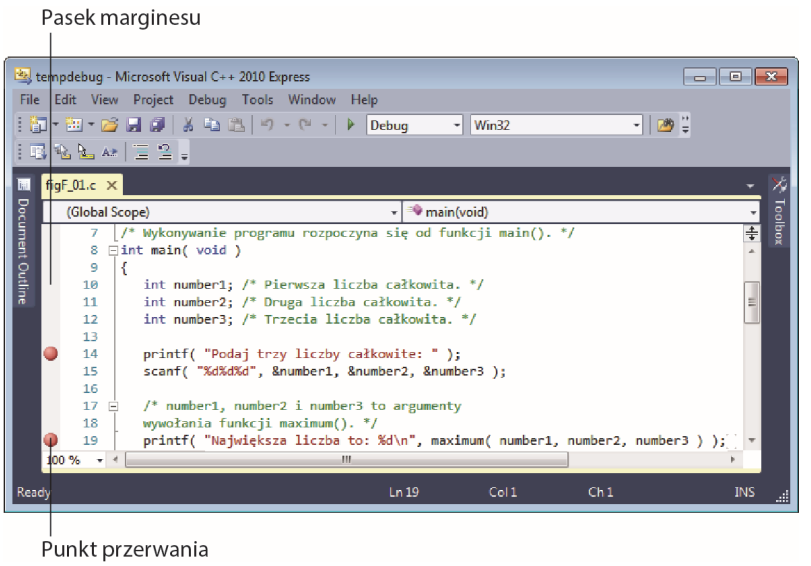
W następujących krokach wykorzystasz punkty przerwania i różne polecenia debugera do przeanalizowania wartości zmiennej `number1` zdefiniowanej w kodzie na listingu F.1:

1. **Włączenie debugera**. Debuger jest domyślnie włączony. Jeżeli tak nie jest, trzeba zmienić ustawienia *rozwijanego menu konfiguracji rozwiązania* (rysunek F.1), które znajduje się na pasku narzędzi. W tym celu kliknij strzałkę skierowaną w dół w tym menu, a następnie wybierz opcję *Debug*.



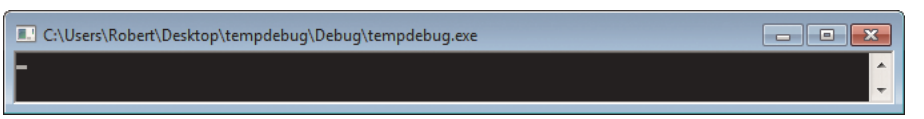
RYSUNEK F.1. Włączenie debugera

2. **Wstawienie punktów przerwania**. Otwórz plik *figF_01.c* przez jego dwukrotne kliknięcie w oknie *Solution Explorer*. Aby wstawić punkt przerwania, trzeba kliknąć myszą szary *pasek marginesu* (po lewej stronie okna kodu źródłowego, jak widać na rysunku F.2) na wysokości tego wiersza kodu, w którym chcesz zdefiniować punkt przerwania. Ewentualnie można kliknąć prawym przyciskiem myszy wiersz kodu, a następnie z menu kontekstowego wybrać opcję *Breakpoint/Insert Breakpoint*. Nie ma ograniczenia w zakresie liczby punktów przerwania definiowanych w kodzie. W omawianym przykładzie dodaj punkty przerwania w wierszach 14. i 19. Na pasku marginesu pojawi się czerwona kropka wskazująca na zdefiniowanie punktu przerwania w danym wierszu (rysunek F.2). Po uruchomieniu programu debuger wstrzyma jego działanie w każdym wierszu zawierającym punkt przerwania. Mówimy wówczas, że program znajduje się w **trybie debugowania**. Punkty przerwania można dodawać przed uruchomieniem programu, podczas działania programu w trybie debugowania oraz w trakcie działania programu w zwykłym trybie.



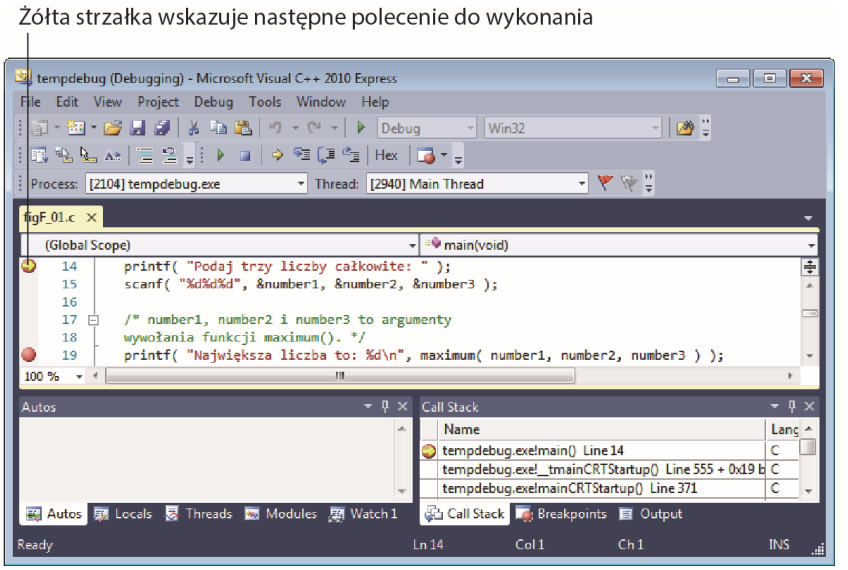
RYSUNEK F.2. Zdefiniowanie dwóch punktów przerwania

