Mechanizmy wywołania rekurencyjnego.

Na poprzednich zajęciach rozważaliśmy dwa proste przypadki definicji rekurencyjnych: pojęcia silni oraz wymiarów znormalizowanych formatów papieru serii A. Dziś rozważymy dwa kolejne proste przykłady, które posłużą do analizy działania funkcji, napisanej z użyciem wywołań rekurencyjnych.

Na początek przeanalizujmy następujący przykład (plik *binary.cpp*)

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

void binarnie(unsigned long);

int main()
{
unsigned long wartosc;

   cout<<"Program wypisuje binarna postac liczby"<<endl;
   cout<<"Podaj liczbe :";
   cin>>wartosc;
   cout<<"Wartosc zapisana binarnie :";
   binarnie(wartosc);
   cout<<endl;
}

void binarnie(unsigned long liczba)
{
   if(liczba>=2) binarnie(liczba/2);
   cout<< liczba % 2;
}
```

Powyższy program wypisuje na ekran liczbę, podaną przez użytkownika, w postaci binarnej – zerojedynkowej. Jak uzyskać postać binarną danej liczby? Należy wziąć, w odwrotnej kolejności, reszty z dzielenia przez dwa (ilości możliwych wartości cyfry binarnej) – najpierw szukanej liczby, a następnie kolejnych wyników dzielenia poprzedniej liczby przez dwa. Procedurę należy kontynuować, dopóki dzielna nie stanie się liczbą, mniejszą od 2 - stanowi ona pierwszą z cyfr binarnych wyniku. Pozostałe cyfry to reszty z dzielenia, zapisane w odwrotnej kolejności. Na przykład dla liczby 13 znalezienie jej postaci binarnej wymaga przeprowadzenia następujących obliczeń:

$$\begin{aligned} 13 &= 6 \cdot 2 + 1 \\ 6 &= 3 \cdot 2 + 0 \\ 3 &= 1 \cdot 2 + 1 \\ 1 & \end{aligned}$$

Według powyższej reguły liczba 13 ma następującą postać binarną: 1101. Właśnie konieczność odwrócenia kolejności reszt jest przyczyną zastosowania rekurencji; gdyby reszty można było wziąć po kolei, procedurę można by było w prosty sposób zaprogramować, korzystając z pętli.

Funkcja rekurencyjna *binarnie* działa w następujący sposób: jeżeli zapis binarny wartości argumentu funkcji ma więcej niż jedną cyfrę (co równoważne jest stwierdzeniu, że wartość argumentu jest równa co najmniej dwa), to poprzez wywołanie rekurencyjne wymuszane jest wypisanie *najpierw* poprzedzających ją cyfr, a następnie wypisywana jest reszta z dzielenia podanej liczby przez dwa.

Do rozważenia działania funkcji rekurencyjnej konieczne jest przeanalizowanie procesu wywołania funkcji. Umieszczając w programie wywołanie funkcji oczekujemy, że po wykonaniu

czynności, realizowanych przez wywołaną funkcję, nastąpi realizacja dalszych zaprogramowanych czynności. Aby było to możliwe, w momencie wywołania funkcji należy zapamiętać adres w pamięci, pod którym znajduje się instrukcja, od której należy kontynuować wykonanie programu (tzw. adres powrotu) po zakończeniu działania wywołanej funkcji. Ponieważ wywoływana funkcja może zawierać kolejne wywołania pewnych funkcji, konieczne jest sekwencyjne zapamiętywanie adresów powrotu. W miarę kończenia działania poszczególnych wywołań funkcji, należy przywracać działanie programu w kolejnych zapamiętanych adresach, przy czym adresy te wykorzystywane są w *odwrotnej kolejności*. Zapisywanie sekwencji adresów powrotu wykonywane jest na tzw. *stosie* – strukturze danych, cechującej się tym, że dane zapamiętane w niej jako ostatnie, zostaną odczytane jako pierwsze (ang. LIFO – *Last In-First Out*), co oznacza po prostu, że dane ze stosu pobiera się w odwrotnej kolejności. Dokładnie takiego zachowania oczekujemy od struktury danych, przechowującej adresy powrotu z funkcji, stąd realizacja procesu wywoływania funkcji w oparciu o stos jest powszechna i ma wsparcie w zestawie instrukcji popularnych mikroprocesorów. Rola stosu w procesie wywoływania funkcji nie ogranicza się do przechowywania adresów powrotu z funkcji. Na stosie przechowywane są również zmienne lokalne funkcji, jej argumenty, a w zależności od sytuacji, może być również przechowywana wartość, zwracana przez funkcję.

