

# Techniki programowania

INP001002WI

rok akademicki 2018/19

semestr letni

## Wykład 8

Karol Tarnowski

[karol.tarnowski@pwr.edu.pl](mailto:karol.tarnowski@pwr.edu.pl)

A-1 p. 411B



# Plan prezentacji

- Biblioteka GSL
- Wykorzystanie debuggera

Na podstawie:

- <https://www.gnu.org/software/gsl/doc/html/index.html>



# Biblioteka GSL

- Biblioteka GSL - GNU Scientific Library to zbiór implementacji numerycznych metod obliczeniowych. Biblioteka została napisana w języku C, ale możliwe jest jej wywoływanie z języków wysokiego poziomu. Kod jest rozpowszechniany w oparciu o licencję GNU General Public License.



# Biblioteka GSL

- Biblioteka GSL - GNU Scientific Library udostępnia wiele struktur i funkcji pozwalających na wykonywanie obliczeń związanych między innymi z:
  - liczbami zespolonymi,
  - wielomianami,
  - sortowaniem,
  - algebrą liniową,
  - liczbami losowymi,
  - różniczkowaniem i całkowaniem numerycznym.



# Biblioteka GSL

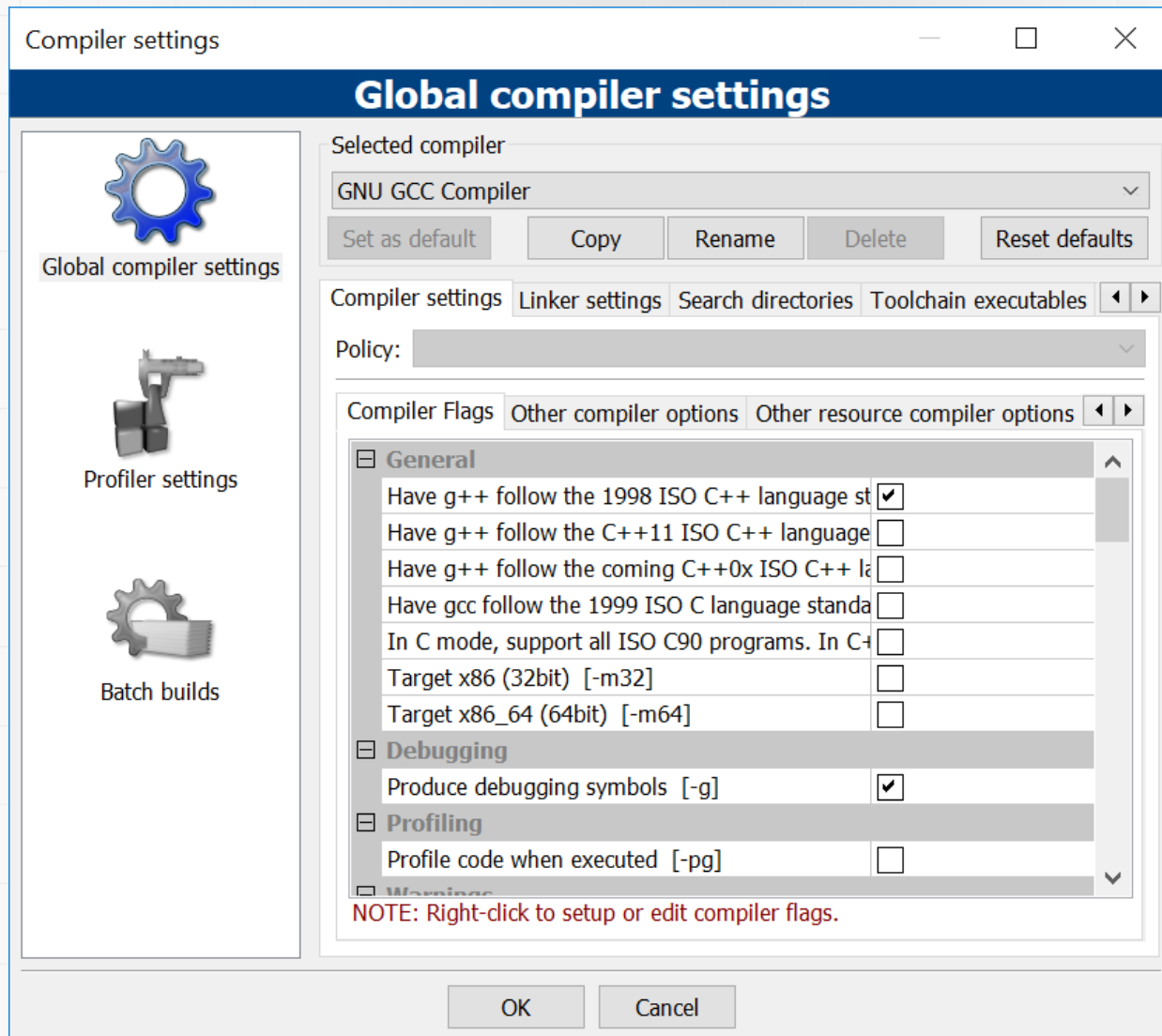
- Kod źródłowy biblioteki GSL jest ogólnodostępny (<http://www.gnu.org/software/gsl/> ).
- Pliki skompilowane (dla systemu Windows) są dostępne na stronie:  
<http://www.neff.co.at/2017/05/01/GSL-and-FGSL-for-CodeBlocks-16.01.html>



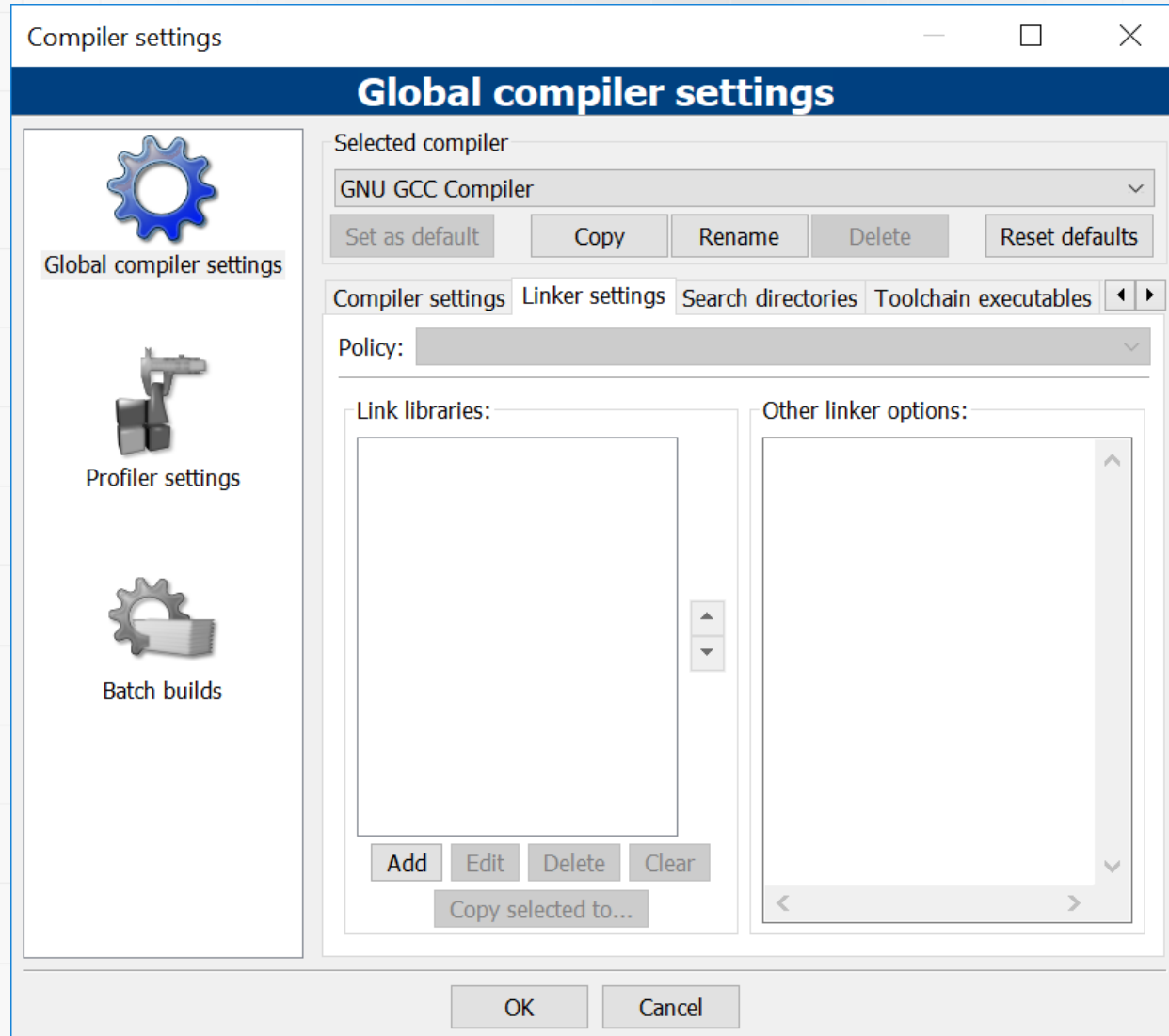
# Biblioteka GSL

- Konfiguracja Code::Blocks i GSL
  - pobrany plik `gsl-2.3.zip` należy rozpakować na dysku,
  - w środowisku Code::Blocks należy edytować ustawienia kompilatora (menu **Settings** -> **Compiler...**),
  - w zakładce **Linker settings** należy dodać skompilowane pliki biblioteki (pliki `*.a` w katalogu **lib**),
  - w zakładce **Search directories** należy dodać ścieżkę dostępu do katalogu **include** (zawierającego pliki nagłówkowe).