3. **Rozpoczęcie debugowania**. Po zdefiniowaniu punktów przerwania w edytorze kodu źródłowego wybierz opcję menu *Debug/Build Solution* w celu skompilowania programu, a następnie wybierz opcję menu *Debug/Start Debugging*, aby rozpocząć proces debugowania. (*Uwaga*: jeśli najpierw nie skompilujesz programu, po wybraniu opcji menu *Debug/Start Debugging* zostanie on skompilowany automatycznie). Podczas debugowania aplikacji konsoli na ekranie pojawia się okno *wiersza poleceń* (rysunek F.3), które umożliwia podanie danych wejściowych programu i wyświetlenie danych wyjściowych programu. Gdy rozpocznie się wykonywanie wiersza 14. omawianego programu, debuger przechodzi do trybu debugowania.



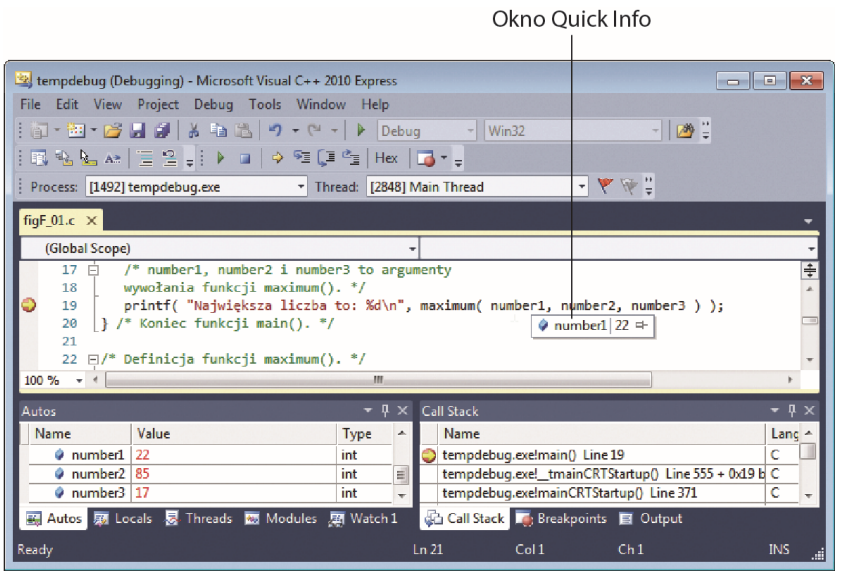
RYSUNEK F.3. Okno wiersza poleceń po uruchomieniu programu w trybie debugowania

4. **Analiza sposobu działania programu**. Po napotkaniu pierwszego punktu przerwania (wiersz 14.) podczas pracy w trybie debugowania aktywnym oknem staje się okno środowiska IDE (rysunek F.4). Żółta strzałka po lewej stronie okna kodu źródłowego wskazuje wiersz zawierający następne polecenie do wykonania.



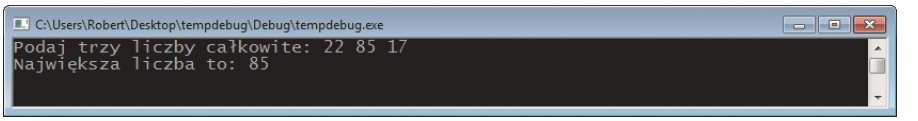
RYSUNEK F.4. Wykonywanie programu zostaje wstrzymane po napotkaniu pierwszego punktu przerwania

5. **Użycie polecenia *Continue* do wznowienia działania programu**. Aby wznowić działanie programu, należy wybrać opcję menu *Debug/Continue*. *Polecenie* ***Continue*** wznawia działanie programu, które jest kontynuowane do momentu napotkania następnego punktu przerwania lub końca funkcji `main()`, w zależności od tego, co nastąpi wcześniej. W omawianym przykładzie program kontynuuje działanie i zatrzymuje się w wierszu 15. w oczekiwaniu na podanie danych wejściowych. Wpisz wartości 22, 85 i 17 jako trzy liczby całkowite rozdzielone spacjami. Działanie programu będzie kontynuowane do momentu napotkania następnego punktu przerwania (wiersz 19.). Po umieszczeniu kursora myszy na nazwie zmiennej `number1` przechowywana w niej wartość zostanie wyświetlona w oknie *Quick Info* (rysunek F.5). Jak widzisz, to okno pomaga w wychwytywaniu błędów logicznych w programach.



