Język ANSI C

część 16

struktury rekurencyjne i ich zastosowania
drzewa binarne, algorytmy rekurencyjne dla drzew binarnych

Jarosław Gramacki
Instytut Informatyki i Elektroniki

- rekurencja
  - wiele czynności wykonywanych na drzewach łatwo zaprogramować w postaci rekurencyjnej
  - poznanie istoty rekurencji jest więc bardzo ważne
  - rekurencja to zdolność funkcji do wywoływania samej siebie

przykład 1

```
1       #include <stdio.h>
2       int czytaj( int* );
3
4       int main(int argc, char *argv[])
5       {
6               int c = -1;
7               printf("Podaj liczbe: ");
8               czytaj (&c);
9               printf("(%d) \n", c);
10              return 0;
11      }
12
13      int czytaj (int *p)
14      {
15              int ok = 0;
16              char buf[3];
17              static int ile = 1;             // zm. statyczna
18
19              fgets(buf, 4, stdin);
20              ok = sscanf(buf, "%d", p);      // 1 gdy przekształcenie do %d możliwe
21              if ( !ok ) {
22                      printf ("%d ?: ", ile++);
23                      czytaj (p);             // bez &
24              }
25              return 1;                       // na potrzeby analizy w gdb
26      }
```

- rekurencja
  - Uwagi ogólne dotyczące rekurencji:
  - zarówno iteracja jak i rekurencja wymagają powtarzania. Iteracja jawnie korzysta z powtarzania; rekurencja osiąga powtarzanie przez wielokrotne wywołania funkcji. Stąd iteracja i rekurencja są w pewnym sensie wymienne
  - zarówno iteracja jak i rekurencja wymagają testu zakończenia. Rekurencja kończy sie gdy zostanie rozpoznany tzw. przypadek podstawowy
  - rekurencja może się (tak jak iteracja) zapętlić. Nastąpi to gdy krok rekurencji nie zredukuje problemu za każdym razem w sposób, który prowadzi do przypadku podstawowego
  - rekurencja wielokrotnie wykorzystuje i w konsekwencji przeciąża mechanizm wywołania funkcji. Może to mieć duży wpływ na czas procesora oraz zużytą pamięć

```
// recursive definition of function factorial
// def: n! = n * (n-1)!
unsigned long factorial( unsigned long number )
{
   // base case
   if ( number <= 1 )
      return 1;

   // recursive step
   else
      return number * factorial( number - 1 );
}
```

```
// proces wywołań rekurencyjnych
 4!
 4 * 3!
   3 * 2!
     2 * 1!
         1!

// wartości zwracane z każdego
// wywołania rekurencyjnego
 4!                  // wartość końcowa 24
 4 * 3!              // zwrócone 24
   3 * 2!            // zwrócone 6
     2 * 1!          // zwrócone 2
         1!          // zwrócone 1
```


- rekurencja
  - analiza stosu programu (polecenie **backtrace** debugera **gdb**)

przykład 1

```
(gdb) break 23 if ile == 3
Breakpoint 1 at 0x80484c3: file rekur_1.c, line 23.

(gdb) run
Starting program: /home/users/jarek/a.out
Podaj liczbe: A
1 ?: B

Breakpoint 1, czytaj (p=0xbffffa44) at rekur_1.c:23
23                      czytaj (p);

(gdb) s
czytaj (p=0xbffffa44) at rekur_1.c:15
15              int ok = 0;

(gdb) s
19              fgets(buf, 4, stdin);

(gdb) s
2 ?: 5
20              ok = sscanf(buf, "%d", p);

(gdb) s
21              if ( !ok ) {

(gdb) backtrace
#0  czytaj (p=0xbffffa44) at rekur_1.c:21
#1  0x080484ce in czytaj (p=0xbffffa44) at rekur_1.c:23
#2  0x080484ce in czytaj (p=0xbffffa44) at rekur_1.c:23
#3  0x08048442 in main (argc=1, argv=0xbffffaa4) at rekur_1.c:8
```

Debugging with GDB
The GNU Source-Level Debugger
Seventh Edition, for GDB version 4.18
February 1999
Richard M. Stallman and Roland H. Pesch

http://www.sunsite.ualberta.ca/Documentation/Gnu/gdb-4.18/html_chapter/gdb_toc.html

```
...
# 6 Examining the Stack
    * 6.1 Stack frames
    * 6.2 Backtraces
    * 6.3 Selecting a frame
    * 6.4 Information about a frame
...
```

- rekurencja
  - analiza stosu programu (polecenie backtrace debugera gdb)

przykład 1

```
(gdb) backtrace full

#0  czytaj (p=0xbffffa44) at rekur_1.c:21
        ok = 1
        buf = "5\n"
        ile = 3
#1  0x080484ce in czytaj (p=0xbffffa44) at rekur_1.c:23
        ok = 0
        buf = "B\n"
        ile = 3
#2  0x080484ce in czytaj (p=0xbffffa44) at rekur_1.c:23
        ok = 0
        buf = "A\n"
        ile = 3
#3  0x08048442 in main (argc=1, argv=0xbffffaa4) at rekur_1.c:8
        c = 5
```

kolejne wcielenia parametru funkcji (przekazywanego przez wartość) oraz zmiennych lokalnych odkładanych w pamięci na stosie