# Biblioteka GSL

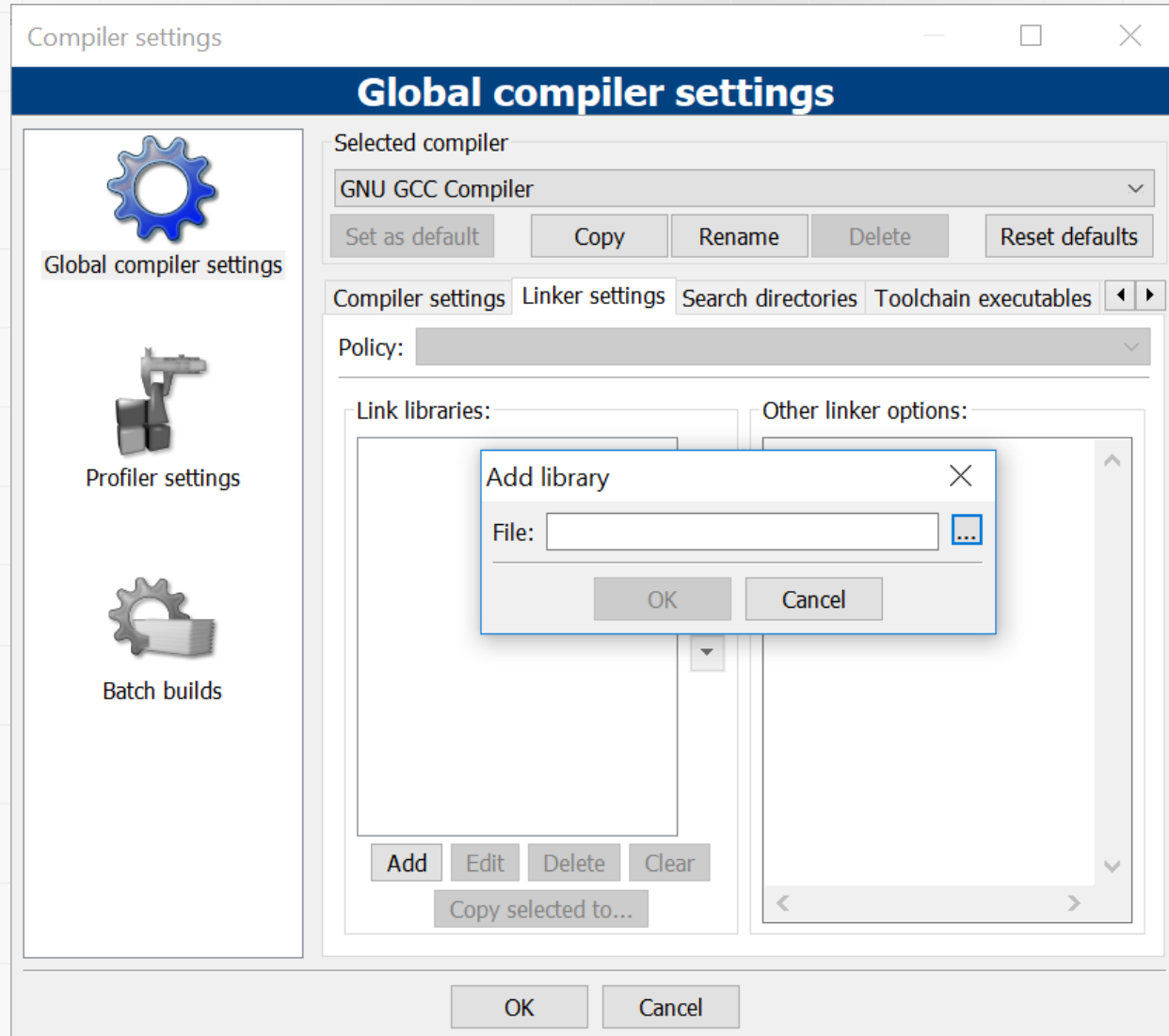


# Biblioteka GSL





# Biblioteka GSL





# Biblioteka GSL

Choose library to link

Ten komputer > Pulpit > 201718 > tp > gsl-2.3 > lib

Przeszukaj: lib

Organizuj Nowy folder

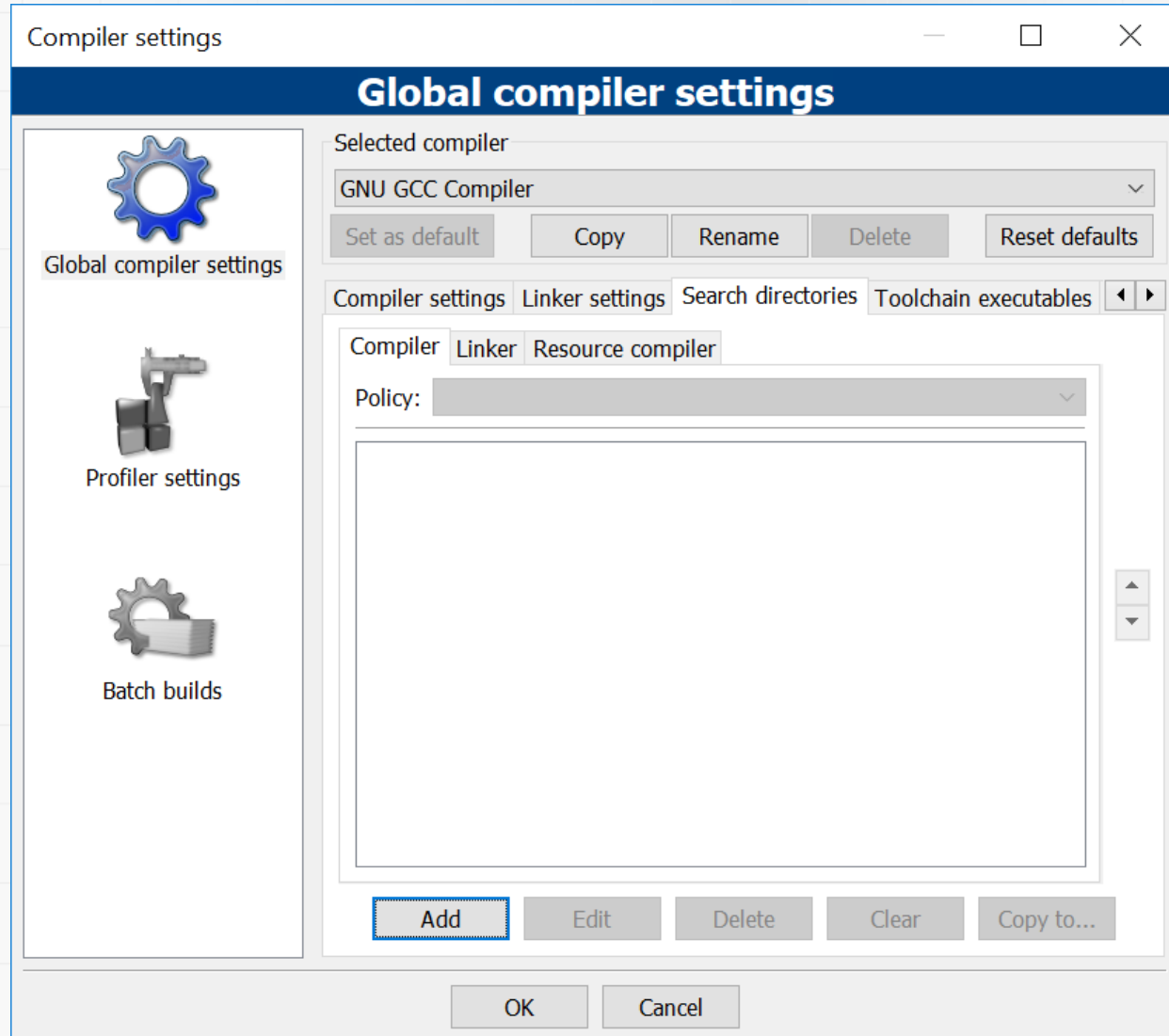
Nazwa	Data modyfikacji	Typ	Rozmiar
libgsl.a	29.04.2017 20:18	Plik A	10 367 KB
libgsl.dll.a	29.04.2017 20:18	Plik A	3 032 KB
libgslcblas.a	29.04.2017 20:16	Plik A	1 044 KB
libgslcblas.dll.a	29.04.2017 20:16	Plik A	88 KB

Nazwa pliku: "libgsl.a" "libgsl.dll.a" "libgslcblas.a" "libgslcblas.dll.a"

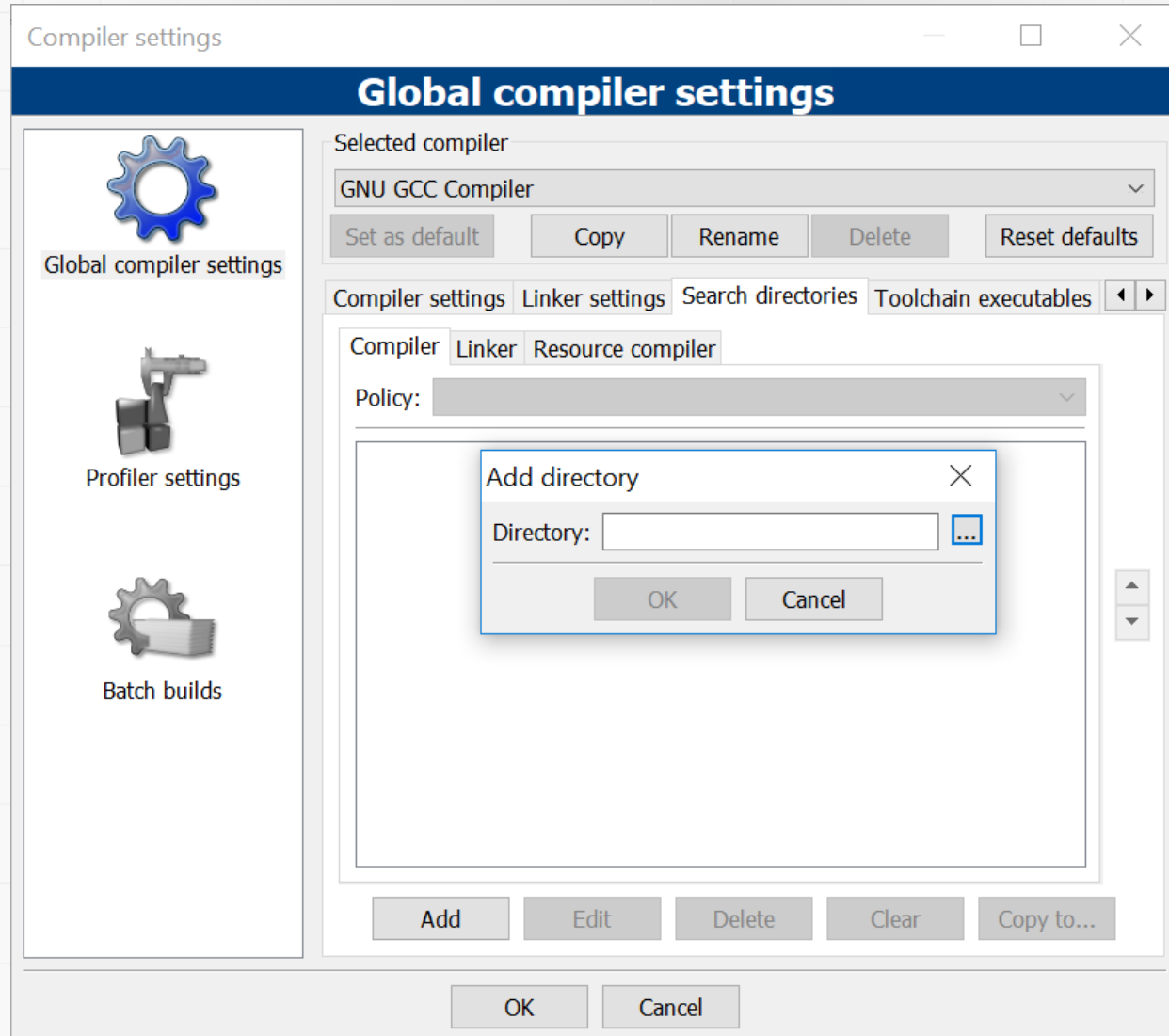
Library files (\*.a, \*.so, \*.lib, \*.dyli)

Otwórz Anuluj

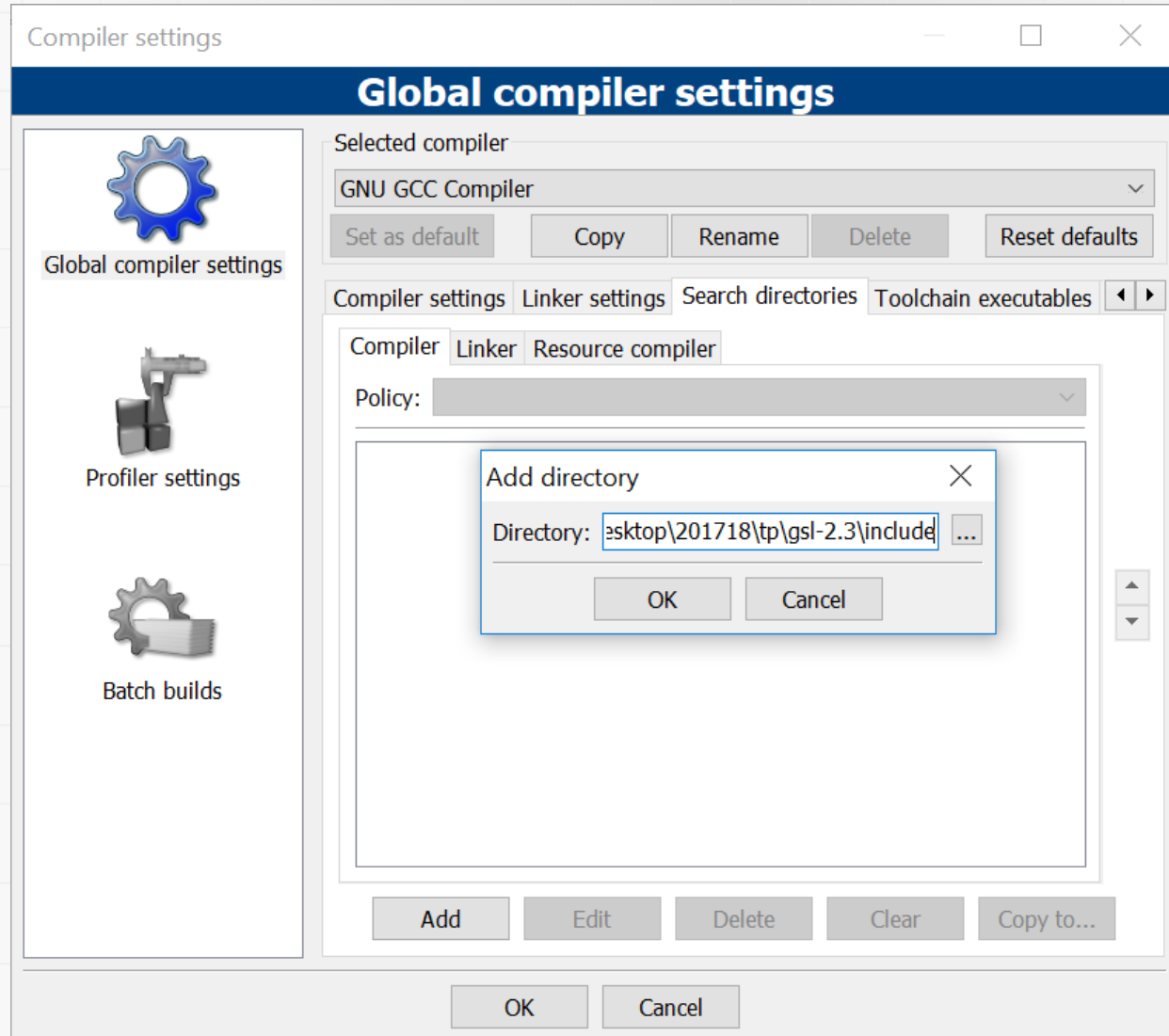
# Biblioteka GSL



# Biblioteka GSL



# Biblioteka GSL





# Biblioteka GSL

- Pliki nagłówkowe napisano w taki sposób, aby możliwe było wywoływanie biblioteki GSL bezpośrednio z programów języka C++

# Biblioteka GSL

## Przykładowy program

```
main.cpp x
1  #include <iostream>
2  #include <iomanip>
3  //dołączenie pliku nagłówkowego
4  #include <gsl/gsl_sf_bessel.h>
5
6  using namespace std;
7
8  int main (void)
9  {
10     double x = 5.0;
11     //wywołanie funkcji z biblioteki GSL
12     double y = gsl_sf_bessel_J0 (x);
13     cout << "J0(" << x << ") = ";
14     cout << setprecision(18) << scientific << y << endl;
15     return 0;
16 }
17
```



# Biblioteka GSL

## Wyznaczenie pierwiastków wielomianu

```
main.cpp x
1  /*Program wyznacza pierwiastki wielomianu
2  p(x) = -1 + x^5 korzystając z biblioteki GSL.*/
3  #include <iostream>
4  #include <iomanip>
5  #include <gsl/gsl_poly.h>
6
7  using namespace std;
8
9  int main (void)
10 {
11     int i;
12     /*Tablica a zawiera współczynniki wielomianu p(x).
13     Wielomian stopnia 5 ma 6 współczynników,
14     podawane są od wyrazu wolnego do współczynnika
15     przy najwyższej potędze.*/
16     double a[6] = { -1, 0, 0, 0, 0, 1 };
17
18     /*Współrzędne 5 miejsc zerowych będą zapisane
19     jako 5 liczb zespolonych, każda jako dwie liczby
20     typu double, razem 10-elementów.*/
21     double z[10];
22 }
```





# Biblioteka GSL

## Wyznaczenie pierwiastków wielomianu

```
main.cpp x
23  /*Alokacja pamięci roboczej wymaganej przez funkcję
24  gsl_poly_complex_solve().*/
25  gsl_poly_complex_workspace * w
26  = gsl_poly_complex_workspace_alloc (6);
27
28  /*Wywołanie funkcji
29  int gsl_poly_complex_solve(const double * a, size_t n, gsl_poly_complex_workspace * w, gsl_complex_packed_ptr z)*/
30  gsl_poly_complex_solve (a, 6, w, z);
31
32  /*Zwolnienie pamięci roboczej*/
33  gsl_poly_complex_workspace_free (w);
34
35  cout << setprecision(18);
36  for (i = 0; i < 5; i++){
37      cout << "z" << noshwpos << i << " = ";
38      cout << showpos << z[2*i] << z[2*i+1] << "i" << endl;
39  }
40
41  return 0;
42 }
```