RYSUNEK F.5. Okno Quick Info wyświetla wartość zmiennej

6. **Dodanie punktu przerwania w wierszu zawierającym nawias zamykający funkcji `main()`**. Punkt przerwania dodaj także w wierszu 20. kodu źródłowego, klikając margines po lewej stronie na wysokości tego wiersza. To uniemożliwi natychmiastowe zakończenie działania programu po wyświetleniu wyniku. Gdy nie ma kolejnych punktów przerwania wstrzymujących działanie programu, program będzie wykonany do końca i okno *Command Prompt* zostanie zamknięte. Jeżeli nie zdefiniujesz punktu przerwania, nie zobaczysz danych wyjściowych programu przed zamknięciem jego okna.
7. **Kontynuowanie wykonywania programu**. Wybierz opcję menu *Debug/Continue* w celu wykonania kodu do kolejnego punktu przerwania. Program wyświetli wynik przeprowadzonych obliczeń (rysunek F.6).



RYSUNEK F.6. Dane wyjściowe wygenerowane przez omawiany program

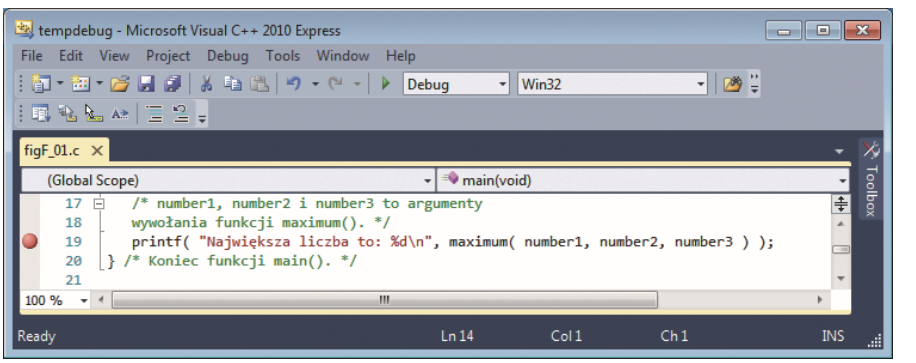
8. **Usunięcie punktu przerwania**. Kliknij punkt przerwania wyświetlony na pasku marginesu.
9. **Zakończenie działania programu**. Wybierz opcję menu *Debug/Continue*, aby zakończyć działanie programu.

W tej części dodatku dowiedziałeś się, jak włączyć debuger i zdefiniować punkty przerwania, co umożliwia analizowanie wyników działania kodu po uruchomieniu programu. Ponadto zobaczyłeś, jak można kontynuować działanie programu po jego wstrzymaniu w miejscu zdefiniowania punktu przerwania, a także jak usunąć niepotrzebne punkty przerwania.

F.3. Okna Locals i Watch

Z poprzedniej części tego dodatku wiesz, że okno *Quick Info* pozwala sprawdzić wartość zmiennej. W tej części zobaczysz, jak wykorzystać okno *Locals* do przypisania nowych wartości zmiennych podczas działania programu. Okna *Watch* natomiast użyjesz do przeanalizowania wartości znacznie bardziej skomplikowanych wyrażeń.

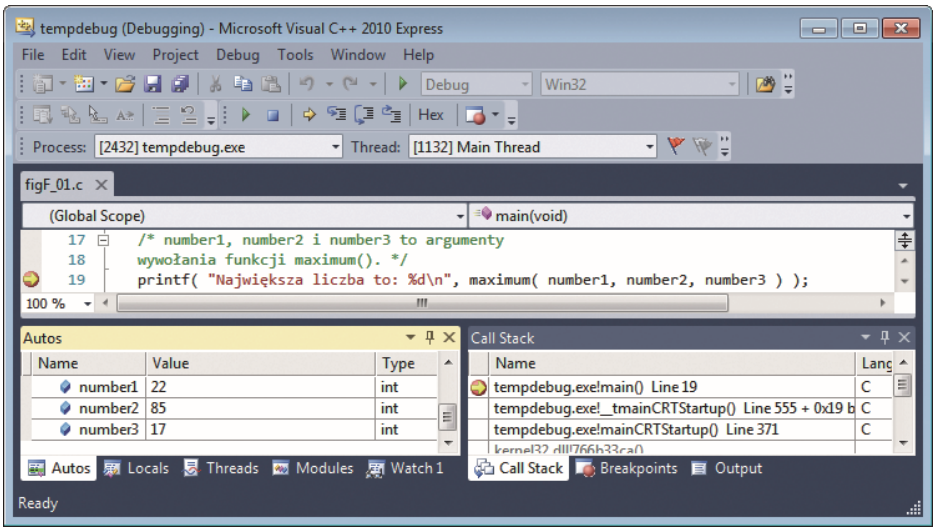
1. **Dodanie punktu przerwania**. Usuń istniejące punkty przerwania przez ich kliknięcie na pasku marginesu. Następnie dodaj nowy punkt przerwania, klikając pasek marginesu na wysokości wiersza 19. (rysunek F.7).