Spójrzmy teraz na listing. Wewnątrz funkcji *main* wywołujemy funkcję *binarnie* – na stosie zostaje zapisany adres powrotu do funkcji *main* oraz wartość argumentu, która będzie dostępna z wnętrza funkcji *binarnie* jak zmienna o nazwie *liczba*. Jak zatem widać, wywołanie funkcji sprowadza się do zapamiętania adresu powrotu oraz utworzeniu kontekstu działania funkcji (argumenty, zmienne lokalne) poprzez zapisanie tych informacji na stosie, a następnie przekazania sterowania do punktu, w którym w pamięci znajdują się instrukcje, przeznaczone do realizacji w wywoływanej funkcji. Użycie stosu jest przyczyną jednego z największych niebezpieczeństw rekurencji – wielokrotne wywołania funkcji (nie tylko rekurencyjne) mogą po prostu spowodować wyczerpanie się pamięci, przeznaczonej na stos, czego skutki na ogół są fatalne. Proszę zauważyć, że nie istnieją powody techniczne, uniemożliwiające wywołania funkcji z wnętrza niej samej – nowe wywołanie funkcji będzie działać w ramach swojego własnego kontekstu, a po jego zakończeniu nastąpi powrót pod zapamiętany adres. Nie ma żadnych przeciwwskazań, aby lista instrukcji funkcji wywoływanej i wywołującej nie mogły być takie same, stąd też współczesne języki programowania zazwyczaj udostępniają możliwość wywołań rekurencyjnych funkcji.

Rozważając wywołania rekurencyjne należy pamiętać o następujących faktach:

1. wywołanie funkcji z wnętrza samej siebie to nie skok do początku funkcji: tworzony jest nowy kontekst wywołania, a więc nowo wywołana funkcja ma swoje własne argumenty, zmienne lokalne oraz wartość zwracaną, jakkolwiek realizuje tę samą listę instrukcji, co funkcja wywołująca.
2. do czasu zakończenia wykonania funkcji wywoływanej, zatrzymane jest działanie funkcji wywołującej.

Obie powyższe uwagi znajdują swoje odzwierciedlenie w działaniu funkcji *binarnie* w omawianym programie. Zadaniem funkcji jest wypisanie na ekran jednej cyfry binarnej oraz ewentualne wywołanie rekurencyjne. Jeśli wartość argumentu jest większa niż jeden wiadomo, że trzeba będzie wypisać więcej niż jedną cyfrę binarną (ponieważ każda liczba większa lub równa 2 ma w zapisie binarnym co najmniej dwie cyfry), a zatem będzie co najmniej jedna cyfra poprzedzająca w zapisie resztę z dzielenia argumentu funkcji przez dwa, jeśli zaś argument nie jest większy od dwóch, wywołanie rekurencyjne zostaje pominięte. Aby wypisać poprzedzające cyfry, wywoływana jest funkcja *binarnie* z nowym argumentem, będącym wynikiem dzielenia wartości aktualnego argumentu przez dwa. Po zakończeniu jej działania (lub po pominięciu wywołania rekurencyjnego, gdy argument funkcji był mniejszy od dwóch) zostaje wypisana reszta z dzielenia przez dwa wartości argumentu. Oczywiście każde wywołanie funkcji *binarnie* powoduje wykonanie dokładnie tej samej sekwencji czynności, uzależnionej od aktualnych wartości argumentu wywołania. Ciąg wywołań rekurencyjnych będzie miał długość, równą ilości cyfr binarnych liczby,

podanej jako argument pierwszego wywołania funkcji.

Wykorzystajmy jeszcze raz możliwość zrealizowania odwracania elementów za pomocą rekurencji. Rozważmy następujący przykład (plik *reverse.cpp*):

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

void reverse();

int main()
{
   cout<<"Program demonstruje odwracanie kolejnosci przy wypisywaniu"<<endl;
   cout<<"Poproszony zostaniesz o wpisanie liczb. Aby zakonczyc, podaj zero"<<endl;
   cout<<endl;
   reverse();
   cout<<endl;
}

void reverse()
{
unsigned long liczba;

   cout<<"Podaj liczbe (zero konczy, nie zostanie wypisane):";
   cin>>liczba;
   if(liczba!=0)
   {
         reverse();
         cout<<liczba<<" ";
   }
}
```

O ile w pierwszym programie korzystaliśmy głównie z możliwości odroczenia wypisania cyfry binarnej, tutaj posłużymy się zmienną lokalną w każdym z wywołań rekurencyjnych jak „notatnikiem“ na wartość jednej liczby do wypisania. Program ma pobrać od użytkownika kilka liczb do wypisania, a następnie wypisać je w odwrotnej kolejności. Co robi funkcja *reverse*? Po wywołaniu, wypisuje komunikat dla użytkownika, a następnie pobiera od niego jedną liczbę, którą zapamiętuje w zmiennej *liczba*. Następnie, jeśli podano wartość 0 (co ma oznaczać koniec podawania liczb przez użytkownika) kończy się, zaś w przeciwnym wypadku:

1. wywołuje się rekurencyjnie
2. wypisuje zapamiętaną uprzednio liczbę