

Sprawozdanie z Języków i Metoda Programowania

Laboratorium z dnia 04.11.2020

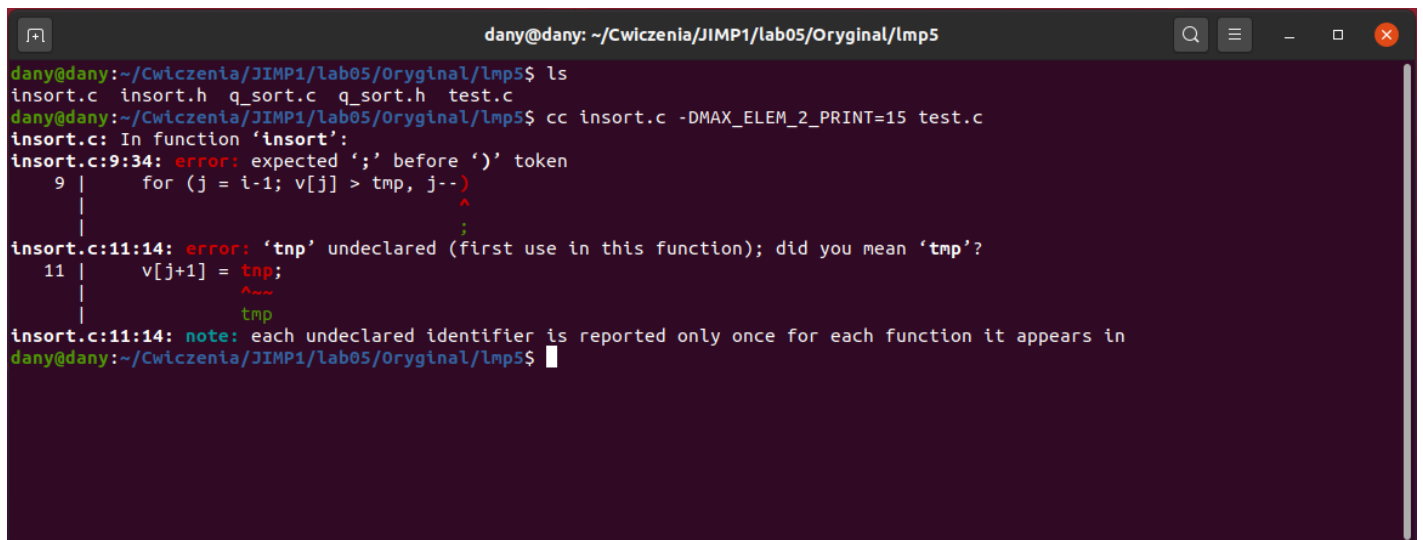
Daniel Ślusarczyk, gr. 3, nr albumu 311511

Sortowanie przez wstawianie:

Pliki z odpowiednimi plikami z laboratorium zostały pobrane i rozpakowane pod ścieżką:

~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5

Pierwsza próba kompilacji funkcji zawartej w pliku insort.c dołączanej do pliku test.c spowodowały komunikat kompilatora dotyczące użycia nieprawidłowej składni:



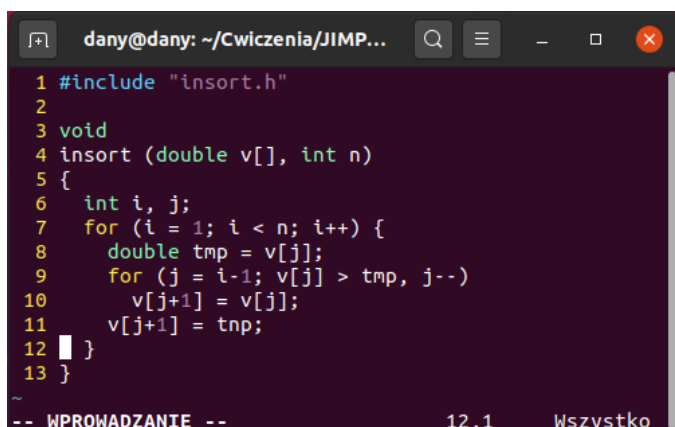
```
dany@dany: ~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ls
insort.c  insort.h  q_sort.c  q_sort.h  test.c
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ cc insort.c -DMAX_ELEM_2_PRINT=15 test.c
insort.c: In function 'insort':
insort.c:9:34: error: expected ';' before ')' token
    9 |         for (j = i-1; v[j] > tmp, j--)
      |                                   ^
insort.c:11:14: error: 'tnp' undeclared (first use in this function); did you mean 'tmp'?
    11 |         v[j+1] = tnp;
      |                  ^~~~
insort.c:11:14: note: each undeclared identifier is reported only once for each function it appears in
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$
```