- rekurencja
  - analiza stosu programu (polecenie backtrace debugera gdb)

przykład 1

```
21              if ( !ok ) {
(gdb) s
25              return 1;
(gdb) s
26      }
(gdb) s
czytaj (p=0xbffffa44) at rekur_1.c:25
25              return 1;
(gdb) s
26      }
(gdb) s
czytaj (p=0xbffffa44) at rekur_1.c:25
25              return 1;

(gdb) backtrace
#0  czytaj (p=0xbffffa44) at rekur_1.c:25
#1  0x08048442 in main (argc=1, argv=0xbffffaa4) at rekur_1.c:8

(gdb) s
26      }

(gdb) s
main (argc=1, argv=0xbffffaa4) at rekur_1.c:9
9               printf("(%d) \n", c);

(gdb) backtrace
#0  main (argc=1, argv=0xbffffaa4) at rekur_1.c:9
(gdb)
```

- rekurencja
  - analiza stosu programu (polecenie backtrace debugera gdb)

przykład 1

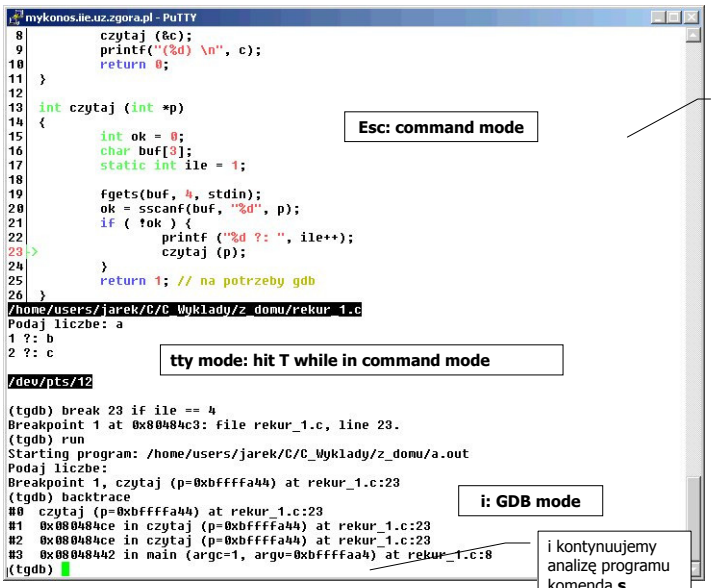
```
inne przydatne polecenia gdb operujące na stosie:

(gdb) backtrace
#0  czytaj (p=0xbffffa44) at rekur_1.c:21
#1  0x080484ce in czytaj (p=0xbffffa44) at rekur_1.c:23
#2  0x080484ce in czytaj (p=0xbffffa44) at rekur_1.c:23
#3  0x08048442 in main (argc=1, argv=0xbffffaa4) at rekur_1.c:8

(gdb) frame 2                          // ustawiamy się w wybranym miejscu stosu
#2  0x080484ce in czytaj (p=0xbffffa44) at rekur_1.c:23
23                     czytaj (p);

(gdb) return                           pop selected stack frame without executing [setting return value]
Make czytaj return now? (y or n) y
#0  main (argc=1, argv=0xbffffaa4) at rekur_1.c:9
9               printf("(%d) \n", c);

(gdb) backtrace                        // "przeskoczyliśmy" kawałek programu
#0  main (argc=1, argv=0xbffffaa4) at rekur_1.c:9
```


- rekurencja
  - analiza stosu programu (polecenie backtrace debugera gdb)