# Biblioteka GSL

## Rozwiązywanie układu równań $Ax = b$

main.cpp ×

```
1  /*Program rozwiązuje układ równań  $Ax = b$ ,
2  dla podanej macierzy A rozmiaru (4x4)
3  oraz wektora b.*/
4  #include <iostream>
5  #include <gsl/gsl_linalg.h>
6
7  using namespace std;
8
9  int main (void)
10 {
11     /*16 liczb tworzących macierz A.*/
12     double a_data[] = { 0.18, 0.60, 0.57, 0.96,
13                        0.41, 0.24, 0.99, 0.58,
14                        0.14, 0.30, 0.97, 0.66,
15                        0.51, 0.13, 0.19, 0.85 };
16
17     /*Cztery liczby wektora b.*/
18     double b_data[] = { 1.0, 2.0, 3.0, 4.0 };
19 }
```

# Biblioteka GSL

## Rozwiązywanie układu równań $Ax = b$

main.cpp x

```
20      /*Struktury danych obsługujące macierz A i wektor b.*/
21      gsl_matrix_view m
22      = gsl_matrix_view_array (a_data, 4, 4);
23
24      gsl_vector_view b
25      = gsl_vector_view_array (b_data, 4);
26
27      /*Alokacja pamięci na wektor rozwiązań x.*/
28      gsl_vector *x = gsl_vector_alloc (4);
29
30      /*Struktura obsługująca permutacje (wykorzystywana
31      przez eliminację Gaussa z częściowym wyborem elementu głównego).*/
32      gsl_permutation * p = gsl_permutation_alloc (4);
33      /*Zmienna przechowująca znak permutacji.*/
34      int s;
35
```



# Biblioteka GSL

## Rozwiązywanie układu równań $Ax = b$

```
main.cpp x
36  /*Funkcja:
37  int gsl_linalg_complex_LU_decomp(gsl_matrix_complex * A, gsl_permutation * p, int * signum)
38  Znajduje rozkład LU macierzy kwadratowej A:
39  PA = LU
40  Po wywołaniu macierz A zawiera macierz U w górnym trójkącie (z diagonalą).
41  Dolny trójkąt (bez diagonali) zawiera macierz L.
42  Elementy diagonalne macierzy L to jedynki (nie są przechowywane w pamięci).
43
44  Informacje o permutacjach są zapisane w strukturze p.
45  signum - znak permutacji.
46
47  Funkcja używa algorytmu eliminacji Gaussa z częściowym wyborem elementu głównego
48  (Golub & Van Loan, Matrix Computations, Algorithm 3.4.1).*/
49
50  gsl_linalg_LU_decomp (&m.matrix, p, &s);
51
52  /*Funkcja:
53  int gsl_linalg_LU_solve(const gsl_matrix * LU, const gsl_permutation * p, const gsl_vector * b, gsl_vector * x)
54  Rozwiązuje układ równań Ax = b.
55  Na podstawie rozkładu LU, permutacji p i wektora b oblicza wektor x.*/
56
57  gsl_linalg_LU_solve (&m.matrix, p, &b.vector, x);
58
```

# Biblioteka GSL

## Rozwiązywanie układu równań $Ax = b$

main.cpp x

```
59      /*Wypisanie wektora rozwiązań na standardowe wyjście (stdout)*/
60      cout << "x = " << endl;
61      gsl_vector_fprintf (stdout, x, "%g");
62
63      /*Zwolnienie pamięci roboczej*/
64      gsl_permutation_free (p);
65      gsl_vector_free (x);
66      return 0;
67  }
```



# Biblioteka GSL

## Rozwiązywanie równań różniczkowych zwyczajnych

- ogólna postać układu równań różniczkowych

$$\frac{dy_i(t)}{dt} = f_i(t, y_1(t), \dots, y_n(t)) \text{ dla } i = 1, \dots, n$$

- macierz Jacobiego

$$J_{ij} = \frac{\partial f_i(t, y(t))}{\partial y_j}$$

# Biblioteka GSL

## Rozwiązywanie równań różniczkowych zwyczajnych

- Przykład - równanie nieliniowego oscylatora Van der Pola

$$u''(t) + \mu u'(t) \left( (u(t))^2 - 1 \right) + u(t) = 0$$

po podstawieniu  $v = u'(t)$

otrzymujemy układ równań

$$u' = v$$

$$v' = -u + \mu v (1 - u^2)$$

# Biblioteka GSL

## Rozwiązywanie równań różniczkowych zwyczajnych

- Program rozwiązuje zagadnienie początkowe: układ równań różniczkowych dla  $\mu = 10$  z warunkiem początkowym  $(u, v) = (1, 0)$  na zakresie zmiennej niezależnej  $t$  od 0 do 100.
- Program wyświetla rozwiązania - wartości funkcji  $u$  oraz  $v$  - dla  $t_i = 1, 2, \dots, 100$ .



# Biblioteka GSL

## Rozwiązywanie równań różniczkowych zwyczajnych

```
main.cpp x
1  #include <iostream>
2  #include <iomanip>
3  #include <gsl/gsl_errno.h>
4  #include <gsl/gsl_matrix.h>
5  #include <gsl/gsl_odeiv2.h>
6
7  using namespace std;
8
9  /*Definicja funkcji obliczającej wektor prawych stron układu równań*/
10 int
11 func (double t, const double y[], double f[],
12       void *params)
13 {
14     (void) (t); /* avoid unused parameter warning */
15     double mu = *(double *)params;
16     f[0] = y[1];
17     f[1] = -y[0] - mu*y[1]*(y[0]*y[0] - 1);
18     return GSL_SUCCESS;
19 }
20
```



# Biblioteka GSL

## Rozwiązywanie równań różniczkowych zwyczajnych

```
main.cpp ×
21  /*Definicja funkcji obliczającej macierz Jacobiego*/
22  int
23  jac (double t, const double y[], double *dfdy,
24       double dfdt[], void *params)
25  {
26      (void) (t); /* avoid unused parameter warning */
27      double mu = *(double *)params;
28      gsl_matrix_view dfdy_mat
29      = gsl_matrix_view_array (dfdy, 2, 2);
30      gsl_matrix * m = &dfdy_mat.matrix;
31      gsl_matrix_set (m, 0, 0, 0.0);
32      gsl_matrix_set (m, 0, 1, 1.0);
33      gsl_matrix_set (m, 1, 0, -2.0*mu*y[0]*y[1] - 1.0);
34      gsl_matrix_set (m, 1, 1, -mu*(y[0]*y[0] - 1.0));
35      dfdt[0] = 0.0;
36      dfdt[1] = 0.0;
37      return GSL_SUCCESS;
38  }
39
```



# Biblioteka GSL

## Rozwiązywanie równań różniczkowych zwyczajnych

```
main.cpp x
40  int
41  main (void)
42  {
43      double mu = 10;
44      /*Definicja układu równań, w strukturze znajdują się kolejno:
45       - funkcja obliczająca prawe strony równania,
46       - funkcja obliczająca macierz Jacobiego,
47       - wymiar układu,
48       - parametry.*/
49      gsl_odeiv2_system sys = {func, jac, 2, &mu};
50
51      /*Alokacja struktury sterującej rozwiązywaniem układu równań.
52       Argumentami są:
53       - struktura przechowująca układ równań,
54       - określenie rodzaju kroku (w tym przypadku metoda Rungego-Kutty
55         Prince'a-Dormanda rzędu 8/9),
56       - początkowa wartość kroku,
57       - dokładność względna,
58       - dokładność bezwzględna.*/
59
60      gsl_odeiv2_driver * d =
61      gsl_odeiv2_driver_alloc_y_new (&sys, gsl_odeiv2_step_rk8pd,
62                                   1e-6, 1e-6, 0.0);
```

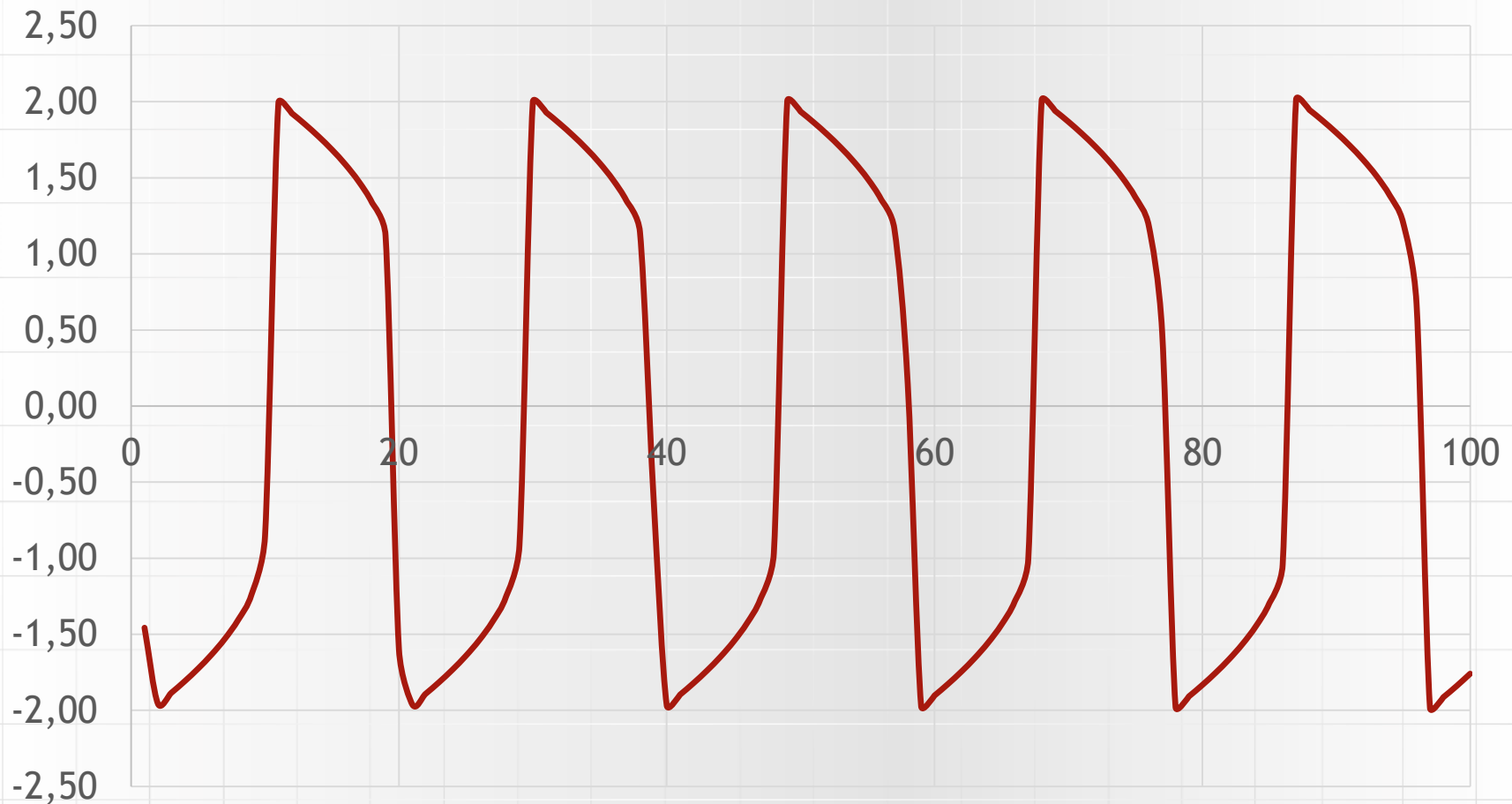
# Biblioteka GSL

## Rozwiązywanie równań różniczkowych zwyczajnych

```
main.cpp x
63     double t = 0.0, t1 = 100.0;
64     double y[2] = { 1.0, 0.0 };
65
66     cout << setprecision(5) << scientific;
67
68     for (int i = 1; i <= 100; i++)
69     {
70         double ti = i * t1 / 100.0;
71         /*Przejscie sterownikiem d od aktualnej wartosci t
72         do wartosci ti. Wartości początkowe w y zastapione są
73         wartościami końcowymi.*/
74         int status = gsl_odeiv2_driver_apply (d, &t, ti, y);
75
76         if (status != GSL_SUCCESS)
77         {
78             cout << "error, return value=" << status << endl;
79             break;
80         }
81
82         cout << t << " " << y[0] << " " << y[1] << endl;
83     }
84
85     gsl_odeiv2_driver_free (d);
86     return 0;
87 }
```

# Biblioteka GSL

## Rozwiązywanie równań różniczkowych zwyczajnych





# Wykorzystanie debuggera

- Przykładowy projekt - program oblicza odsetki od podanej kwoty przy zadanym oprocentowaniu rocznym przez ustaloną liczbę lat



# Wykorzystanie debuggera

main.c x

```
1  /*Przykładowy projekt do demonstracji debugowania.*/
2
3  #include <stdio.h>
4
5  double obliczOdsetki(double podstawa, double oprocentowanie, int lata);
6
7  int main(){
8      double podstawa;
9      double oprocentowanie;
10     int lata;
11     printf("Podaj podstawe: ");
12     scanf("%lf", &podstawa);
13     printf("Podaj oprocentowanie: ");
14     scanf("%lf", &oprocentowanie);
15     printf("Podaj liczbe lat: ");
16     scanf("%d", &lata);
17     printf("Po %d latach bedziesz miec %f PLN.\n", lata,
18           obliczOdsetki(podstawa, oprocentowanie, lata ));
19     return 0;
20 }
```



# Wykorzystanie debuggera

```
main.c x
22  double obliczOdsetki(double podstawa, double oprocentowanie, int lata){
23      int i;
24      double koncowy_mnoznic;
25      for(i=0; i<lata; i++){
26          koncowy_mnoznic *= (1 + oprocentowanie);
27      }
28      return podstawa * koncowy_mnoznic;
29  }
```

Podaj podstawe: 100

Podaj oprocentowanie: .1

Podaj liczbe lat: 1

Po 1 latach bedziesz miec 0.000000 PLN.