Na podstawie tej informacji można stwierdzić, że funkcja posiada błędy składniowe w 9 i 11 linii kodu.

W pierwszym przypadku błąd polega na nieprawidłowym rozdzieleniu części warunku sterującego pętlą for od części przyrostu za pomocą przecinka („ , ”) zamiast średnika („ ; ”).

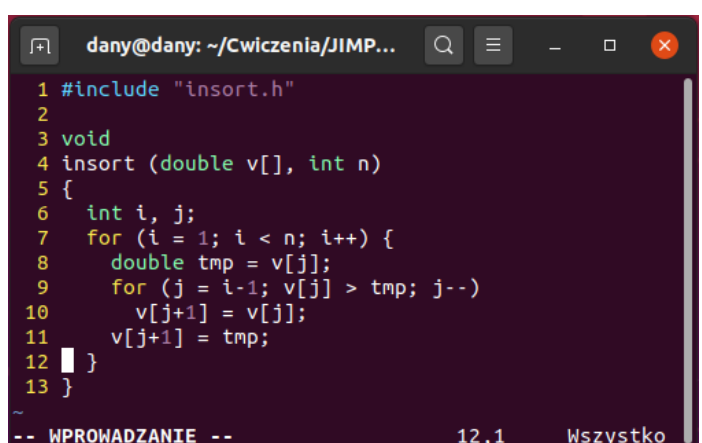
Błąd w linii 11 jest spowodowany odwołaniem do zmiennej „tnp”, która nie została zadeklarowana.

Prawidłową nazwą zmiennej jest „tmp”.



```
1 #include "insort.h"
2
3 void
4 insort (double v[], int n)
5 {
6     int i, j;
7     for (i = 1; i < n; i++) {
8         double tmp = v[j];
9         for (j = i-1; v[j] > tmp, j--)
10             v[j+1] = v[j];
11         v[j+1] = tnp;
12     }
13 }
```

Kod funkcji „insort” przed poprawieniem błędów



```
1 #include "insort.h"
2
3 void
4 insort (double v[], int n)
5 {
6     int i, j;
7     for (i = 1; i < n; i++) {
8         double tmp = v[j];
9         for (j = i-1; v[j] > tmp; j--)
10             v[j+1] = v[j];
11         v[j+1] = tmp;
12     }
13 }
```

Kod funkcji „insort” po poprawieniu błędów

Poprawienie błędów składniowych pozwala na skompilowanie programu, ale jego uruchomienie z wybranym parametrem powoduje błąd naruszanie ochrony pamięci, który jest prawdopodobnie spowodowany odwołaniem się do elementu spoza zainicjonowanych zmiennych.

```
dany@dany: ~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ cc insort.c -DMAX_ELEM_2_PRINT=15 test.c
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 5
Wygenerowany wektor: [ -1.73536 2.8248 3.00696 2.22029 -4.80354 ]
Naruszenie ochrony pamięci (zrzut pamięci)
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$
```

Narzędziem, którym się posługuje do zdiagnozowania błędu jest debugger. Pierwszym krokiem jest kompilacja programu z opcją powodującą wygenerowanie kodu z informacjami wykorzystywanymi przez debugger i uruchomienie. Breakpoint zostaje ustawiony na funkcję main w celu zatrzymywania programu po dojściu do instrukcji main.