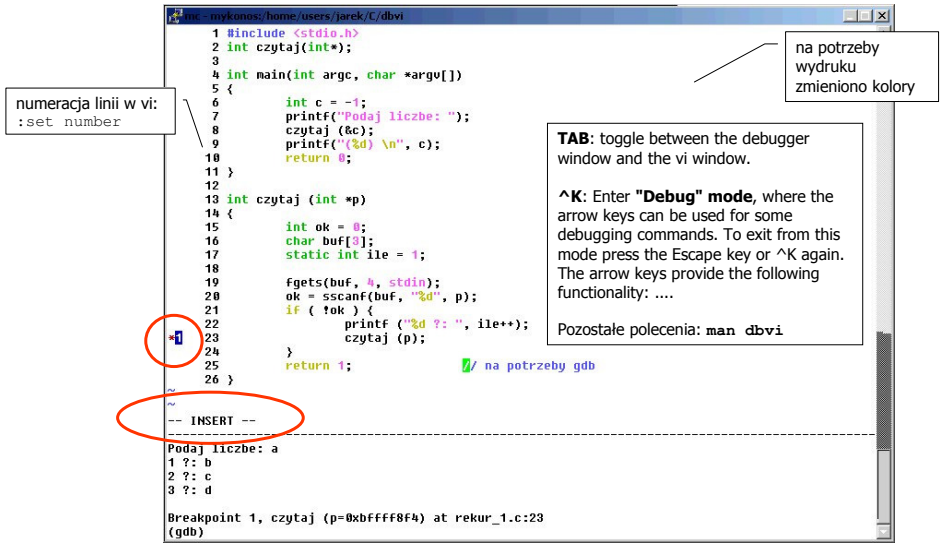
przykład 1

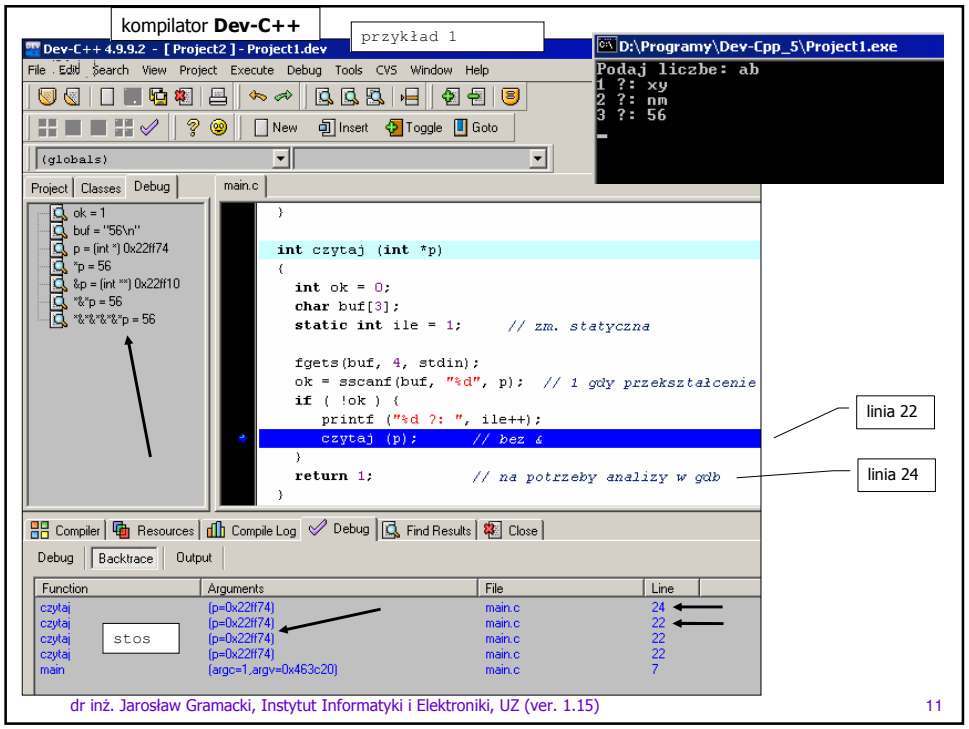
```
inne przydatne polecenia gdb operujące na stosie:

(gdb) backtrace
#0  czytaj (p=0xbffffa44) at rekur_1.c:21
#1  0x080484ce in czytaj (p=0xbffffa44) at rekur_1.c:23
#2  0x080484ce in czytaj (p=0xbffffa44) at rekur_1.c:23
#3  0x08048442 in main (argc=1, argv=0xbffffaa4) at rekur_1.c:8

(gdb) frame 1                          // ustawiamy się w wybranym miejscu stosu
#1  0x080484ce in czytaj (p=0xbffffa44) at rekur_1.c:23
23                     czytaj (p);

(gdb) finish                           run until selected stack frame returns
Run till exit from #1  0x080484ce in czytaj (p=0xbffffa44) at rekur_1.c:23

3 ?: 5
czytaj (p=0xbffffa44) at rekur_1.c:25
25              return 1;     // na potrzeby analizy w gdb
Value returned is $5 = 1

(gdb) backtrace
#0  czytaj (p=0xbffffa44) at rekur_1.c:25
#1  0x08048442 in main (argc=1, argv=0xbffffaa4) at rekur_1.c:8
```