RYSUNEK F.7. Dodanie punktu przerwania w wierszu 19.

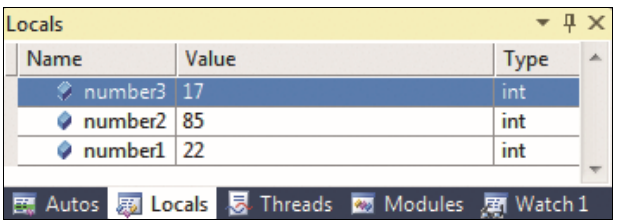
2. **Rozpoczęcie debugowania**. Wybierz opcję menu *Debug/Start Debugging*. Wpisz wartości 22, 85 i 17 po wyświetleniu na ekranie komunikatu *Podaj trzy liczby całkowite*. Naciśnij klawisz *Enter*, aby program odczytał podane wartości.

3. **Zawieszenie wykonywania programu**. Podczas wykonywania polecenia w wierszu 19. debuger wchodzi do trybu debugowania (rysunek F.8). W tym momencie polecenie zdefiniowane w wierszu 15. odczytało wartości podane dla zmiennych `number1` (22), `number2` (85) i `number3` (17). Polecenie w wierszu 19. zostanie wykonane jako następne.



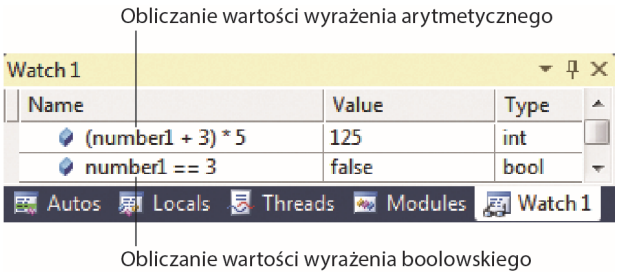
RYSUNEK F.8. Wykonywanie programu zostało zawieszone, gdy debuger dotarł do polecenia w wierszu 19.

4. **Analiza danych**. W trybie debugowania można analizować wartości zmiennych lokalnych, wykorzystując do tego okno debugera o nazwie *Locals*, które standardowo jest wyświetlane w lewym dolnym rogu okna środowiska IDE. Jeżeli na ekranie nie widzisz tego okna, możesz je wyświetlić za pomocą opcji menu *Debug/Windows/Locals*. Na rysunku F.9 można zobaczyć wyświetlone w oknie *Locals* wartości zmiennych lokalnych funkcji `main()`: `number1` (22), `number2` (85) i `number3` (17).



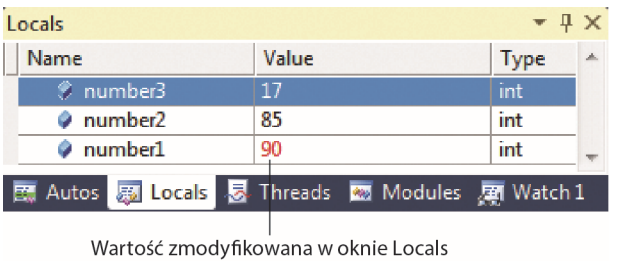
RYSUNEK F.9. Przeglądanie zmiennych number1, number2 i number3

5. **Obliczanie wartości wyrażeń arytmetycznych i boolowskich**. Do obliczenia wartości wyrażenia arytmetycznego lub boolowskiego można wykorzystać okno *Watch*. Istnieje możliwość wyświetlenia do czterech takich okien. Wybierz opcję menu *Debug/Windows/Watch 1*. W pierwszym wierszu kolumny *Name* wpisz `(number1 + 3) * 5` i naciśnij klawisz *Enter*. Wartość tego wyrażenia (w omawianym jest przykładzie to 125) zostanie wyświetlona w kolumnie *Value* (rysunek F.10). W następnym wierszu kolumny *Name* wpisz `number1 == 3` i naciśnij klawisz *Enter*. Zadanie tego wyrażenia polega na ustaleniu, czy wartością `number1` jest 3. Wyrażenie zawierające operator `==` (bądź każdy inny operator relacji lub równości) jest traktowane jako wyrażenie boolowskie. W omawianym przykładzie wartością wyrażenia jest `false` (rysunek F.10), ponieważ zmienna `number1` aktualnie zawiera wartość 22, a nie 3.