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) break main
Breakpoint 1 at 0x1396: file test.c, line 18.
(gdb) run 6
Starting program: /home/dany/Cwiczenia/JIMP1/lab05/Oryginal/lmp5/a.out 6

Breakpoint 1, main (argc=21845, argv=0x0) at test.c:18
18      main( int argc, char **argv ) {
(gdb) n
19          int n= argc > 1 ? atoi( argv[1] ) : 1000;
(gdb) n
22          double *v= malloc( n * sizeof *v );
(gdb) n
24          srand( argc > 2 ? atoi(argv[2]) : time(NULL) );
(gdb) n
26          for( i= 0; i < n; i++ )
(gdb) n 20
Wygenerowany wektor: [ -4.71924 2.83604 -1.02022 -3.27309 5.48776 -5.2306 ]

Program received signal SIGSEGV, Segmentation fault.
insort (v=0x5555555592a0, n=6) at insort.c:8
8          double tmp = v[j];
(gdb) p j
$1 = 21845
(gdb) p tmp
$2 = 1.2731974748756028e-313
(gdb)
```

Analiza wartości przyjmowanych przez poszczególne linie kodu prowadzi do wniosku, że w linii 8 program próbuje odwołać się do wartości, które nie mają sensu z perspektywy jego działania. Element 21845 tablicy v nie powinien być używany przez funkcję w tym przypadku.

```
#include "insort.h"

void
insort (double v[], int n)
{
    int i, j;
    for (i = 1; i < n; i++) {
        double tmp = v[i];
        for (j = i-1; v[j] > tmp; j--)
            v[j+1] = v[j];
        v[j+1] = tmp;
    }
}
```

Zmienna tmp powinna przyjmować wartość v[i], ponieważ to zmienna „i” służy do interakcji po danych wektorze.

Ponowna kompilacja i uruchomienia programu z podaniem zmiennej powoduje pojawienie się komunikatu o naruszeniu pamięci

```
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ vi insort.c
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ cc insort.c -DMAX_ELEM_2_PRINT=16 test.c
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 5
Wygenerowany wektor: [ -1.04005 1.93413 4.04008 1.83288 -4.18251 ]
Naruszenie ochrony pamięci (zrzut pamięci)
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$
```

W celu weryfikacji błędu ponownie uruchamiam debugger ustawiając breakpoint na funkcję main i analizuje poszczególne wartości przyjmowane przez zmienne z szczególną uwagą na zmienne „i” oraz „j”, które mogą powodować odwołanie się do elementu tablicy, który nie został zadeklarowany.

```

(gdb) run 6
Starting program: /home/dany/Cwiczenia/JIMP1/lab05/Oryginal/lmp5/a.out 6

Breakpoint 1, main (argc=21845, argv=0x0) at test.c:18
18      main( int argc, char **argv ) {
(gdb) n 18
30          printf( v, n );
(gdb) n
Wygenerowany wektor: [ 4.22734 1.786 -3.08196 1.80798 4.88145 3.52931 ]
32          insort( v, n );
(gdb) s
insort (v=0x5555555592a0, n=6) at insort.c:5
5      {
(gdb) n 3
9          for (j = i-1; v[j] > tmp; j--)
(gdb) p tmp
$1 = 1.785999226284213
(gdb) n
10          v[j+1] = v[j];
(gdb) n
9          for (j = i-1; v[j] > tmp; j--)
(gdb) p j
$2 = 0
(gdb) n
11          v[j+1] = tmp;
(gdb) p j
$3 = -1
(gdb)

```

Analiza wartości przyjmowanych przez zmienne pozwala stwierdzić, że gdy warunek pętli for zostanie spełniony to pętla powinna się wykonywać do momentu „przesunięcia” wszystkich wartości większych od zmiennej tmp, lub przesunięcia wszystkich elementów należących do tablicy. Z powodu braku drugiego warunku pętla może się odwołać do elementu $v[j] = v[-1]$, który nie należy do zadeklarowanej tablicy.