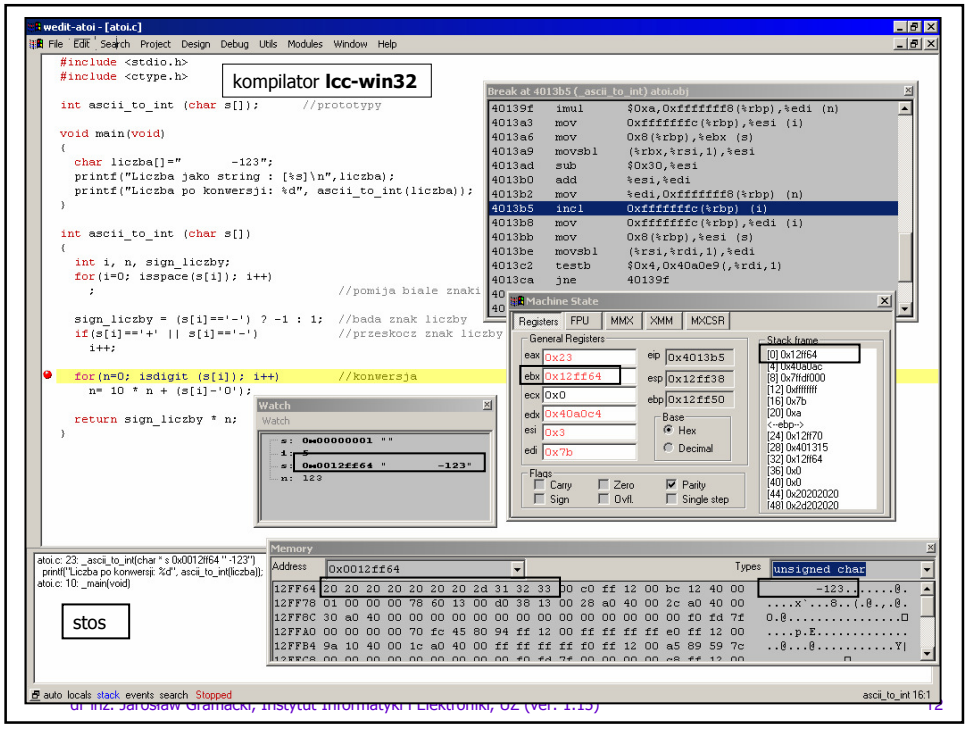
- rekurencja
  - analiza stosu programu ("graficzna" nakładka **cgdb**, `http://cgdb.sourceforge.net/`)



- rekurencja
  - analiza stosu programu ("graficzna" nakładka **dbvi**, `http://dbvi.sourceforge.net/`)
  - debuger **gdb** skojarzony z edytorem **vi**




kompilator Dev-C++
przykład 1
Dev-C++ 4.9.9.2 - [Project2] - Project1.dev
D:\Programy\Dev-Cpp_5\Project1.exe
Podaj liczbe: ab
1 ?: xy
2 ?: nm
3 ?: 56
int czytaj (int *p)
{
int ok = 0;
char buf[3];
static int ile = 1; // zm. statyczna
fgets(buf, 4, stdin);
ok = sscanf(buf, "%d", p); // 1 gdy przekształcenie
if (!ok) {
printf ("%d ?: ", ile++);
czytaj (p); // bez &
}
return 1; // na potrzeby analizy w gdb
}
linia 22
linia 24
stos
Function
Arguments
File
Line
czytaj [p=0x22ff74] main.c 24
czytaj [p=0x22ff74] main.c 22
czytaj [p=0x22ff74] main.c 22
czytaj [p=0x22ff74] main.c 22
main [argc=1,argv=0x463c20] main.c 7


wedit-atoi - [atoi.c]
kompilator lcc-win32
#include <stdio.h>
#include <ctype.h>
int ascii_to_int (char s[]); //prototypy
void main(void)
{
char liczba[]=" -123";
printf("Liczba jako string : [%s]\n",liczba);
printf("Liczba po konwersji: %d", ascii_to_int(liczba));
}
int ascii_to_int (char s[])
{
int i, n, sign_liczby;
for(i=0; isspace(s[i]); i++)
;
sign_liczby = (s[i]=='-') ? -1 : 1;
if(s[i]=='+' || s[i]=='-')
i++;
for(n=0; isdigit (s[i]); i++)
n= 10 * n + (s[i]-'0');
return sign_liczby * n;
}
Machine State
Registers
FPU
MMX
XMM
MXCSR
Memory
Address
Types
unsigned char
stos

- rekurencja

przykład 2

```
1       #include <stdio.h>
2       int czytaj( void );
3
4       int main(int argc, char *argv[])
5       {
6               int c = -1;
7               printf("Podaj liczbe: ");
8               c = czytaj ();
9               printf("(%d) \n", c);
10              return 0;
11      }
12
13      int czytaj (void)
14      {
15              int ok = 0, p;          // p = ?
16              char buf[3];
17              static int ile = 1;     // zm. statyczna
18
19              fgets(buf, 4, stdin);
20              ok = sscanf(buf, "%d", &p);
21              if ( !ok ) {
22                      printf ("%d ?:", ile++);
23                      p = czytaj ();
24              }
25              return p;
26      }
```


- rekurencja
  - analiza stosu programu (polecenie **backtrace** debugera **gdb**)

przykład 2

```
(gdb) break 23 if ile == 3
Breakpoint 1 at 0x80484c0: file rekur_2.c, line 23.

(gdb) run
Starting program: /home/users/jarek/a.out
Podaj liczbe: A
1 ?:B

Breakpoint 1, czytaj () at rekur_2.c:23
23                      p = czytaj ();

(gdb) s
czytaj () at rekur_2.c:15
15              int ok = 0, p;

(gdb) s
19              fgets(buf, 4, stdin);

(gdb) s
2 ?:5
20              ok = sscanf(buf, "%d", &p);

(gdb) s
21              if ( !ok ) {

(gdb) backtrace
#0  czytaj () at rekur_2.c:21
#1  0x080484c5 in czytaj () at rekur_2.c:23
#2  0x080484c5 in czytaj () at rekur_2.c:23
#3  0x0804843c in main (argc=1, argv=0xbffffaa4) at rekur_2.c:8
```