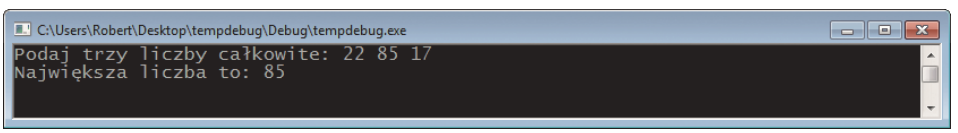
RYSUNEK F.10. Analiza wartości wyrażeń

6. **Modyfikowanie wartości**. Największą liczbą spośród danych wejściowych pochodzących od użytkownika (22, 85 i 17) powinna być 85. Jednak okno *Locals* można wykorzystać do zmiany wartości zmiennych w trakcie wykonywania programu. Jest to cenna możliwość podczas eksperymentowania z różnymi wartościami, która ponadto ułatwia wychwytywanie błędów logicznych. W oknie *Locals* kliknij w kolumnie *Value* pole w wierszu `number1` i tym samym zaznacz wartość 22. Wpisz 90 i naciśnij klawisz *Enter*. Debuger zmieni wartość zmiennej `number1` i w kolorze czerwonym wyświetli nową wartość (rysunek F.11).



RYSUNEK F.11. Modyfikowanie wartości zmiennej

7. **Dodanie punktu przerwania w wierszu zawierającym nawias zamykający funkcji `main()`**. Punkt przerwania dodaj także w wierszu 20. kodu źródłowego, klikając margines po lewej stronie na wysokości tego wiersza. To uniemożliwi natychmiastowe zakończenie działania programu po wyświetleniu wyniku.
8. **Kontynuowanie wykonywania programu**. Wybierz opcję menu *Debug/Continue* w celu wykonania kodu do kolejnego punktu przerwania. Funkcja `main()` będzie wykonywana dalej aż do końca, a program wyświetli wynik przeprowadzonych obliczeń, którym jest 90 (rysunek F.12). To potwierdza, że zmiana w punkcie 6. faktycznie spowodowała zmianę pierwotnej wartości zmiennej `number1` z 85 na 90.



RYSUNEK F.12. Wyświetlone dane wyjściowe potwierdzają modyfikację zmiennej number1

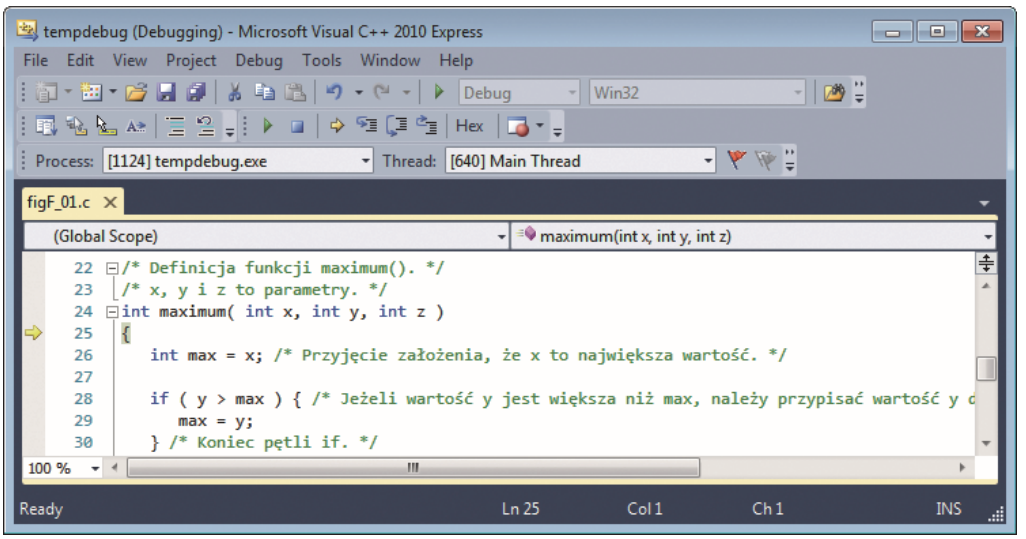
9. **Zatrzymanie sesji debugowania**. Wybierz opcję menu *Debug/Stop Debugging*. To spowoduje zamknięcie okna *Command Prompt*. Usuń wszystkie pozostałe punkty przerwania w projekcie.

W tej części dodatku dowiedziałeś się, jak korzystać z okien *Watch* i *Locals* do obliczania wartości wyrażeń arytmetycznych i boolowskich. Zobaczyłeś również, jak zmodyfikować wartość zmiennej podczas działania programu.