```

dany@dany: ~/Cwiczenia/JIMP1/la...
#include "insort.h"

void
insort (double v[], int n)
{
    int i, j;
    for (i = 1; i < n; i++) {
        double tmp = v[i];
        for (j = i-1; v[j] > tmp && j >= 0 ; j--)
            v[j+1] = v[j];
        v[j+1] = tmp;
    }
}
~
~
~
~
12,1      Wszystko

```

Dopisanie drugiego warunku ($j \geq 0$) zatrzymuje funkcję w sytuacji gdy wszystkie odpowiednie elementy zostaną przesunięte

Zmiany wprowadzone w obrębie kodu prowadzą do prawidłowego działania funkcji insort.

```
dany@dany: ~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ cc insort.c -DMAX_ELEM_2_PRINT=16 test.c
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 6
Wygenerowany wektor: [ -5.32409 5.10377 -4.20466 1.58686 0.857662 5.57842 ]
Posortowany wektor: [ -5.32409 -4.20466 0.857662 1.58686 5.10377 5.57842 ]
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 7
Wygenerowany wektor: [ -1.19424 -5.49039 2.43187 -1.21633 0.914155 3.62634 4.09269 ]
Posortowany wektor: [ -5.49039 -1.21633 -1.19424 0.914155 2.43187 3.62634 4.09269 ]
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 8
Wygenerowany wektor: [ -4.89049 -5.5188 2.5787 7.97442 -4.56334 5.31485 -6.74624 2.43234 ]
Posortowany wektor: [ -6.74624 -5.5188 -4.89049 -4.56334 2.43234 2.5787 5.31485 7.97442 ]
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 9
Wygenerowany wektor: [ -3.53743 -3.33468 -5.00398 4.14111 -7.39638 -7.53377 -4.94066 -6.43832 -0.332118 ]
Posortowany wektor: [ -7.53377 -7.39638 -6.43832 -5.00398 -4.94066 -3.53743 -3.33468 -0.332118 4.14111 ]
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 10
Wygenerowany wektor: [ 3.21087 -0.274031 4.90641 0.205606 -8.60045 -2.52851 -7.67554 -9.35373 -9.2214 -6.88176 ]
Posortowany wektor: [ -9.35373 -9.2214 -8.60045 -7.67554 -6.88176 -2.52851 -0.274031 0.205606 3.21087 4.90641 ]
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$
```

Sortowanie szybkie:

Pierwsza próba kompilacji funkcji zawartej w pliku q_sort.c dołączanej do pliku qtest.c (plik analogiczny do test.c, ale odwołujący się do sprawdzanej funkcji zamiast funkcji insort) spowodowały komunikat dotyczący błędnego sortowania funkcji.

```
dany@dany: ~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ cc q_sort.c -DMAX_ELEM_2_PRINT=16 qtest.c
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 5
Wygenerowany wektor: [ 3.81591 -2.15758 1.76542 0.123447 2.59762 ]
Posortowany wektor: [ -2.15758 1.76542 0.123447 2.59762 3.81591 ]
Zły algorytm sortowania: v[1]==1.76542 > v[2]==0.123447
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 5
Wygenerowany wektor: [ 4.62242 4.14538 1.998 2.09453 0.994139 ]
Posortowany wektor: [ 4.14538 0.994139 2.09453 1.998 4.62242 ]
Zły algorytm sortowania: v[0]==4.14538 > v[1]==0.994139
Zły algorytm sortowania: v[2]==2.09453 > v[3]==1.998
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 5
Wygenerowany wektor: [ -1.70437 -4.17516 -2.78802 -4.9814 0.773253 ]
Posortowany wektor: [ -4.17516 -4.9814 -2.78802 0.773253 -1.70437 ]
Zły algorytm sortowania: v[0]==-4.17516 > v[1]==-4.9814
Zły algorytm sortowania: v[3]==0.773253 > v[4]==-1.70437
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$
```

Pozwala to stwierdzić, że funkcja posiada jedynie błędy merytoryczne, które umożliwiają poprawną kompilację. W celu śledzenia wartości przyjmowanych przez poszczególne zmienne otwieram debugger z program wykonywalnym z zaimplementowaną funkcją q_sort.

Analiza dla kilku przypadków pozwala zauważyć, że dla wektora, który spełnia przypadek ,że wszystkie elementy są większe od pierwszego (v[0]) to funkcja nie ma możliwości odwołania się do zerowego elementu przez linijkę kodu z „f++;”