- rekurencja
  - analiza stosu programu (polecenie backtrace debugera gdb)

przykład 2

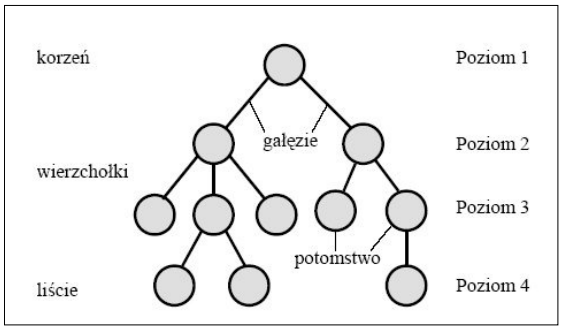
```
(gdb) backtrace full

#0  czytaj () at rekur_2.c:21
        ok = 1
        p = 5
        buf = "5\n"
        ile = 3
#1  0x080484c5 in czytaj () at rekur_2.c:23
        ok = 0
        p = 1075081952
        buf = "B\n"
        ile = 3
#2  0x080484c5 in czytaj () at rekur_2.c:23
        ok = 0
        p = 1075081952
        buf = "A\n"
        ile = 3
#3  0x0804843c in main (argc=1, argv=0xbffffaa4) at rekur_2.c:8
        c = -1
```


- rekurencja
  - analiza stosu programu (polecenie backtrace debugera gdb)

przykład 2

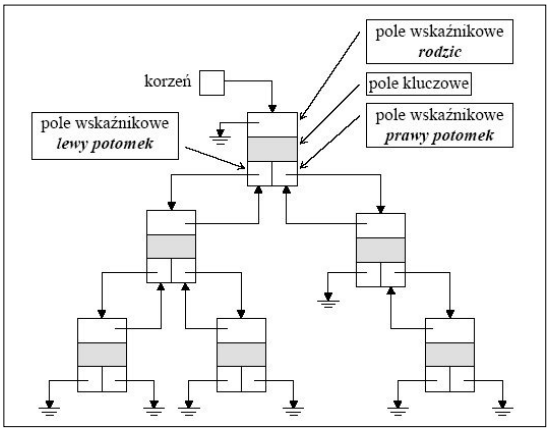
```
21              if ( !ok ) {

(gdb) s
25              return p;
(gdb) s
26      }
(gdb) s
25              return p;
(gdb) s
26      }
(gdb) s
25              return p;


(gdb) backtrace
#0  czytaj () at rekur_2.c:25
#1  0x0804843c in main (argc=1, argv=0xbffffaa4) at rekur_2.c:8

(gdb) s
26      }

(gdb) s
main (argc=1, argv=0xbffffaa4) at rekur_2.c:9
9               printf("(%d) \n", c);

(gdb) backtrace
#0  main (argc=1, argv=0xbffffaa4) at rekur_2.c:9
(gdb)
```

- drzewa binarne *
  - drzewo binarne (drzewo poszukiwań binarnych - ang. binary search tree BST) to drzewo, w którym rząd wyjściowy wierzchołków ograniczony jest do 2
  - podstawowe elementy drzewa



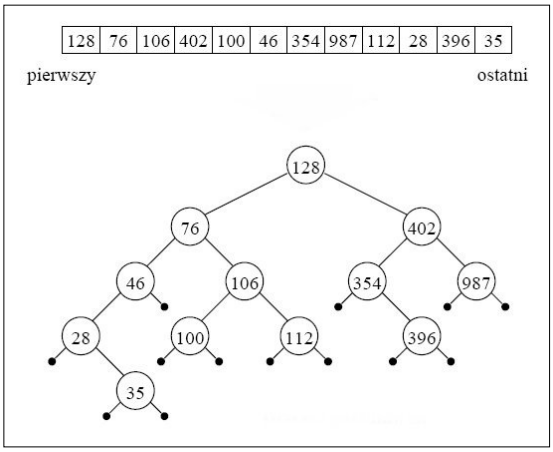
* na podstawie materiałów: dr inż. Jarosława A. Sikorskiego, http://informatyka.wit.edu.pl/

- drzewa binarne *
  - struktura danych opisująca drzewa
    (pola wskaźnikowego *rodzic* nie będziemy wykorzystywać w pryk ładach)



* na podstawie materiałów: dr inż. Jarosława A. Sikorskiego, http://informatyka.wit.edu.pl/

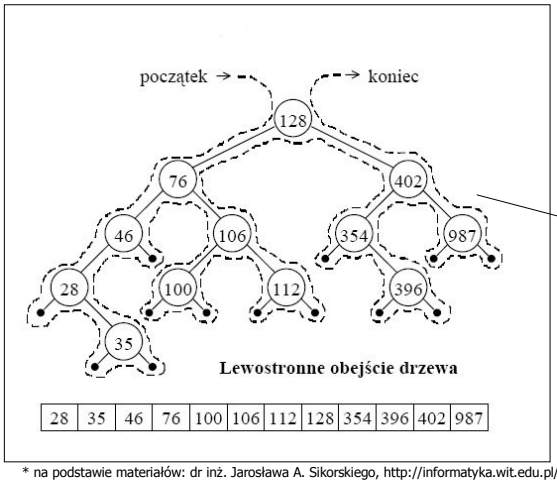
- drzewa binarne *
  - tworzenie drzewa binarnego. Dane napływają szeregowo
  - kolejny element wstawiamy po lewej lub prawej stronie w zależności od jego wartości

