Podstawy Programowania

Wykład nr 8: Rekurencja

dr hab. inż. Dariusz Dereniowski

Katedra Algorytmów i Modelowania Systemów
Wydział ETI, Politechnika Gdańska

Rekurencja – ogólna idea

Rekurencja – sposób implementacji algorytmu, polegający na zagnieżdżonym wywoływaniu funkcji z poziomu jej samej, najczęściej w celu rozwiązania „podproblemu" większego zadania obliczeniowego:

- rekurencja pozwala na łatwiejsze rozwiązywanie zadań zdefiniowanych indukcyjnym przepisem,
- w językach programowania, innych niż języki funkcyjne, procedura zostanie uruchomiona każdorazowo, nawet w przypadku wywołania z tym samym zestawem parametrów, co przy wcześniejszym wywołaniu,
- zmienne lokalne oraz argumenty wywołania funkcji są każdorazowo odkładane na stosie,
- rekurencja wiąże się ze zwiększonym narzutem pamięciowym (każde wywołanie funkcji powoduje odłożenie na stosie wskaźnika instrukcji oraz zawartości rejestrów w momencie wywołania) oraz narzutem czasowym.

Uwaga: każdą procedurę rekurencyjną można zastąpić przez równoważną procedurę iteracyjną (używając tylko pętli i pomocniczej struktury danych – stosu), nie zmieniając w sposób istotny czasu działania programu i ilości użytej pamięci.

Przykład 1: liczby Fibonacciego

- n-tą liczbę Fibonacciego, gdzie $n > 0$ jest liczbą całkowitą, obliczamy wg wzoru:

$$f_n = \begin{cases} 1, & \text{gdy } n = 1 \text{ lub } n = 2 \\ f_{n-2} + f_{n-1}, & \text{gdy } n > 2 \end{cases}$$

- Deklaracja funkcji obliczającej liczbę Fibonacciego o numerze n mogłaby wyglądać następująco: int fib(int n);. Jest ona oczywiście niezależna od sposobu implementacji (tzn. na przykład od tego czy funkcja będzie rekurencyjna czy też nie).

```
#include <iostream>
using namespace std;

/* Funkcja zwracająca n-tą liczbę Fibonacciego. Wywołanie funkcji
   z n <= 0 jest niepoprawne i wówczas funkcja zwraca 1 */
int fib( int n ) {
    if ( n <= 2 )
        return 1;
    else
        return fib(n-2) + fib(n-1);
}

int main() {
    for ( int i=1; i <= 10; i++ )
        cout << "fib(" << i << ") = " << fib(i) << endl;
    return 0;
}
```

```
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
fib(6) = 8
fib(7) = 13
fib(8) = 21
fib(9) = 34
fib(10) = 55
```