# Wykorzystanie debuggera

## Konfiguracja

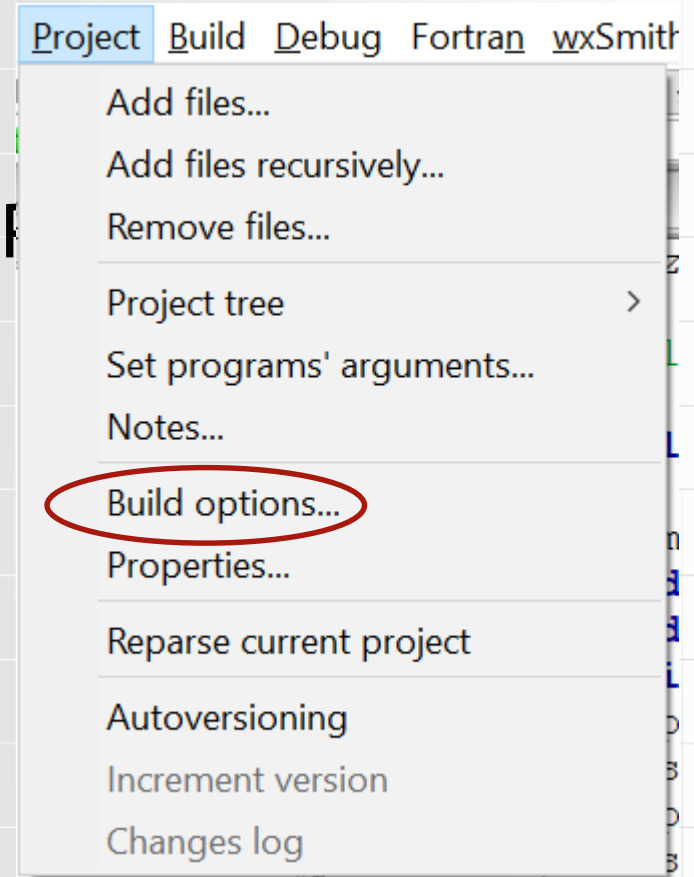
- Symbole debuggowania  
Project >> Build options >> Produce debugging symbols [-g] (zaznaczone)



# Wykorzystanie debuggera

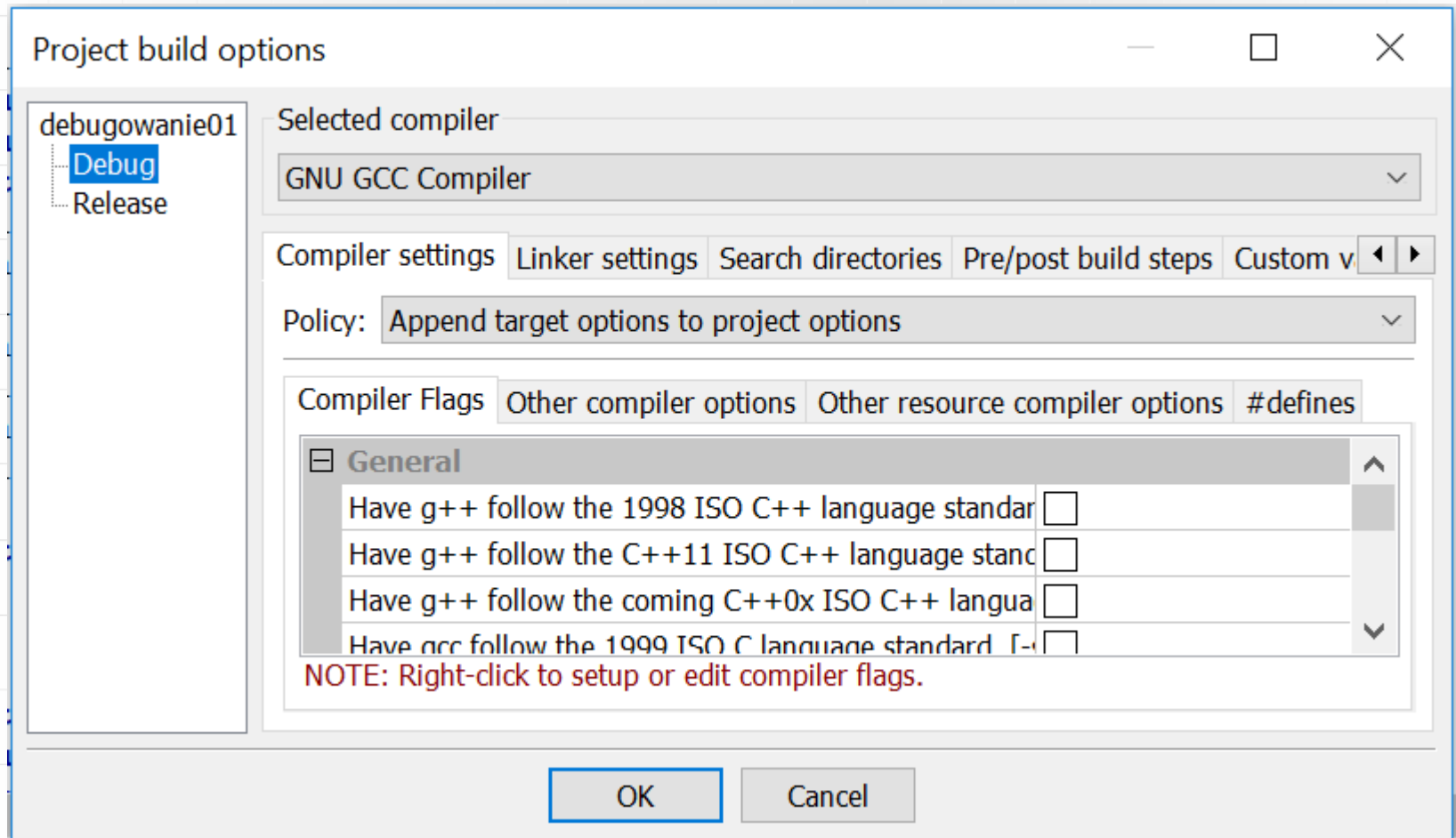
## Konfiguracja

- Symbole debuggowania  
Project >> Build options >> symbols [-g] (zaznaczone)



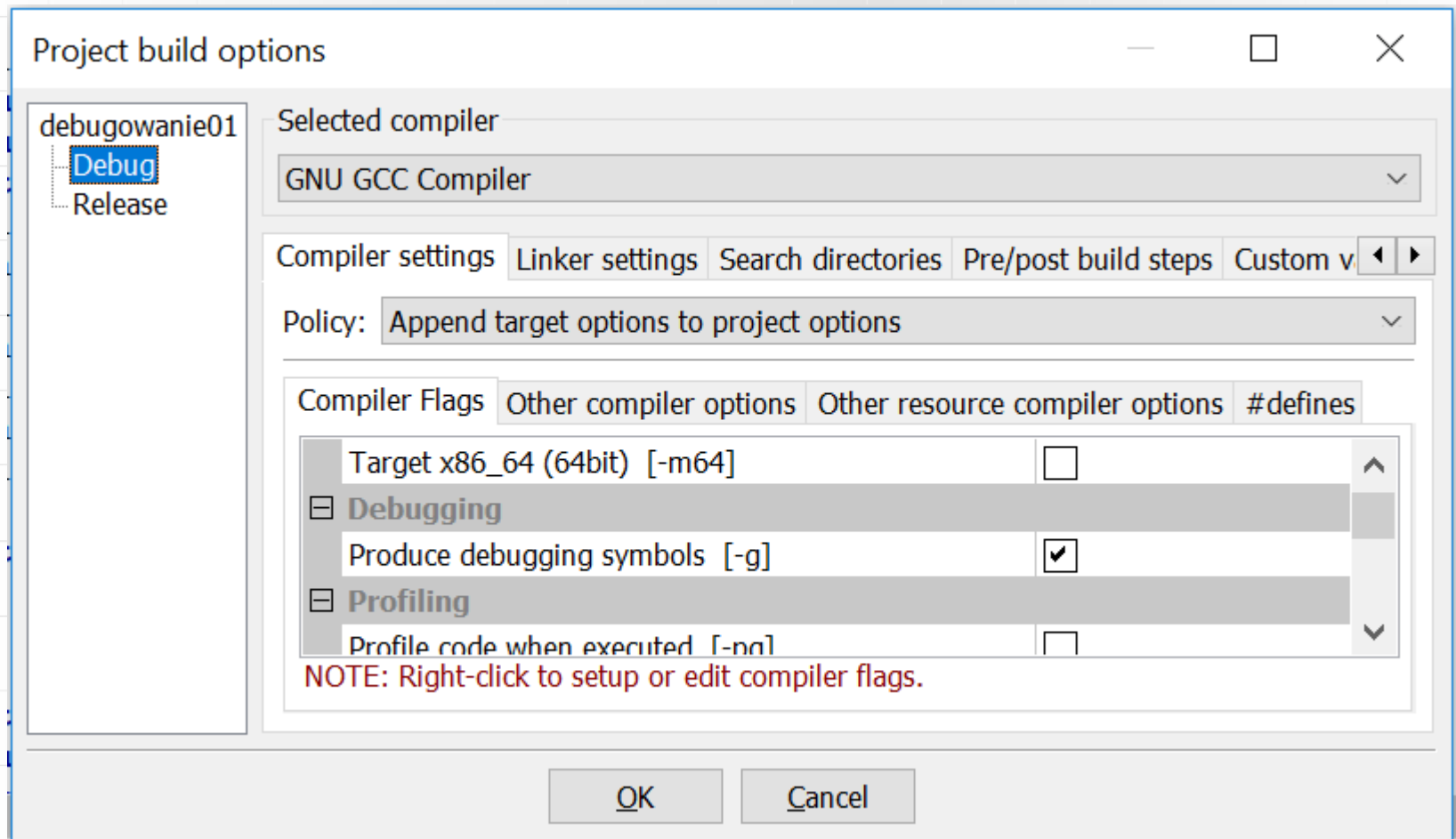
# Wykorzystanie debuggera

## Konfiguracja



# Wykorzystanie debuggera

## Konfiguracja



# Wykorzystanie debuggera

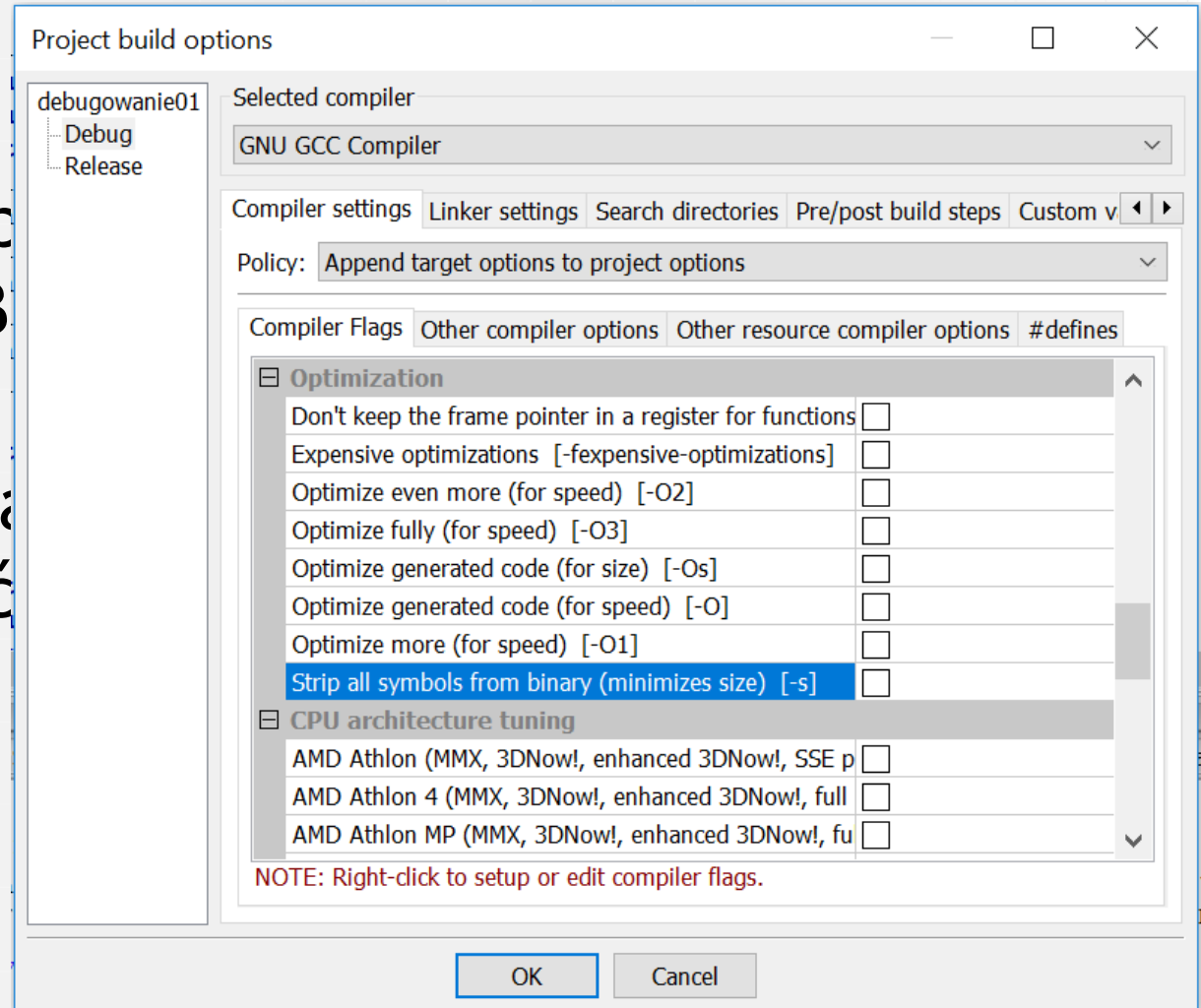
## Konfiguracja

- Symbole debuggowania  
Project >> Build options >> Produce debugging symbols [-g] (zaznaczone)
- Opcja Strip all symbols from binary [-s] nie powinna być zaznaczona