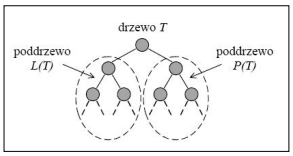


  - teraz bardzo szybko można odszukać element "w bazie danych". Sprawdzenie na przykład czy liczba 112 znajduje się w drzewie wymaga jedynie **3** kroków

* na podstawie materiałów: dr inż. Jarosława A. Sikorskiego, http://informatyka.wit.edu.pl/

- drzewa binarne *
  - rekurencyjne przeszukiwanie całego drzewa (odwiedzanie każdego węzła dokładnie raz)
  - przejdź drzewo w kolejności "najpierw w lewo" i wypisz elementy umieszczone w wierzchołkach przy ich powtórnych odwiedzinach
  - jest to algorytm przechodzenia drzewa w głąb tzw. *inorder (istnieją również algorytmy typu preorder, postorder)*





1. odszukanie elementu w drzewie (w odróżnieniu od przeszukania całego drzewa) nie wymaga rekurencji

2. Algorytm rekurencyjny będzie wręcz nienaturalny dla tego zastosowania

* na podstawie materiałów: dr inż. Jarosława A. Sikorskiego, http://informatyka.wit.edu.pl/

- drzewa binarne
  - struktura danych, tworzenie drzewa

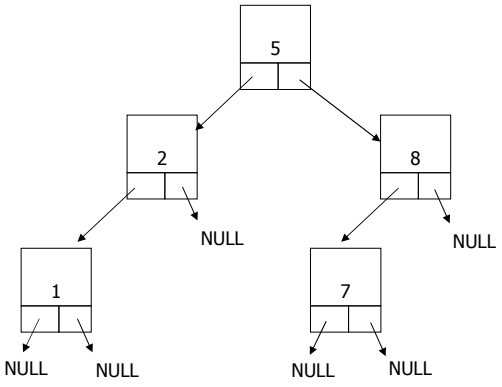
drzewo 1

```
struct node1 {
  int data;
  struct node *left;
  struct node *right;
};

typedef struct node1 *Ttree1;           // wskaźnik do elementu
```

```
int main( int argc, char *argv[] )
{
  int c;
  Ttree1 root = NULL;                   // korzeń

  while ((c = getchar()) != '\n')
    insert1( &root, c-'0' );            // uwaga na &

  if ( tmp = find1(root, 5) )
    printf("\n%c\n", tmp->data);

  printTree ( root );

  return 0;
}
```


- drzewa binarne
  - zasada budowy drzewa binarnego

```
5 8 2 7 1 ...
```



- drzewa binarne
  - tworzenie drzewa

drzewo 1

```
// elementy wstawiamy na dole (lewego lub prawego) poddrzewa

void insert1 (Ttree1 *root, int _data)            // wskaźnik do wskaźnika !!!
{                                                 // porównaj użyty typedef
  Ttree1 p, prev = NULL, new_node;

  new_node = (Ttree1) malloc(sizeof(struct node1));
  new_node->left = new_node->right = NULL;
  new_node->data = _data;

  // szukaj miejsca gdzie wstawić
  // zawsze zaczynamy od korzenia i idziemy w doł drzewa
  p = *root;
  while ( p )
  {
    prev = p;
    if (p->data > _data)
      p = p->left;
    else
      p = p->right;
  }

  if ( !*root )                                   // drzewo puste
    *root = new_node;
  else if (prev->data > _data)
    prev->left = new_node;
  else
    prev->right = new_node;
}
```


- drzewa binarne
  - szukanie elementu w drzewie

drzewo 1

```
// wersja iteracyjna, OK
Ttree1 find1 (Ttree1 root, int _data)
{
  while (root)
  {
    if (_data == root->data)
      return root;                    // znalazłem
    else if ( root->data > _data)
      root = root->left;
    else
      root = root->right;
  }
  return NULL;
}
```

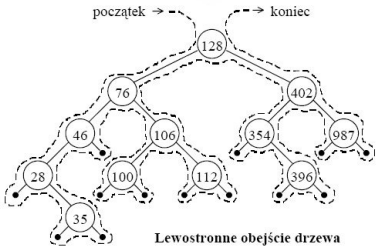
```
// wersja rekurencyjna, ???
Ttree1 find2 (Ttree1 root, int _data)
{
  if (root)
    if (_data == root->data)
      return root;                    // znalazłem
    else if ( root->data > _data)
      return find2 (root->left, _data);
    else
      return find2( root->right, _data);
  else
    return NULL;
}
```