F.4. Użycie poleceń Step Into, Step Over, Step Out i Continue do kontrolowania wykonywania programu

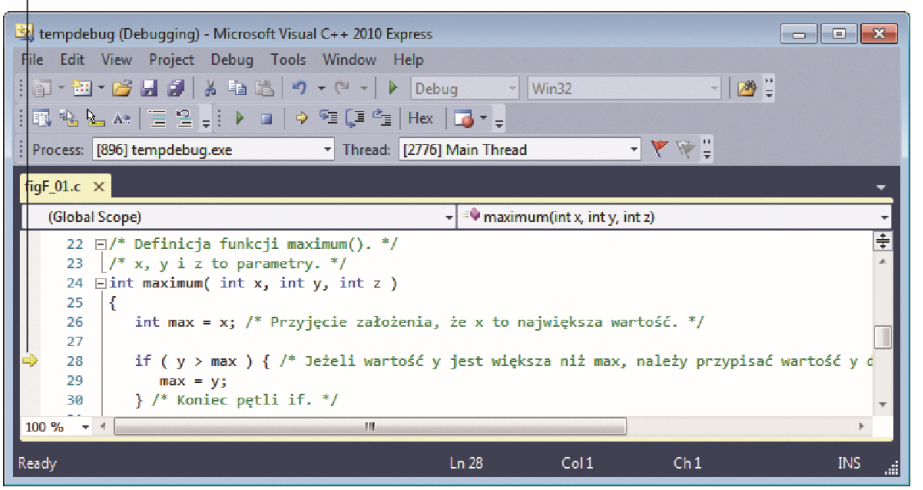
Czasami wykonywanie programu wiersz po wierszu może pomóc w ustaleniu, czy kod funkcji działa prawidłowo, a także ułatwić odszukanie i usunięcie błędów logicznych. Polecenia zaprezentowane w tej części dodatku umożliwiają wykonywanie kodu funkcji wiersz po wierszu, jednoczesne wykonanie wszystkich poleceń funkcji lub wykonanie jedynie pozostałych poleceń funkcji (jeżeli zostały już wykonane pewne polecenia zdefiniowane w funkcji).

1. **Dodanie punktu przerwania**. Dodaj nowy punkt przerwania, klikając pasek marginesu na wysokości wiersza 19.
2. **Rozpoczęcie debugowania**. Wybierz opcję menu *Debug/Start Debugging*. Wpisz wartości 22, 85 i 17 po wyświetleniu na ekranie komunikatu *Podaj trzy liczby całkowite*. Wykonywanie programu zostanie wstrzymane po dotarciu do punktu przerwania zdefiniowanego w wierszu 19.
3. **Użycie polecenia `Step Into`**. *Polecenie **Step Into*** wykonuje następne polecenie w programie (wiersz 19.), po czym natychmiast wstrzymuje działanie programu. Jeżeli to polecenie zawiera wywołanie funkcji (jak to ma miejsce w omawianym przykładzie), wówczas kontrola nad przebiegiem działania programu jest przekazywana do wywołanej funkcji. To pozwala wywoływać pojedynczo poszczególne polecenia zdefiniowane w funkcji i tym samym dokładnie sprawdzić sposób jej działania. Wybierz opcję menu *Debug/Step Into* w celu rozpoczęcia wykonywania funkcji `maximum()`. Następnie znów wybierz opcję menu *Debug/Step Into* — zwróć uwagę na żółtą strzałkę, która została umieszczona w wierszu 25. (rysunek F.13).



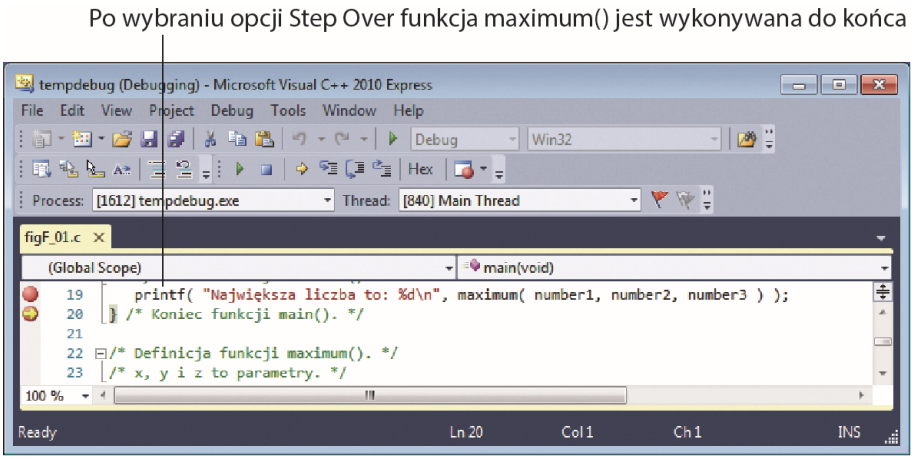
RYSUNEK F.13. Wejście do funkcji maximum()

4. **Użycie polecenia `Step Over`**. Wybierz opcję menu *Debug/Step Over*, aby wykonać bieżące polecenie (wiersz 26. na rysunku F.13) i przekazać kontrolę nad działaniem programu do wiersza 28. (rysunek F.14). Działanie polecenia *Step Over* jest podobne do działania polecenia *Step Into*, o ile następne polecenie do wykonania nie zawiera wywołania funkcji. W punkcie 10. poznasz różnice między poleceniami *Step Over* i *Step Into*.