# Wykorzystanie debuggera

## Konfiguracja

- Symbole debugowania  
Project >> Build Options  
symbols [-g]
- Opcja Strip symbolów  
powinna być wybrana



# Wykorzystanie debuggera

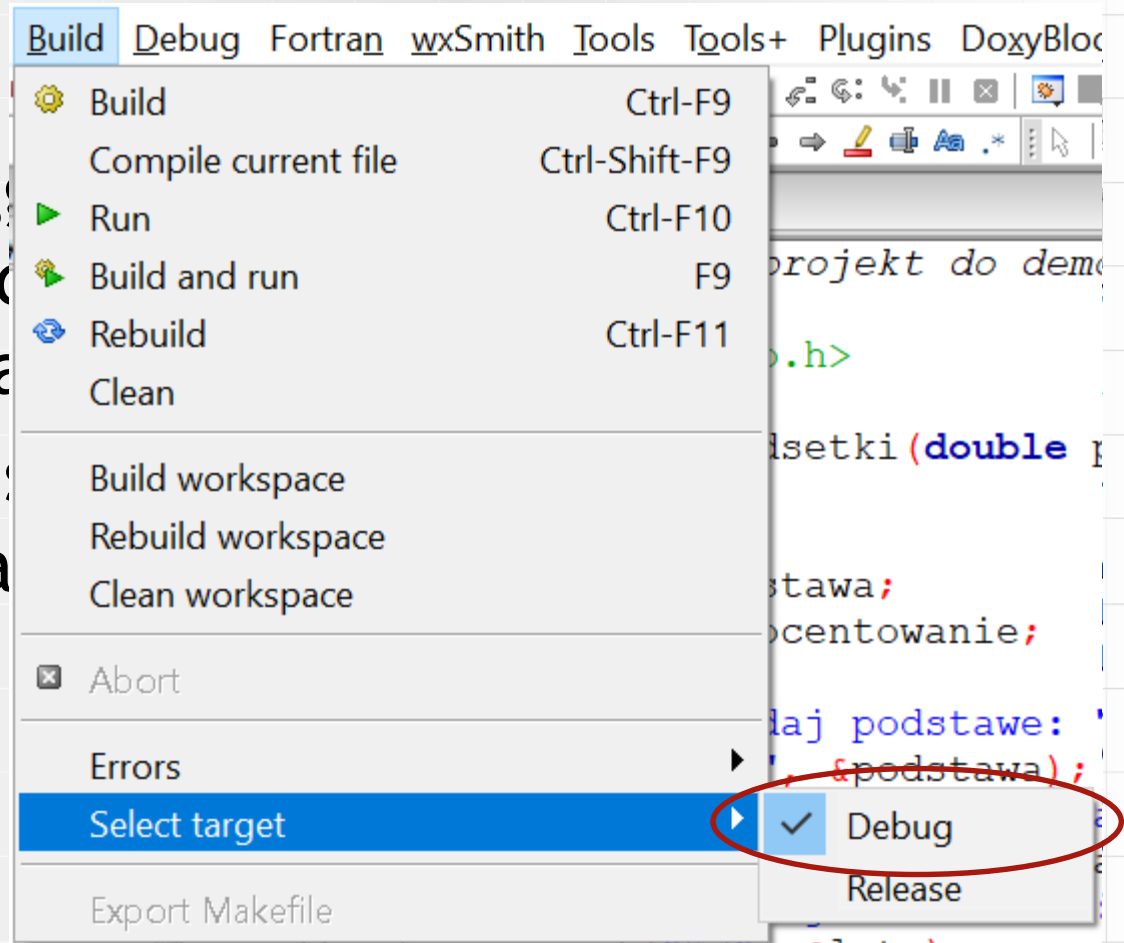
## Konfiguracja

- Symbole debuggowania  
Project >> Build options >> Produce debugging symbols [-g] (zaznaczone)
- Opcja Strip all symbols from binary [-s] nie powinna być zaznaczona
- Cel kompilacji  
Build >> Select target >> Debug

# Wykorzystanie debuggera

## Konfiguracja

- Symbole debugera  
Project >> Build  
symbols [-g] (za
- Opcja Strip all  
powinna być za
- Cel kompilacji  
Build >> Select







# Wykorzystanie debuggera

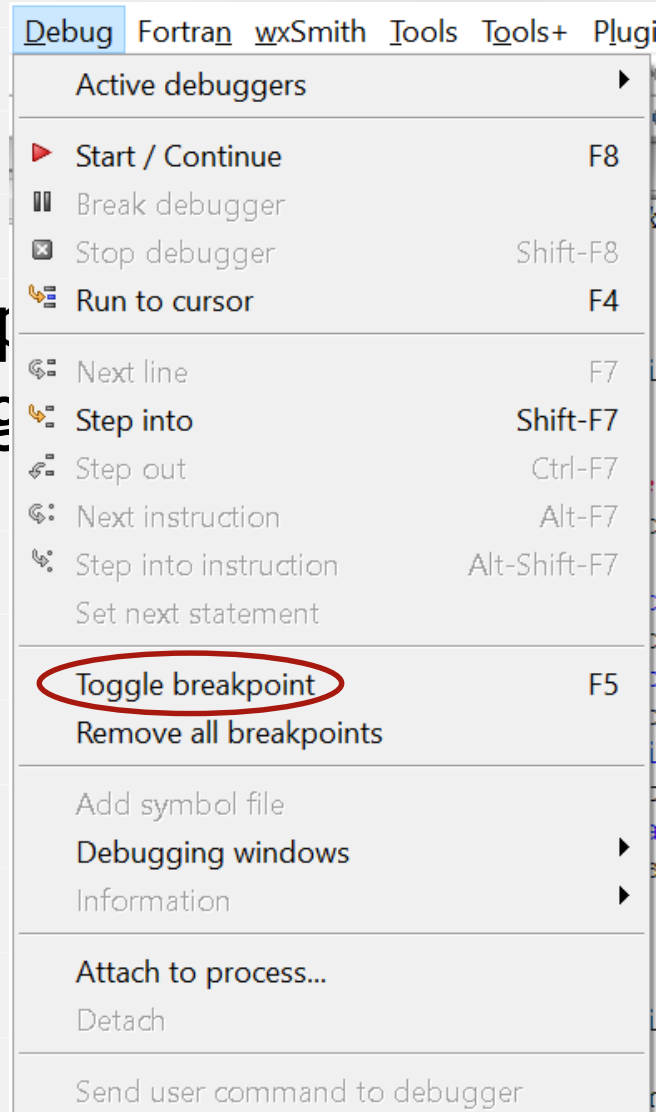
## Wstrzymywanie działania

- Definiowanie punktu wstrzymania  
Debug >> Toggle breakpoint  
lub F5

# Wykorzystanie debuggera

## Wstrzymywanie działania

- Definiowanie p  
Debug >> Togg  
lub F5





# Wykorzystanie debuggera

## Wstrzymywanie działania

- De  
De  
lu

```
7 int main() {  
8     double podstawa;  
9     double oprocentowanie;  
10    int lata;  
11    printf("Podaj podstawe: ");  
12    scanf("%lf", &podstawa);  
13    printf("Podaj oprocentowanie: ");  
14    scanf("%lf", &oprocentowanie);  
15    printf("Podaj liczbe lat: ");  
16    scanf("%d", &lata);  
17    printf("Po %d latach bedziesz miec %f PLN.\n", lata,  
18           obliczOdsetki(podstawa, oprocentowanie, lata ));  
19    return 0;  
20 }
```

# Wykorzystanie debuggera

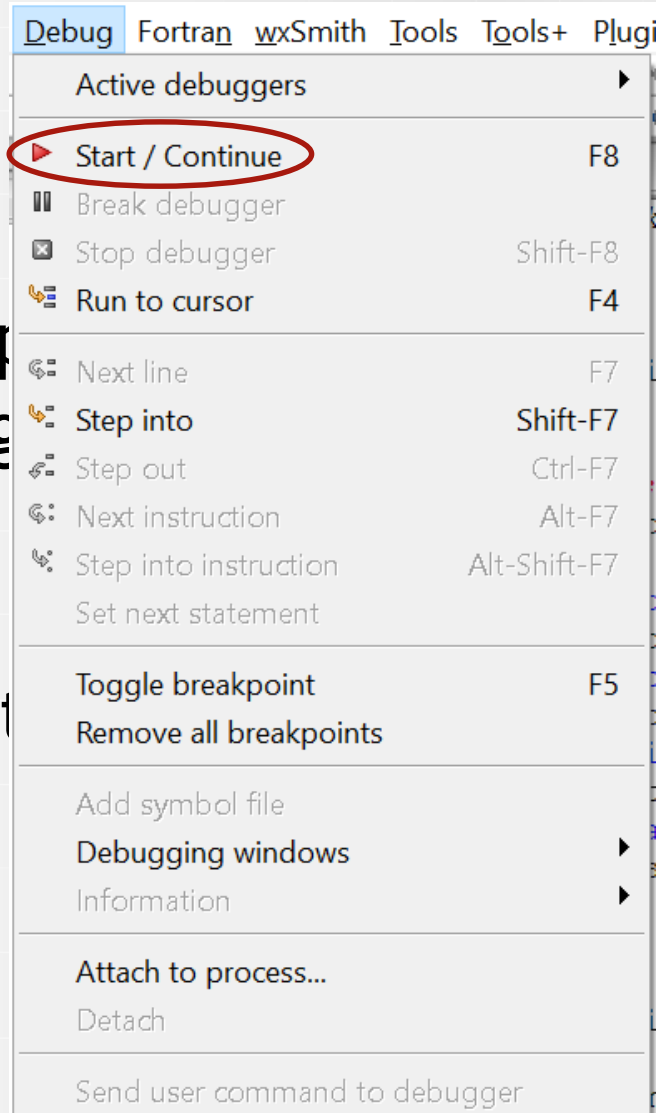
## Wstrzymywanie działania

- Definiowanie punktu wstrzymania  
Debug >> Toggle breakpoint  
lub F5
- Uruchomienie programu  
Debug >> Start  
lub F8

# Wykorzystanie debuggera

## Wstrzymywanie działania

- Definiowanie punktów zatrzymania  
Debug >> Toggle breakpoint  
lub F5
- Uruchomienie debugera  
Debug >> Start / Continue  
lub F8





# Wykorzystanie debuggera

## Wstrzymywanie działania

- De

- De

- lu

- U

- De

- lu

```
7 int main(){
8     double podstawa;
9     double oprocentowanie;
10    int lata;
11    printf("Podaj podstawe: ");
12    scanf("%lf", &podstawa);
13    printf("Podaj oprocentowanie: ");
14    scanf("%lf", &oprocentowanie);
15    printf("Podaj liczbe lat: ");
16    scanf("%d", &lata);
17    printf("Po %d latach bedziesz miec %f PLN.\n", lata,
18           obliczOdsetki(podstawa, oprocentowanie, lata ));
19    return 0;
20 }
```

# Wykorzystanie debuggera

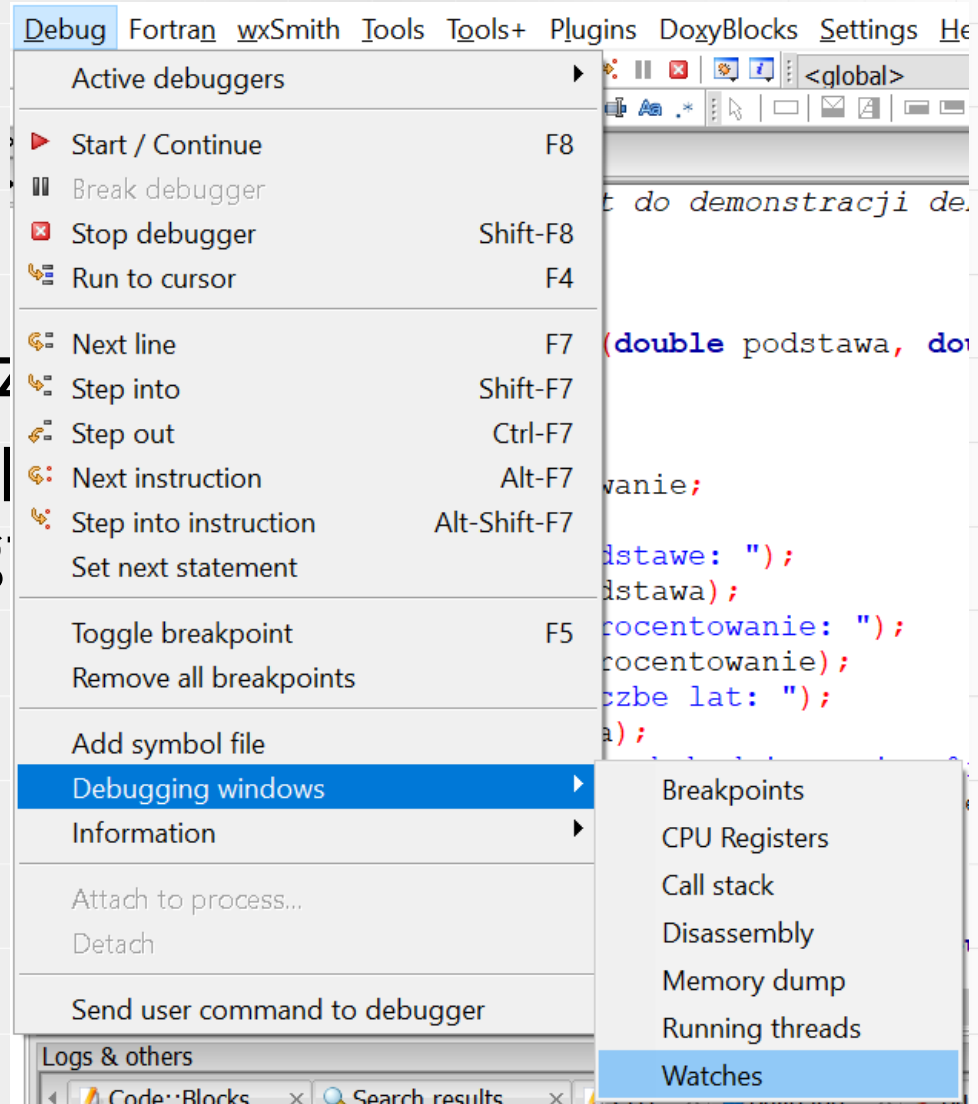
## Okna debuggera

- Okna debuggera zawierają różne informacje o wykonywanym programie  
Debug >> Debugging windows