Przykład 2: zliczanie znaków w napisie

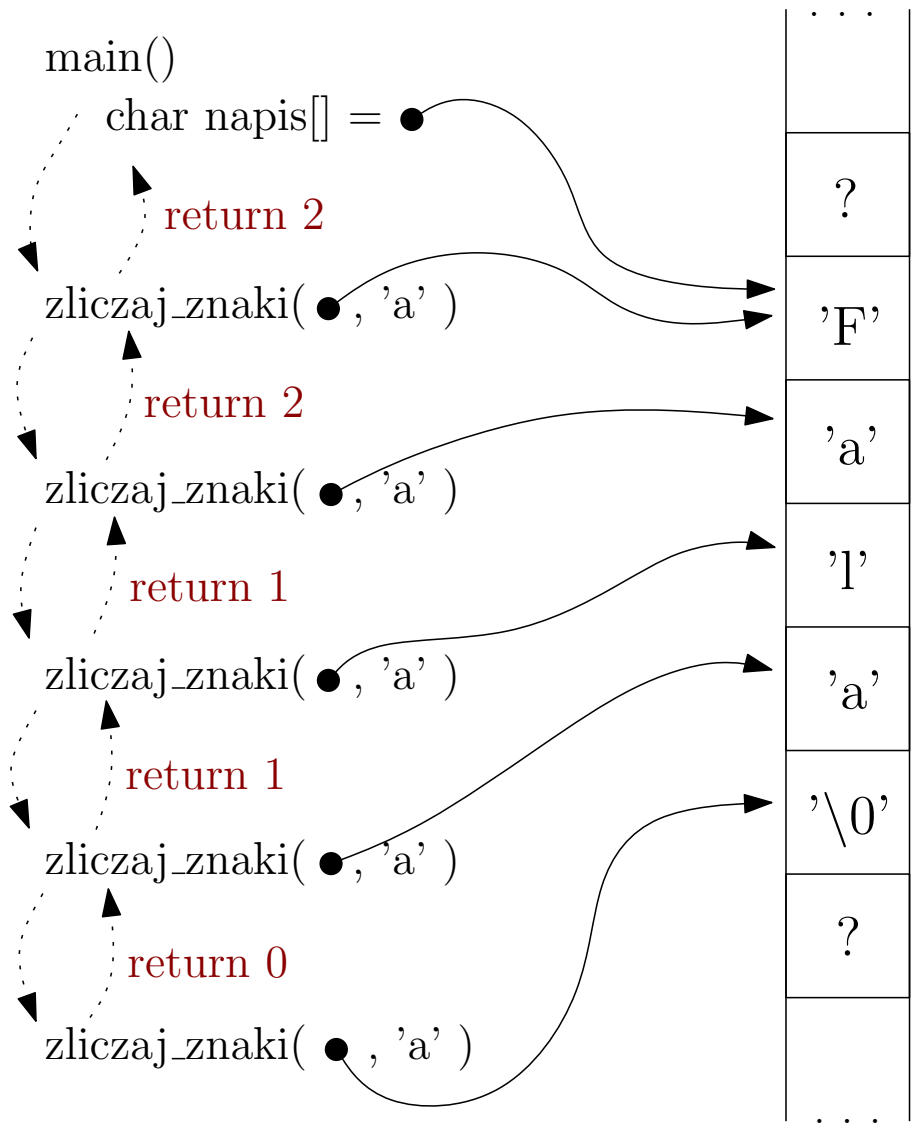
Napiszmy funkcję, która zlicza liczbę wystąpień zadanego znaku w podanym na wejściu napisie. Taka funkcja oczywiście przyjmie jako argument napis (typu char *). Skoro funkcja nie ma potrzeby modyfikować tego napisu (a nawet nie powinna tego robić!), użyjemy kwalifikatora const. Drugim argumentem będzie zliczany znak (typu char).

```
#include <iostream>
using namespace std;

/* Funkcja zwraca liczbę całkowitą równą liczbie wystąpień znaku podanego
   jako drugi argument w napisie s */
int zliczaj_znaki( const char *s, char znak ) {
   if ( *s == '\0' ) // jeśli napis jest pusty, to zwracamy 0
      return 0;
   else {
      int licz = zliczaj_znaki( s+1, znak );
      if ( *s == znak )
         licz++;
      return licz;
   }
}

int main() {  // Dołączamy prosty program testowy.
  char napis[] = { "Fala" };
  cout << zliczaj_znaki( napis, 'a' ) << endl;
  return 0;
}
```

Przykład 2: zliczanie znaków w napisie c.d.

Rysunek ilustruje kolejne wywołania funkcji zliczaj_znaki. Zauważ, że:

- W pamięci operacyjnej znajduje się tylko jedna kopia napisu "Fala".
- Poszczególne instancje funkcji zliczaj_znaki otrzymują kopie wskaźnika na odpowiednie miejsce w pamięci operacyjnej.
- Każda instancja funkcji zliczaj_znaki, za wyjątkiem ostatniej, posiada swoją własną kopię zmiennej licz.
- W związku w powyższym, przy czwartym wywołaniu funkcji zliczaj_znaki tworzona jest (na stosie) czwarta kopia zmiennej licz.



Przykład 3: wyszukiwanie największego elementu w tablicy

```
#include <iostream>
using namespace std;

int najwiekszy( int t[], int rozmiar ) {
    if ( rozmiar == 1 )
        return 0;
    else {
        int a = najwiekszy( t, rozmiar-1 );
        if ( t[a]>t[rozmiar-1] )
            return a;
        else
            return rozmiar-1;
    }
}
```

Funkcja int najwiekszy(int t[], int rozmiar) dzięki wywołaniu siebie samej:

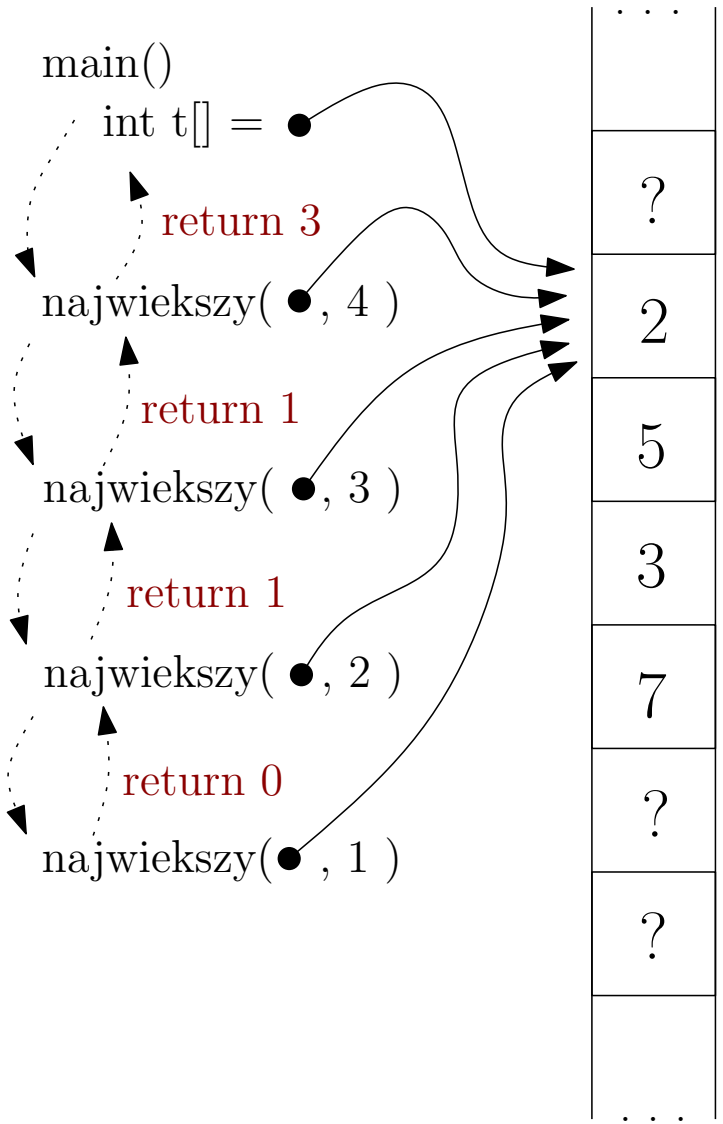
a = najwiekszy(t, rozmiar-1);

dowiaduje się jaki jest indeks a taki, że t[a] jest największą liczbą spośród t[0], t[1],...,t[rozmiar-2]. Innymi słowy, znajdujemy indeks najwięszego elementu tablicy t spośród jej pierwszych rozmiar-1 elementów.

Oznacza to, że indeks największego elementu spośród pierwszych rozmiar elementów to a lub rozmiar-1.

Przykład 3: wyszukiwanie największego elementu w tablicy c.d.

```
#include <iostream>
using namespace std;

int najwiekszy( int t[], int rozmiar ) {
    if ( rozmiar == 1 )
        return 0;
    else {
        int a = najwiekszy( t, rozmiar-1 );
        if ( t[a]>t[rozmiar-1] )
            return a;
        else
            return rozmiar-1;
    }
}
```



- Wywołanie najwiekszy dla tablicy {2,5,3,7} (rozmiar jest równy 4) jest pokazane na rysunku.
- Zauważmy, że dla tablicy rozmiaru n nastąpi dokładnie n wywołań funkcji najwiekszy.

Przykład 3: porządkowanie tablicy

```
#include <iostream>
using namespace std;

int najwiekszy( int t[], int rozmiar ) {    // Funkcja zwraca indeks
   if ( rozmiar == 1 )                      // w tablicy t, pod którym
      return 0;                             // znajduje się jej
   else {                                   // największy element.
      int a = najwiekszy( t, rozmiar-1 );
      return ( t[a]>t[rozmiar-1] ? a : rozmiar-1 );
   }
}

void zamien(int t[], int indeks1, int indeks2 ) {
   int pomocnicza = t[indeks1];     // Funkcja dokonuje zamiany elementów
   t[indeks1] = t[indeks2];         // pod indeksami indeks1 i indeks2
   t[indeks2] = pomocnicza;         // w tablicy t
}

void porzadkuj_tablice( int t[], int n ) {  // Funkcja porządkuje
   if ( n > 1 ) {                           // niemalejąco liczby
     zamien( t, n-1, najwiekszy( t, n ) );  // w talicy t o rozmiarze n
     porzadkuj_tablice( t, n-1 );
   }
}

int main() {  // Dołączamy prosty program testowy.
  int t[] = { 0, 41, 0, 4, 0, 9, 0, 4, 5, 3, 4, 5 , 6, 0, -3, -3, -1, 20 };

  porzadkuj_tablice( t, (sizeof t)/(sizeof t[0]) );
  for ( int i=0; i < (sizeof t)/(sizeof t[0]); i++ )
     cout << t[i] << endl;
  return 0;
}
```

Przykład 3: porządkowanie tablicy – komentarz do kodu

- W powyższej implementacji, funkcje najwiekszy oraz porzadkuj_tablice są rekurencyjne.
- Funkcja porzadkuj tablice najpierw sprawdza czy n>1. Działanie funkcji jest kontynuowane tylko wówczas, gdy warunek ten jest prawdziwy, gdyż w przeciwnym razie zadaniem takiego wywołania funkcji jest uporządkowanie jedno-elementowej tablicy – w takim wypadku niczego nie trzeba robić.
- Jeśli n>1, to funkcja porzadkuj_tablice dokonuje zamiany elementów, tak aby największy znalazł się na jej końcu, czyli pod indeksem n-1. Wywołanie porzadkuj_tablice(t, n-1) porządkuje pierwsze n-1 elementów tablicy t. Zauważ, że gdy to zostanie wykonane, tablica jest już uporządkowana!