RYSUNEK F.14. Przejście do polecenia w funkcji maximum()

5. **Użycie polecenia `Step Out`**. Wybierz opcję menu *Debug/Step Out*, aby wykonać pozostałe polecenia w funkcji i przekazać kontrolę nad działaniem programu do następnego polecenia wykonywanego (wiersz 21. na listingu F.1). Dość często w obszernych funkcjach będziesz chciał sprawdzić kilka kluczowych wierszy kodu, a następnie przejdziesz do debugowania kodu wywołującego daną funkcję. Polecenie *Step Out* pozwala kontynuować działanie programu w komponencie wywołującym funkcję, bez konieczności jej analizowania wiersz po wierszu.
6. **Dodanie punktu przerwania**. Dodaj nowy punkt przerwania na końcu funkcji `main()`, klikając pasek marginesu na wysokości wiersza 22. (listing F.1). Ten punkt przerwania będzie potrzebny w następnym kroku.
7. **Użycie polecenia `Continue`**. Wybierz opcję menu *Debug/Continue*. To spowoduje wznawianie działania programu do momentu napotkania następnego punktu przerwania zdefiniowanego w wierszu 22. Wykorzystanie tego polecenia jest użyteczne, gdy chcesz wykonać cały kod znajdujący się do kolejnego punktu przerwania.
8. **Zatrzymanie debugera**. Wybierz opcję menu *Debug/Stop Debugging*, aby zakończyć sesję pracy z debugerem. To spowoduje zamknięcie okna *Command Prompt*.
9. **Rozpoczęcie debugowania**. Zanim poznasz następną funkcję debugera, musisz ponownie go uruchomić. Wykonaj więc polecenie z kroku 2. i wpisz 22, 85 i 17 jako dane wejściowe. Po dotarciu do wiersza 19. debuger przejdzie do trybu debugowania.
10. **Użycie polecenia `Step Over`**. Wybierz opcję menu *Debug/Step Over* (rysunek F.15). Jak zapewne pamiętasz, to polecenie działa podobnie jak *Step Into*, o ile następne polecenie nie zawiera wywołania funkcji. Natomiast jeśli następne polecenie zawiera wywołanie funkcji, funkcja zostanie wykonana w całości (bez wstrzymywania któregokolwiek ze zdefiniowanych w niej poleceń), a żółta strzałka zostanie przeniesiona do wiersza zawierającego następne wykonywane polecenie (po wywołaniu funkcji) w bieżącej funkcji. W omawianym przykładzie debuger wykonuje wiersz 19. w funkcji `main()` (listing F.1). To polecenie wywołuje funkcję `maximum()`. Następnie debuger wstrzymuje działanie w wierszu 20., czyli tym, który zawiera następne polecenie możliwe do wykonania w bieżącej funkcji, `main()`.
11. **Zatrzymanie debugera**. Wybierz opcję menu *Debug/Stop Debugging*, aby zakończyć sesję pracy z debugerem. To spowoduje zamknięcie okna *Command Prompt*. Usuń wszystkie pozostałe punkty przerwania w projekcie.



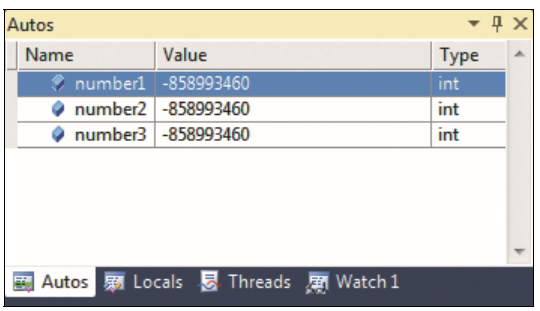
RYSUNEK F.15. Użycie polecenia Step Over w debugerze Visual Studio

W tej części dodatku dowiedziałeś się, jak używać polecenia *Step Into* debugera, aby móc przeprowadzać debugowanie funkcji w trakcie działania programu. Pokazaliśmy również możliwość użycia polecenia *Step Over* do pominięcia wywołania funkcji. Z kolei polecenie *Step Out* zastosowaliśmy do kontynuowania działania aż do końca bieżącej funkcji. Dowiedziałeś się też, że polecenie *Continue* kontynuuje działanie do następnego punktu przerwania w programie lub do zakończenia działania programu.