# Wykorzystanie debuggera

## Okna debuggera

- Okna debuggera z oknem wykonywanym kodem
- Debug >> Debugging windows







# Wykorzystanie debuggera

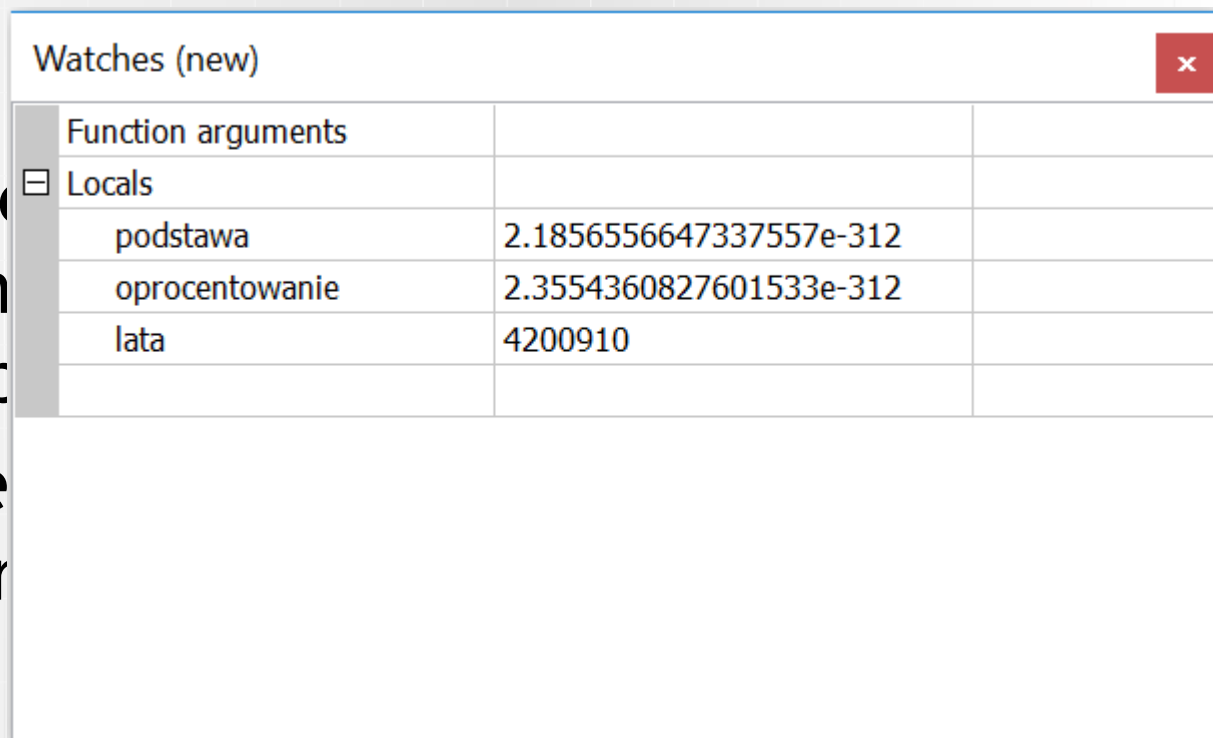
## Okna debuggera

- Okna debuggera zawierają różne informacje o wykonywanym programie  
Debug >> Debugging windows
- Okno Watches pokazuje wartości zmiennych lokalnych i argumentów funkcji

# Wykorzystanie debuggera

## Okna debuggera

- Okna debuggera do wykonywania Debug >> Debug
- Okno Watches lokalnych i argumentów



Watches (new)	
Function arguments	
Locals	
podstawa	2.1856556647337557e-312
oprocentowanie	2.3554360827601533e-312
lata	4200910

# Wykorzystanie debuggera

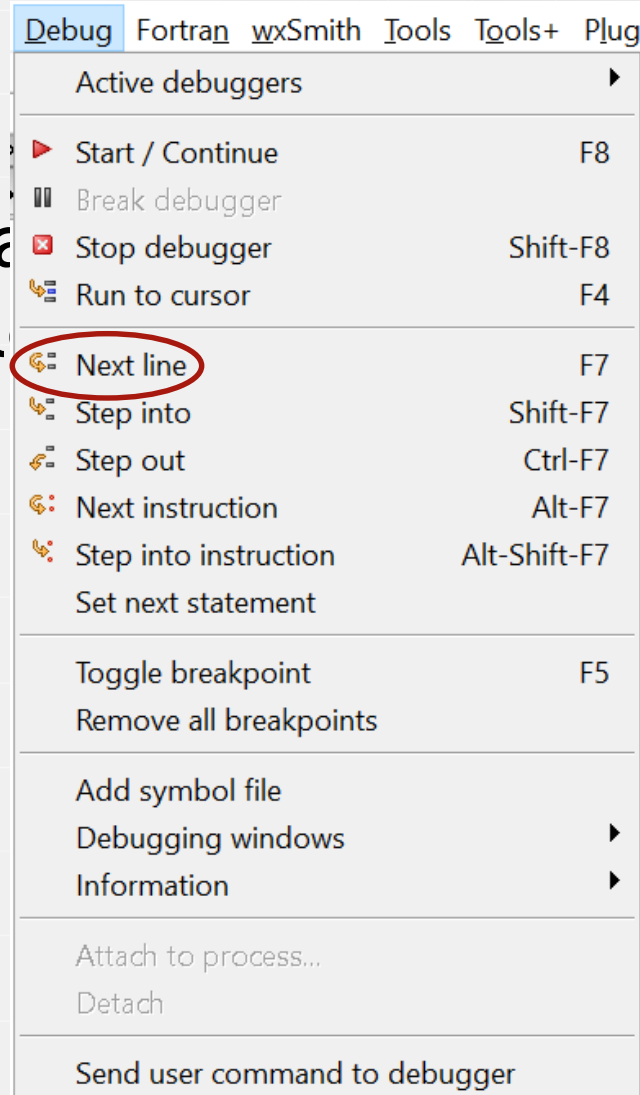
## Śledzenie działania programu

- W celu wykonania następnych wierszy programu wykonamy polecenie Next line (F7)

# Wykorzystanie debuggera

## Śledzenie działania programu

- W celu wykonania następnego programu (7)



# Wykorzystanie debuggera

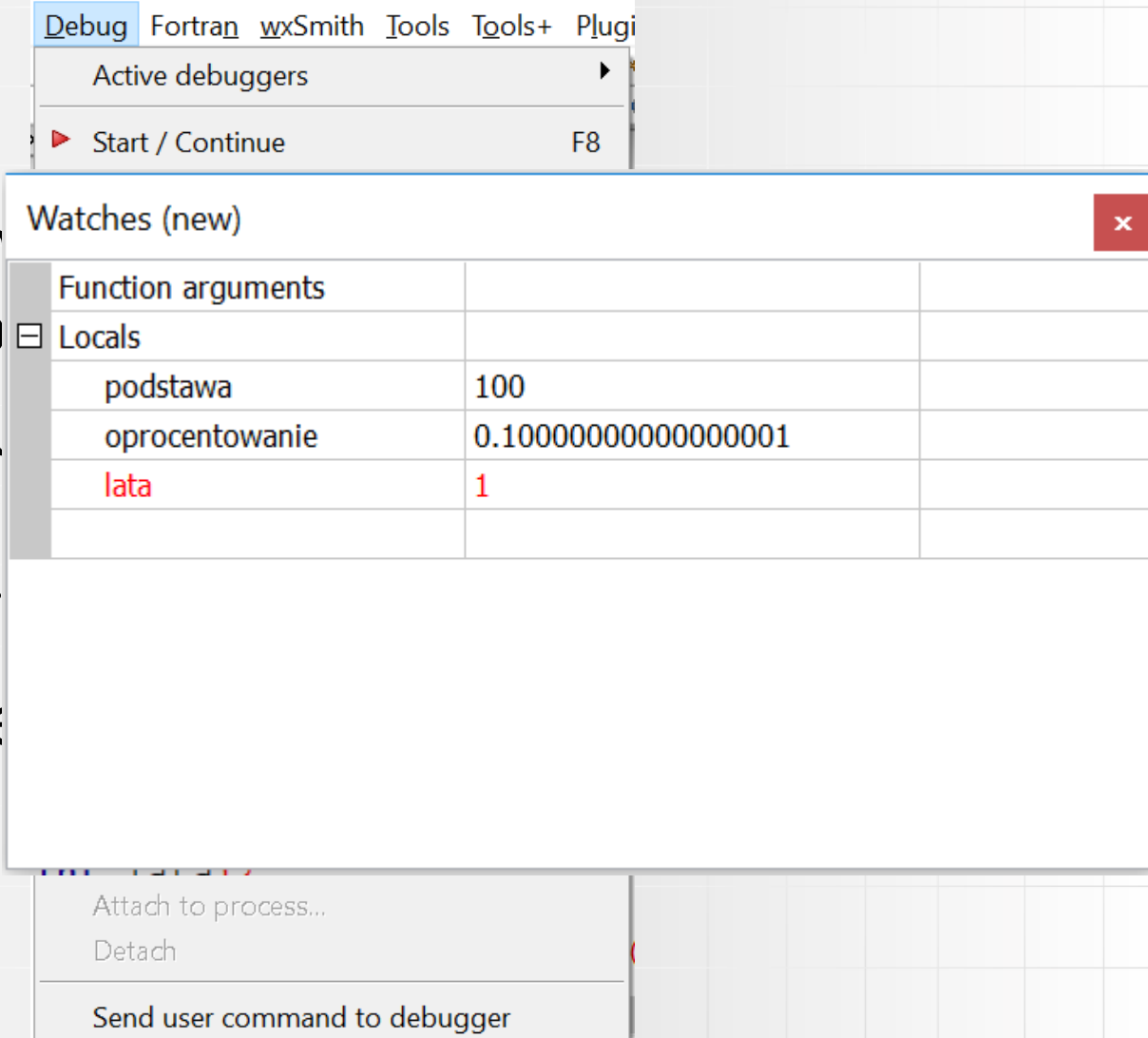
## Śledzenie działania programu

- W celu wykonania następnych wierszy programu wykonamy polecenie Next line (F7)
- Po osiągnięciu linii 12 program oczekuje na podanie danych (funkcja scanf())
- Podobnie w liniach 14 i 16
- Wartości zmiennych możemy sprawdzić w oknie Watches

# Wykorzystanie debuggera

## Śledzenie działania programu

- W celu wykor...
  - Po osiągnięciu...
  - Podobnie w li...
  - Wartości zmie...
- Watches



The screenshot shows a debugger interface. At the top, the 'Debug' menu is open, displaying options: 'Active debuggers', 'Start / Continue' (with a red play button icon and 'F8' shortcut), 'Attach to process...', 'Detach', and 'Send user command to debugger'. Below the menu, the 'Watches (new)' window is visible. It contains a table with the following data:

Function arguments	
Locals	
podstawa	100
oprocentowanie	0.100000000000000001
lata	1



# Wykorzystanie debuggera

## Śledzenie działania programu

- Po osiągnięciu linii 17 wywołamy polecenie Step into, które służy do wejścia do funkcji, która ma zostać wywołana z bieżącej wiersza



# Wykorzystanie debuggera

## Śledzenie działania programu

- Po  
Ste  
któ

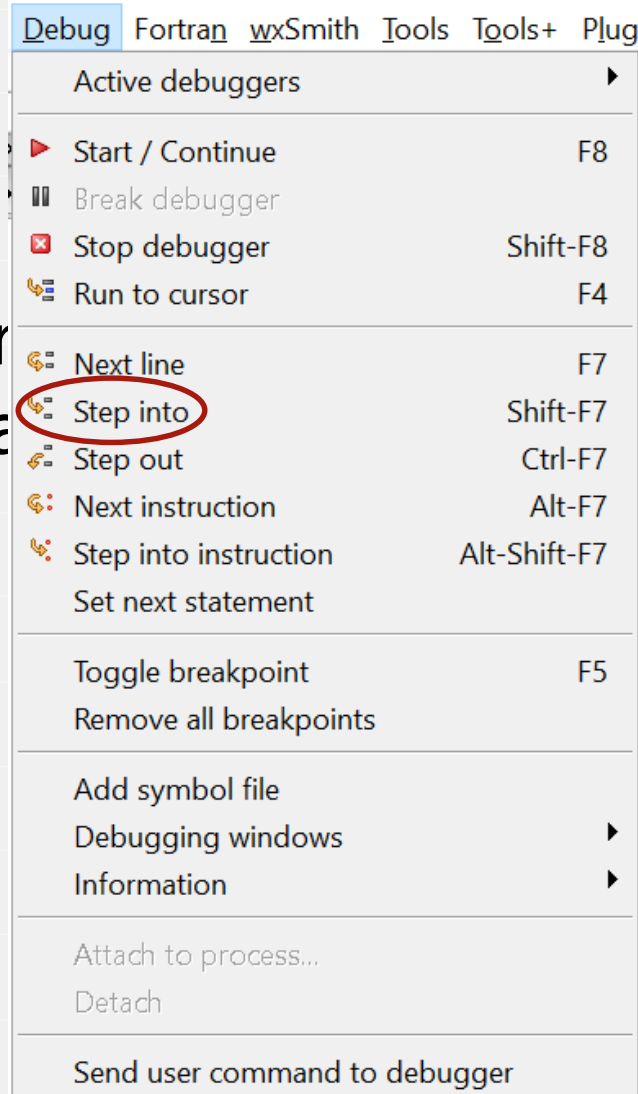
```
7 int main(){
8     double podstawa;
9     double oprocentowanie;
10    int lata;
11    printf("Podaj podstawe: ");
12    scanf("%lf", &podstawa);
13    printf("Podaj oprocentowanie: ");
14    scanf("%lf", &oprocentowanie);
15    printf("Podaj liczbe lat: ");
16    scanf("%d", &lata);
17    printf("Po %d latach bedziesz miec %f PLN.\n", lata,
18           obliczOdsetki(podstawa, oprocentowanie, lata ));
19    return 0;
20 }
```