- drzewa binarne
  - przeszukiwanie drzewa, wersja rekurencyjna

drzewo 1

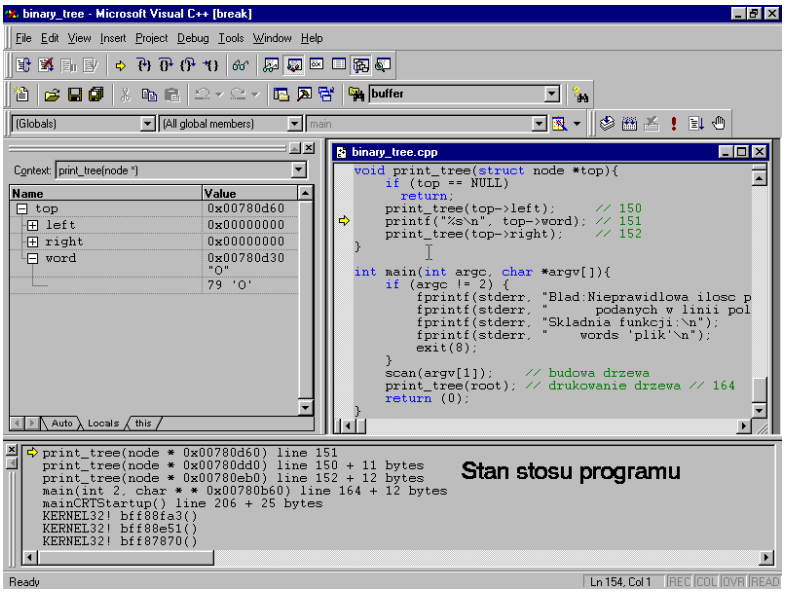
```
// procedura przechodzenia typu inorder
// na wydruku uzyskujemy elementy drzewa w porządku rosnącym

void printTree (Ttree1 root)
{
  if (root == NULL)
    return;
  printTree( root->left );
  printf("%d", root->data);
  printTree( root->right );
}
```

technika wycofywania się



1. przesuwamy się w głąb aż dojdziemy do liścia
2. cofamy się do wierzchołka, przez który już przeszliśmy i od niego realizujemy p. 1

- drzewa binarne
  - przeszukiwanie drzewa, wersja rekurencyjna, graficzny debugger VC++



- drzewa binarne
  - graficzny debugger **Dev-C++**
  - domyślnie korzysta z (przeniesionego pod Win) debuggera gdb (czyli tryb tekstowy !)
  - można "skojarzyć" go z graficznym debuggerem
    `http://sourceware.cygnus.com/insight/`
  - graficzny kompilator **lcc-win32** (z wbudowanym debuggerem)
    `http://www.cs.virginia.edu/~lcc-win32/`

- drzewa binarne
  - przeszukiwanie drzewa, wersja rekurencyjna

fragment sesji debuger-a gdb w trakcie rekurencyjnego wyświetlania drzewa binarnego

```
(gdb) list
147 void print_tree (struct node *top) {
148   if ( top == NULL )
149     return;
150   print_tree (top->left);
151   printf ("%s\n", top->word);
152   print_tree (top->right);

Zakładamy pułapkę. Program ma się zatrzymać na funkcji print_tree().
(gdb) break print_tree
Breakpoint 1 at 0x804890a: file binary_tree.cpp, line 148.

Programu jeszcze nie uruchomiliśmy. Stos jest więc na razie pusty.
(gdb) bt
No stack.

(gdb) run plik
Starting program: /home/jarek/plik
Breakpoint 1, print_tree (top=0x8049f10) at binary_tree.cpp:148
148 if (top == NULL)

Stan stosu programu
(gdb) bt
#0 print_tree (top=0x8049f10) at binary_tree.cpp:148
#1 0x080489d2 in main (argc=2, argv=0xbffffa74) at binary_tree.cpp:164
```

- drzewa binarne
  - przeszukiwanie drzewa, wersja rekurencyjna

```
Wyłączamy pułapkę, aby dalej wykonywać krokowo program. Ponieważ funkcja
print_tree() jest wywoływana rekurencyjnie sama z siebie, wiec pozostawienie tej
pułapki utrudni nam krokowa analizę programu.
(gdb) disable

Teraz, naciskając wiele razy s (polecenie gdb) dochodzimy do mementu z przed
wydrukowania funkcją printf() elementu 'O'
... wiele razy naciśnięto tu s
(gdb) s
150 print_tree(top->left);

Stan stosu w tym momencie
(gdb) bt
#0 print_tree (top=0x8049f70) at binary_tree.cpp:150
#1 0x08048920 in print_tree (top=0x8049f50) at binary_tree.cpp:150
#2 0x08048949 in print_tree (top=0x8049f10) at binary_tree.cpp:152
#3 0x080489d2 in main (argc=2, argv=0xbffffa74) at binary_tree.cpp:164

Kolejny krok programu. 'O' zostaje wydrukowane
(gdb) s
O
152 print_tree (top->right);
```


- drzewa binarne
  - przeszukiwanie drzewa, wersja NIE-rekurencyjna
  - zaproponuj własną implementację.
    Wskazówka: użyj własnego stosu na przykład zaimplementowanego jako lista 1-kierunkowa