```

30         printf( v, n );
(gdb) n
Wygenerowany wektor: [ -3.88516 2.51643 3.89967 2.59052 ]

```

```

21         tmp = v[s];
(gdb) n
22         v[s] = v[f];
(gdb) n
23         v[f] = tmp;
(gdb) n
24         return f;
(gdb) p v[0]
$1 = 2.5164327893948335
(gdb) p v[1]
$2 = -3.8851621504338283
(gdb) p v[2]
$3 = 3.8996722045818681
(gdb) p v[3]
$4 = 2.5905219850086243
(gdb) p f
$5 = 1
(gdb) p l
$6 = 1
(gdb)

```

Dla danego wektora funkcja powinna zwrócić zero bez zamiany żadnego elementu, ale numeracja f zaczyna się od 1.

Usunięcie niewłaściwej linijki kodu pozwala doprowadzić funkcję do prawidłowego sortowania.

```

#include "q_sort.h"

int divide( double v[], int f, int l ) {
    double tmp; // zmienna tymczasowa do zmiany elementów
    int s= f; // dzielimy ze względu na pierwszy element
    while( f < l ) {
        while( f < l && v[f] < v[s] )
            f++;
        while( f < l && v[l] >= v[s] )
            l--;
        if( f < l ) {
            tmp= v[f];
            v[f]= v[l];
            v[l]= tmp;
        }
        // zamieniamy element v[s] z v[f]
        // aby v[s] znalazł się pomiędzy
        // mniejszymi i nie-większymi od niego
        tmp = v[s];
        v[s] = v[f];
        v[f] = tmp;
        return f;
    }
}

void qsort_rec( double v[], int first, int last ) {
    // wygodniej jest zapisać tę funkcję rekurencyjną
    // posługując się indeksem początku i końca wektora
    if( first < last ) {
        int m= divide( v, first, last );
        qsort_rec( v, first, m-1 );
        qsort_rec( v, m+1, last );
    }
}

void q_sort( double v[], int n ) { // to jest tylko opakowanie, aby wywołanie
    // wyglądało tak samo, jak insort
    qsort_rec( v, 0, n-1 );
}

```

```

dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ vi q_sort.c
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ cc q_sort.c -DMAX_ELEM_2_PRINT=16 qtest.c
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 5
Wygenerowany wektor: [ 2.06769 0.7021 3.17103 2.4929 0.463539 ]
Posortowany wektor: [ 0.463539 0.7021 2.06769 2.4929 3.17103 ]
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 6
Wygenerowany wektor: [ 0.0813306 1.68004 3.92154 4.30443 -3.32642 -4.72036 ]
Posortowany wektor: [ -4.72036 -3.32642 0.0813306 1.68004 3.92154 4.30443 ]
dany@dany:~/Cwiczenia/JIMP1/lab05/Oryginal/lmp5$ ./a.out 7
Wygenerowany wektor: [ 1.02121 -3.41364 -2.18936 -6.44942 0.627944 4.24929 -1.98017 ]
Posortowany wektor: [ -6.44942 -3.41364 -2.18936 -1.98017 0.627944 1.02121 4.24929 ]

```