# Wykorzystanie debuggera

## Śledzenie działania programu

- Po osiągnięciu Step into, którą ma zostać



polecenie do funkcji, którego wiersza



# Wykorzystanie debuggera

## Śledzenie działania programu

- Po osiągnięciu linii 17 wywołamy polecenie Step into, które służy do wejścia do funkcji, która ma zostać wywołana z bieżącej wiersza
- Sterowanie w programie przejdzie do linii 25 (pierwsza instrukcja w wywołanej funkcji)
- Zmienne `i` oraz `koncowy_mnoznik` są niezainicjalizowane

# Wykorzystanie debuggera

## Śledzenie działania programu

Do osiągnięcia linii 17 wywołamy polecenie

```
22 double obliczOdsetki(double podstawa, double oprocentowanie, int lata){
23     int i;
24     double koncowy_mnoznik;
25     for(i=0; i<lata; i++){
26         koncowy_mnoznik *= (1 + oprocentowanie);
27     }
28     return podstawa * koncowy_mnoznik;
29 }
30
31
32
33
34
35
36
37
38
39
```

Watches (new)

Function arguments		
podstawa	100	
oprocentowanie	0.10000000000000001	
lata	1	
Locals		
i	6356692	
koncowy_mnoznik	1.8010879973037785e-307	

niezaimplementowane

# Wykorzystanie debuggera

## Śledzenie działania programu

- Po przejściu do kolejnego wiersza (F7) widzimy, że zmienna `i` została zainicjalizowana, natomiast wartość zmiennej `koncowy_mnoznik` pozostała bez zmian

# Wykorzystanie debuggera

## Śledzenie działania programu

```
22  □ double obliczOdsetki(double podstawa, double oprocentowanie, int lata){  
23      int i;  
24      double koncowy_mnoznik;  
25      □ for(i=0; i<lata; i++){  
26  ▶          koncowy_mnoznik *= (1 + oprocentowanie);  
27      }  
28      return podstawa * koncowy_mnoznik;  
29  }  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39
```

Watches (new)

□ Function arguments		
podstawa	100	
oprocentowanie	0.100000000000000001	
lata	1	
□ Locals		
i	0	
koncowy_mnoznik	1.8010879973037785e-307	

# Wykorzystanie debuggera

## Śledzenie działania programu

- Po przejściu do kolejnego wiersza (F7) widzimy, że zmienna `i` została zainicjalizowana, natomiast wartość zmiennej `koncowy_mnoznik` pozostała bez zmian
- Znaleźliśmy błąd - niezainicjalizowaną zmienną - wykorzystując debugger

# Wykorzystanie debuggera

## Debugowanie awarii

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct Node{
5      int v;
6      struct Node* next;
7  };
8
9  typedef struct Node Node;
10
11 void printList(const Node*);
12
```

# Wykorzystanie debuggera

## Debugowanie awarii

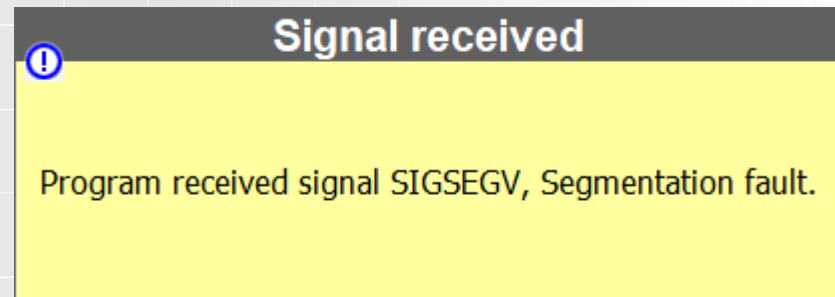
```
main.c x
13  int main()
14  {
15      Node* head;
16      head = malloc(sizeof(Node));
17      head->v = 10;
18      head->next = malloc(sizeof(Node));
19      head->next->v = 11;
20      printList(head);
21      return 0;
22  }
23
24  void printList(const Node* n) {
25      if(n != NULL) {
26          printf("%d\n", n->v);
27          printList(n->next);
28      }
29  }
```



# Wykorzystanie debuggera

## Debugowanie awarii

- Ten program nie działa - kończy działanie w nieprawidłowy sposób
- Błędu można poszukiwać wykorzystując debugger
- Debugger zgłosi błąd segmentacji





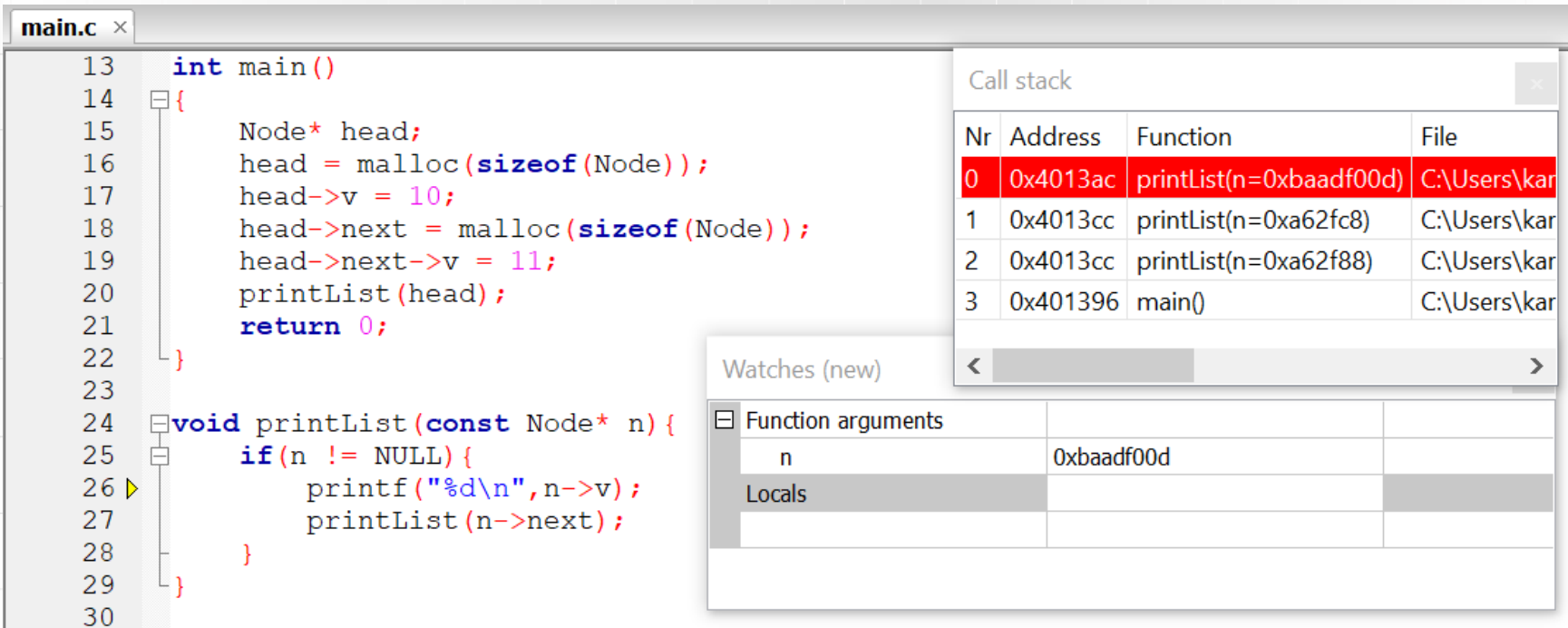
# Wykorzystanie debuggera

## Debugowanie awarii

- Debugger zatrzymał się w miejscu, w którym nastąpiła błędna instrukcja (linia 26)

# Wykorzystanie debuggera

## Debugowanie awarii



The screenshot displays a debugger window with the following components:

- Code Editor:** Shows the source code for `main.c`. The `main` function (lines 13-22) initializes a linked list with two nodes containing values 10 and 11, and calls `printList`. The `printList` function (lines 24-29) iterates through the list, printing each node's value and its next pointer.
- Call Stack:** A table showing the sequence of function calls. The current frame is highlighted in red.
- Watches (new):** A table showing the value of the argument `n` as `0xbaadf00d`.

Nr	Address	Function	File
0	0x4013ac	printList(n=0xbaadf00d)	C:\Users\kar
1	0x4013cc	printList(n=0xa62fc8)	C:\Users\kar
2	0x4013cc	printList(n=0xa62f88)	C:\Users\kar
3	0x401396	main()	C:\Users\kar

Function arguments		
n	0xbaadf00d	
Locals		

```
13 int main()
14 {
15     Node* head;
16     head = malloc(sizeof(Node));
17     head->v = 10;
18     head->next = malloc(sizeof(Node));
19     head->next->v = 11;
20     printList(head);
21     return 0;
22 }
23
24 void printList(const Node* n) {
25     if(n != NULL) {
26         printf("%d\n", n->v);
27         printList(n->next);
28     }
29 }
30
```

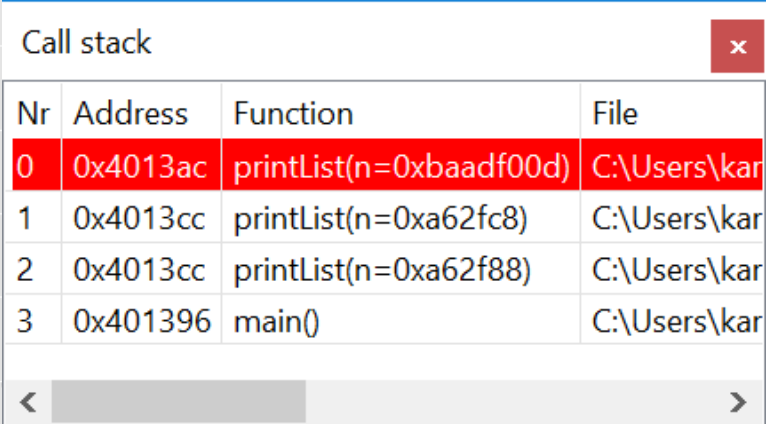
# Wykorzystanie debuggera

## Debugowanie awarii

- Debugger zatrzymał się w miejscu, w którym nastąpiła błędna instrukcja (linia 26)
- Wskaźnik `n` ma wartość `0xbaadf00d` (bad food)
  - jest to specjalna wartość używana podczas uruchamiania programów pod kontrolą debuggera do inicjalizowania pamięci

# Wykorzystanie debuggera

## Debugowanie awarii



Call stack

Nr	Address	Function	File
0	0x4013ac	printList(n=0xbaadf00d)	C:\Users\kar
1	0x4013cc	printList(n=0xa62fc8)	C:\Users\kar
2	0x4013cc	printList(n=0xa62f88)	C:\Users\kar
3	0x401396	main()	C:\Users\kar

- Prześledźmy stos wywołań
- Na dole widoczne jest wywołanie funkcji main(), które rozpoczęło program
- Następnie wywołano funkcję printList() trzy razy, przy czym w ostatnim wywołaniu pojawiła się wartość 0xbaadf00d
- Znaleźliśmy błąd - niezainicjalizowany wskaźnik

# Wykorzystanie debuggera

## Debugowanie awarii

```
main.c x
13  int main()
14  {
15      Node* head;
16      head = malloc(sizeof(Node));
17      head->v = 10;
18      head->next = malloc(sizeof(Node));
19      head->next->v = 11;
20      head->next->next = NULL;
21      printList(head);
22      return 0;
23  }
24
25  void printList(const Node* n) {
26      if(n != NULL) {
27          printf("%d\n", n->v);
28          printList(n->next);
29      }
30  }
```



# Wykorzystanie debuggera

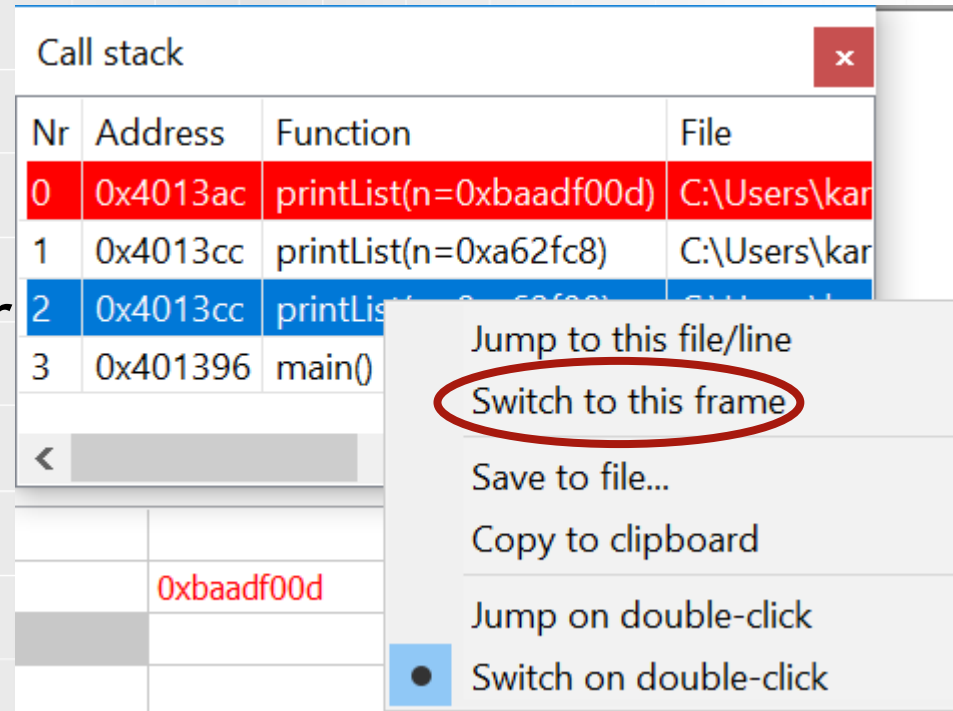
## Debugowanie awarii

- Informacje z innych ramek stosu możemy uzyskać przelączając widok (Swich to this frame)

# Wykorzystanie debuggera

## Debugowanie awarii

- Informacje z innych ramek można uzyskać przelączaając



Nr	Address	Function	File
0	0x4013ac	printList(n=0xbaadf00d)	C:\Users\kar...
1	0x4013cc	printList(n=0xa62fc8)	C:\Users\kar...
2	0x4013cc	printList(n=0xa62fc8)	C:\Users\kar...
3	0x401396	main()	

0xbaadf00d

- Jump to this file/line
- Switch to this frame
- Save to file...
- Copy to clipboard
- Jump on double-click
- Switch on double-click



# Wykorzystanie debuggera

## Zaglądnienie do zawieszzonego programu

- Jeśli program zatrzymał się z nieznanego powodu można wykorzystać debugger, aby włamać się do programu



# Wykorzystanie debuggera

## Zaglądnienie do zawieszzonego programu