- drzewa binarne
  - struktura danych adekwatna do budowy skorowidza słów (w pliku)

drzewo 2

```
struct node1 {
  char *word;                          // znalezione słowo w pliku
  int  freq;                           // jego krotność występowania
  struct node *left;
  struct node *right;
};

typedef struct node1 *Ttree2;          // wskaźnik do elementu
```


- drzewa binarne
  - wstawianie do drzewa

drzewo 2
funkcja iteracyjna

```
void insert2 (Ttree2 *root, char *_word)  {
  Ttree2 p, prev = NULL, new_node;      int v;

  new_node = (Ttree2) malloc(sizeof(struct node2));
  new_node->left = new_node->right = NULL;
  new_node->freq = 1;

  new_node->word = (char*) malloc(strlen(_word+1));
  strcpy (new_node->word, _word);

  // szukaj miejsca gdzie wstawić i czy wstawić
  p = *root;
  while ( p )
  {
    prev = p;
    if ((v = strcmp(p->word, _word)) == 0) {      // gdy słowo już jest w drzewie
      p->freq++;
      return;                                     // wyjście z funkcji
    }
    else if (v > 0)
      p = p->left;
    else
      p = p->right;
  }
  if ( !*root )                                   // drzewo puste
    *root = new_node;
  else if ( v > 0 )
    prev->left = new_node;
  else
    prev->right = new_node;  }
```

- drzewa binarne
  - wstawianie do drzewa - wersja ogólna
    (napisz wersję funkcji wstawiającą dowolny element do dowolnego drzewa)

```
// szkic rozwiązania
void insert1b ( void **root,
                eltype *_data,
                int (compare*) (eltype *, eltype *),        // porównanie elem.
                void (copy*) (eltype *, eltype *)           // przypisanie elem.
              );
```

```
void insert1 (Ttree1 *root, int _data) {
  Ttree1 p, prev = NULL, new_node;

  new_node = (Ttree1) malloc(sizeof(struct node1));
  new_node->left = new_node->right = NULL;
  new_node->data = _data;     // (*copy)(...)

  p = *root;
  while ( p )
  {
    prev = p;
    if (p->data > _data)      // (*compare)(...)
      p = p->left;
    else
      p = p->right;
  }
  if ( !*root )
    *root = new_node;
  else if (prev->data > _data)// (*compare)(...)
    prev->left = new_node;
  else
    prev->right = new_node;  }
```

te miejsca są mało ogólne

- drzewa binarne

drzewo 2
funkcja rekurencyjna

```
// wersja rekurencyjna wstawiania elementu do drzewa 2

// w funkcji main() wczytane słowo "pojawia się" w korzeniu
// w funkcji insert3() gdy jest taka potrzeba zostaje ono przesłane w dół
// do lewego lub prawego poddrzewa poprzez rekurencyjne wywołanie insert3()

Ttree2 insert3 (Ttree2 root, char *_word)  {      // poprzednio były dwie **
{                                                 // teraz zmodyfikowany wskaźnik
  int v;                                          // zwracamy z funkcji

  if (root == NULL) {                             // słowa _word NIE ma w drzewie
    root = Talloc();                              // utwórz więc węzeł
    root->word = Tstr(_word);
    root->freq = 1;
    root->left = root->right = NULL;
  }
  else if (( v = strcmp(root->word, _word)) == 0 )
    root->freq++;                                 // gdy słowo już jest w drzewie
  else if ( v < 0 )
    root->left  = insert3(root->left, _word);
  else
    root->right = insert3(root->right, _word);

  return root;
}

// po utworzeniu nowego węzła funkcja zwraca jego adres, który jest
// wstawiany zamiast zerowego wskaźnika w węźle przodka
```

- drzewa binarne

drzewo 2
funkcja rekurencyjna

```
// funkcje pomocnicze

Tree2 Talloc ( void )  {
{
  return (Ttree2) malloc (sizeof(struct node2));
}


char *Tstr ( char *str)
{
  char *p;
  p = (char*)malloc( strlen(str) + 1 )
  if ( p )
    strcpy(p, str);
  return p;
}


void printTree (Ttree1 root)
{
  if (root == NULL)
    return;
  printTree( root->left );
  printf("%d %s\n", root->freq, root->word);
  printTree( root->right );
}
```


- drzewa binarne

drzewo 2
funkcja rekurencyjna

```
// program główny

int main( void )
{
  Ttree2 root = NULL;                          // wskaźnik do korzenia
  char slowo[30];

  ...
  pobierz kolejne słowo z pliku
  ...

  root = insert3( root, slowo );

  printTree ( root );

  return 0;
}
```

- drzewa binarne, inne zagadnienia
  - przechodzenie drzewa wszerz
  - przechodzenie drzewa wgłąb metodą preorder oraz postorder
  - przechodzenie drzewa bez użycia stosy (konieczne dodatkowe wskaźniki do poprzednika danego węzła)
  - usuwanie elementów z drzewa (złożony problem, gdyż wymaga przebudowywania drzewa)
  - równoważenie drzewa
  - drzewa somoorganizujace się
    (dla danych napływających tak: B D K M P R, utworzy się lista !!!)
  - ...