F.5. Okno Autos

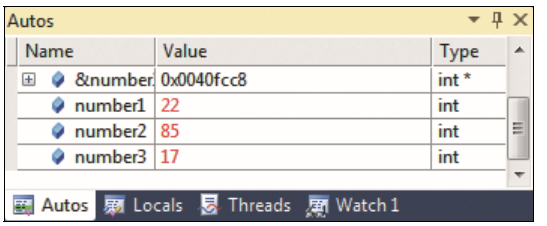
Okno *Autos* wyświetla zmienne użyte w poprzednio wykonanym poleceniu (to obejmuje również wartość zwrotną funkcji, o ile taka istnieje) oraz zmienne użyte w następnym poleceniu.

1. **Dodanie punktów przerwania**. Dodaj nowe punkty przerwania w funkcji `main()` przez kliknięcie paska marginesu na wysokości wierszy 14. i 19.
2. **Użycie okna `Autos`**. Uruchom debuger za pomocą opcji menu *Debug/Start Debugging*. Gdy w wierszu 14. debuger wejdzie do trybu przerwania, otwórz okno *Autos* (rysunek F.16) poprzez wybranie opcji menu *Debug/Windows/Autos*. Skoro wykonywanie programu dopiero się rozpoczęło, okno *Autos* zawiera jedynie zmienne w kolejnym poleceniu do wykonania — w omawianym przykładzie są to zmienne `number1`, `number2` i `number3`. Wyświetlenie wartości przechowywanych w zmiennych pozwala sprawdzić, czy program poprawnie wykorzystuje te zmienne. Zwróć uwagę, że wymienione wcześniej zmienne zawierają ogromne wartości ujemne. Te wartości mogą być odmienne w trakcie każdego uruchomienia programu, są niezainicjalizowanymi wartościami zmiennych. Ta nieprzewidywalna (i często niepożądana) wartość pokazuje, dlaczego tak duże znaczenie ma inicjalizacja wszystkich zmiennych C przed ich użyciem.



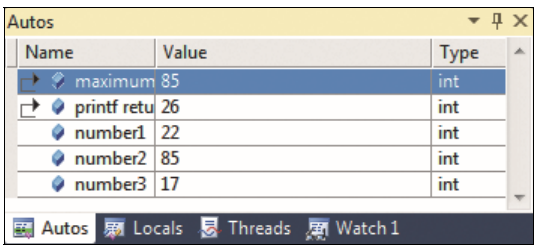
RYSUNEK F.16. Okno Autos wyświetlające wartości zmiennych number1, number2 i number3

3. **Kontynuowanie działania**. Wybierz opcję menu *Debug/Continue* w celu wykonania programu do momentu napotkania następnego punktu przerwania zdefiniowanego w wierszu 19. Gdy pojawi się komunikat z prośbą o podanie danych wejściowych, wpisz trzy liczby całkowite. Okno *Autos* wyświetli uaktualnione wartości zmiennych `number1`, `number2` i `number3` po ich inicjalizacji. Te wartości są wyświetlone w kolorze czerwonym, który wskazuje, że właśnie zostały zmienione (rysunek F.17).



RYSUNEK F.17. Okno Autos wyświetlające uaktualnione wartości zmiennych lokalnych number1, number2 i number3

4. **Wprowadzenie danych**. Wybierz opcję menu *Debug/Step Over* w celu wykonania polecenia w wierszu 19. Okno *Autos* wyświetli wartość zwrotną funkcji `maximum()` (rysunek F.18).



RYSUNEK F.18. Okno Autos wyświetla wartość zwrotną funkcji maximum()

5. **Zatrzymanie debugera**. Wybierz opcję menu *Debug/Stop Debugging*, aby zakończyć sesję pracy z debugerem. Usuń wszystkie pozostałe punkty przerwania w projekcie.

F.6. Podsumowanie

W tym dodatku pokazaliśmy, jak dodać, wyłączyć i usunąć punkty przerwania w debugerze Visual Studio. Punkt przerwania pozwala wstrzymać działanie programu, a tym samym przeanalizować wartości zmiennych. Ta możliwość okazuje się pomocna podczas wyszukiwania i usuwania błędów logicznych w programach. Poznałeś sposoby pracy z oknami *Locals* i *Watch* w celu analizy wartości wyrażenia oraz zmiany wartości zmiennych. Omówiliśmy polecenia debugera — *Step Into*, *Step Over*, *Step Out* i *Continue* — przydatne podczas ustalania, czy funkcja jest wykonywana prawidłowo. Poza tym dowiedziałeś się, jak używać okna *Autos* do analizy zmiennych stosowanych w poprzednich i następnych poleceniach.