```
main.c x
1  int main()
2  {
3      int i;
4      int silnia = 1;
5      for(i=1; i<10; i++){
6          silnia *= i;
7      }
8      int suma = 0;
9      for(i=0; i<10; i++){
10         suma += i;
11     }
12     int silnia_bez_dwojki = 1;
13     for(i=1; i<10; i++){
14         if( i==2 ){
15             continue;
16         }
17         silnia_bez_dwojki *= i;
18     }
19     int suma_bez_dwojki = 0;
20     for(i=0; i<10; i++){
21         if( i=2 ){
22             continue;
23         }
24         suma_bez_dwojki += i;
25     }
26     return 0;
27 }
```



# Wykorzystanie debuggera

## Zaglądnienie do zawieszzonego programu

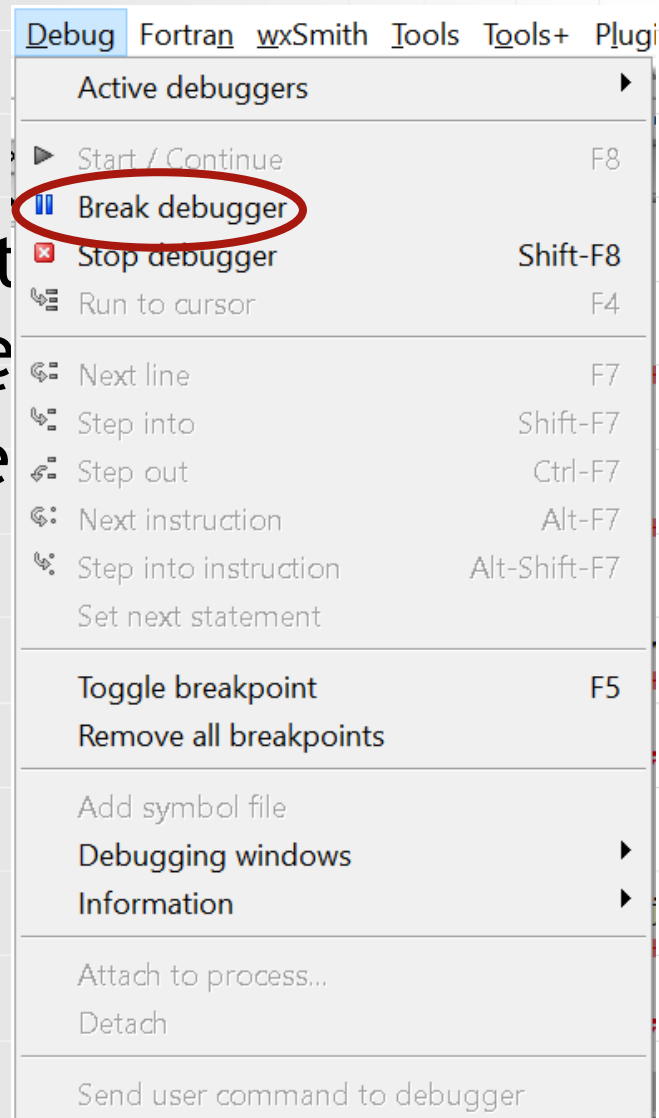
- Po uruchamieniu programu w trybie debugowania mamy możliwość włamania się do niego
- Menu Debug >> Break debugger



# Wykorzystanie debuggera

## Zagląkanie do zawieszzonego programu

- Po uruchamieniu programu w t...  
mamy możliwość włamania się
- Menu Debug >> Break debugge





# Wykorzystanie debuggera

## Zaglądnienie do zawieszzonego programu

- Po uruchamieniu programu w trybie debugowania mamy możliwość włamania się do niego
- Menu Debug >> Break debugger
- Po zatrzymaniu programu sprawdzamy stos wywołań

# Wykorzystanie debuggera

## Zaglądnienie do zawieszzonego programu

- Po uruchomieniu programu w trybie debugowania mamy r
- Menu D
- Po zatrzymaniu programu wywoła

Call stack x

Nr	Address	Function	File
0	0x7797b801	?? ()	
1	0x779b4cb9	?? ()	
2	0x758b8484	KERNEL32!BaseThreadInitThunk()	C:\WINDOWS\SysWOW64\kernel32
3	0x77972ec0	?? ()	
4	0x77972e90	?? ()	
5		?? ()	

< >



# Wykorzystanie debuggera

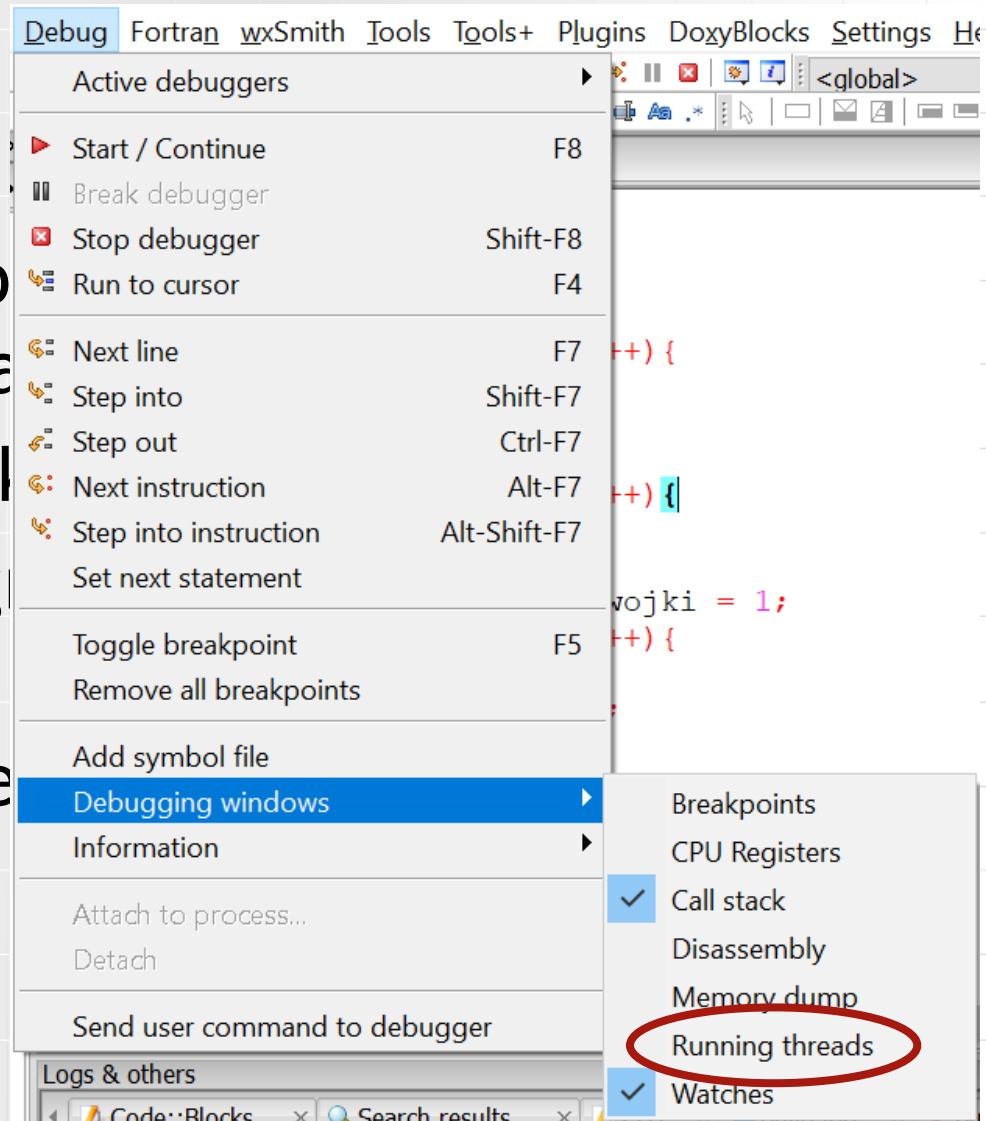
## Zagląwanie do zawieszzonego programu

- Po uruchamieniu programu w trybie debugowania mamy możliwość włamania się do niego
- Menu Debug >> Break debugger
- Po zatrzymaniu programu sprawdzamy stos wywołań
- Aby przejść do naszego programu musimy zmienić wątek

# Wykorzystanie debuggera

## Zagładanie do zawieszzonego programu

- Po uruchamieniu programu mamy możliwość włączyć debuggery
- Menu Debug >> Breakpoints
- Po zatrzymaniu programu wywołać breakpoint
- Aby przejść do naszego programu zmienić wątek





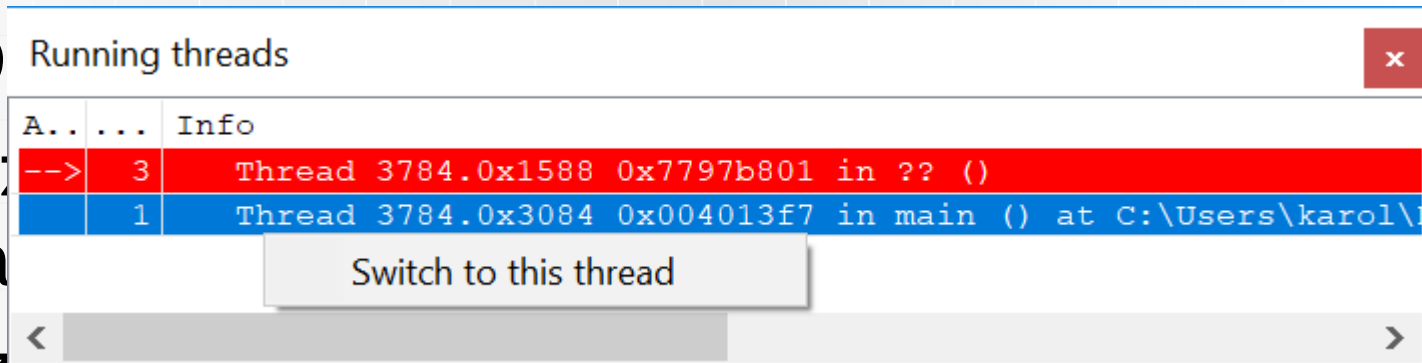
# Wykorzystanie debuggera

## Zaglądnienie do zawieszzonego programu

- Po uruchamieniu programu w trybie debugowania mamy możliwość włamania się do niego

- Menu D

- Po zatrzymaniu programu wywoła



A...	...	Info
-->	3	Thread 3784.0x1588 0x7797b801 in ?? ()
	1	Thread 3784.0x3084 0x004013f7 in main () at C:\Users\karol\

Switch to this thread

- Aby przejść do naszego programu musimy zmienić wątek



# Wykorzystanie debuggera

## Zaglądnienie do zawieszzonego programu

- Przechodząc przez kolejne wiersze, obserwujemy, że program krąży między liniami 20 oraz 21
- Zmienna `i` przyjmuje tylko dwie wartości 2 oraz 3



# Wykorzystanie debuggera

## Zagląkanie do zawieszzonego programu

- Przechodząc przez kolejne wiersze,

The screenshot shows a debugger window with a C program named 'main.c'. The code is as follows:

```
19 int suma_bez_dwojki = 0;
20 for(i=0; i<10; i++){
21     if( i=2 ){
22         continue;
23     }
24     suma_bez_dwojki += i;
25 }
26 return 0;
27 }
28
29
30
```

The 'Watches (new)' panel is open, showing the following data:

Function arguments	
i	3
silnia	0
suma	45
silnia_bez_dwojki	1
suma_bez_dwojki	0

Below the function arguments, there is a section for 'Locals' which is currently empty.



# Wykorzystanie debuggera

## Zaglądnienie do zawieszzonego programu

- Przechodząc przez kolejne wiersze, obserwujemy, że program krąży między liniami 20 oraz 21
- Zmienna `i` przyjmuje tylko dwie wartości 2 oraz 3
- Możemy zauważyć, że pętla w działaniu programu wynikała z użycia operatora przypisania w miejsce operatora porównania

# Wykorzystanie debuggera

## Zaglądnienie do zawieszzonego programu

```
main.c x
1  int main()
2  {
3      int i;
4      int silnia = 1;
5      for(i=1; i<10; i++){
6          silnia *= i;
7      }
8      int suma = 0;
9      for(i=0; i<10; i++){
10         suma += i;
11     }
12     int silnia_bez_dwojki = 1;
13     for(i=1; i<10; i++){
14         if( i==2 ){
15             continue;
16         }
17         silnia_bez_dwojki *= i;
18     }
19     int suma_bez_dwojki = 0;
20     for(i=0; i<10; i++){
21         if( i==2 ){
22             continue;
23         }
24         suma_bez_dwojki += i;
25     }
26     return 0;
27 }
```



# Podsumowanie

- Biblioteka GSL
- Wykorzystanie debuggera