26. Powtarzanie pętli i wywoływanie siebie, czyli iteracja i rekurencja w algorytmach

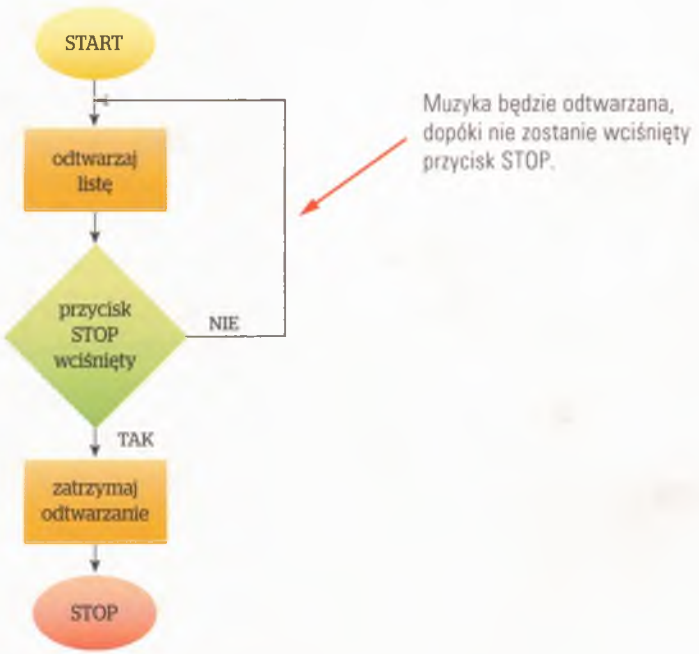
NA TEJ LEKCJI:

- poznasz różnicę pomiędzy iteracją a rekurencją w programowaniu;
- poznasz przykłady procesów rekurencyjnych i iteracyjnych.

Czy można znaleźć odpowiedź na pytanie, która z metod – iteracyjna czy rekurencyjna – jest lepsza? Jak to zwykle bywa w informatyce, nie można dać jednoznacznej odpowiedzi. Aby dobrać optymalne rozwiązanie, należy przeanalizować dany problem. Nie jest to łatwe. W zrozumieniu istoty obu rozwiązań pomoże wam lektura rozdziału.

26.1. Wiele razy ta sama sekwencja, czyli czym jest iteracja

Wyobraź sobie, że słuchasz utworów muzycznych przypisanych w telefonie do jednej listy. Chcesz, aby była ona odtwarzana, dopóki nie wciśniesz przycisku STOP. Ułóżmy algorytm takiej sytuacji (rys. 26.1.).



Rys. 26.1. Przykład algorytmu rekurencyjnego

W tym przypadku proces będzie się powtarzał, dopóki przycisk STOP nie zostanie wciśnięty. Nie można więc określić, jak długo ani ile razy usłyszysz ten sam utwór. Można za to użyć w programie pętli `while`, która będzie powtarzać kod, dopóki warunek jest spełniony. W naszym przypadku konieczne będzie zastosowanie warunku odwrotnego (np. przez użycie negacji !) niż w algorytmie. Odtwarzanie ma bowiem nastąpić, gdy przycisk **nie jest wciśnięty**, a więc gdy warunek z algorytmu ma wartość **false** (nie jest spełniony).

```
while (!(przycisk wciśnięty))
{odtwarzaj listę utworów}
```

Przypomnij sobie, że oprócz pętli `while` znasz także pętlę `do while`. Jej działanie jest podobne do `while`, z tym że kod zostanie wykonany przynajmniej raz, ponieważ **badanie warunku odbywa się na końcu pętli**.

```
do
{
//kod wykonywany dopóki warunek jest spełniony
}
while (warunek);
```

Gdybyśmy w naszym przykładzie użyli `do while`, przycisk zadziałałby dopiero po jednokrotnym odtworzeniu listy.

Algorytm można zmodyfikować i wprowadzić do aplikacji odtwarzającej muzykę opcję liczby odtworzeń (rys. 26.2.).

Rys. 26.2. Przykład algorytmu odtwarzania listy muzycznej *L* razy

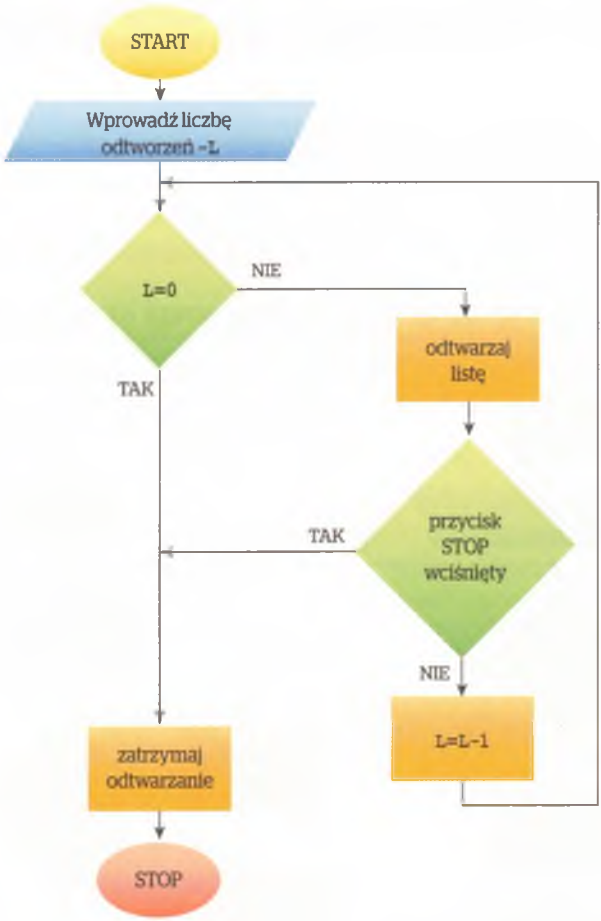
W tym przypadku można użyć pętli `for`, która będzie wykonywana *L* razy.

Przypomnienie: W pętli `for` nadaje się wartość początkową zmiennej (można jednocześnie zadeklarować jej typ), która występuje w warunku, i podaje się,

z jakim krokiem ma się zmieniać, np. --, co oznacza, że zmienna będzie po każdym przejściu pętli pomniejszana o 1. W naszym przypadku pętla będzie wykonywana *L* razy, czyli do osiągnięcia wartości 0 przez zmienną *i*.

```
for (int i=L; i=0; i--)
{
odtwarzaj listę utworów
}
```

Można także połączyć oba algorytmy tak, by odtwarzanie było zakończone po *L* odtworzeniach lub po wciśnięciu przycisku *STOP* (rys. 26.3.). W naszym przykładzie przycisk *STOP* zadziała także w trakcie odtwarzania listy.



Rys. 26.3. Przykładowy algorytm działania aplikacji odtwarzającej listę muzyczną

Co jest wspólną cechą tych algorytmów? **Powtarzanie**. Odtwarzanie listy utworów muzycznych trwa do momentu zajścia pewnych okoliczności – wciśnięcia przycisku *STOP* lub wykonania określonej liczby odtworzeń. Taki właśnie proces nazywamy **iteracją**.

iteracja

Iteracja (łac. *iteratio* – powtarzanie) to czynność wykonywana określoną liczbę razy lub do spełnienia danego warunku.

26.2. Wywołanie do siebie, czyli rekurencja w informatyce

W internecie można znaleźć pewną sentencję nieznanego autora podpowiadającą, jak zrozumieć pojęcie rekurencji:

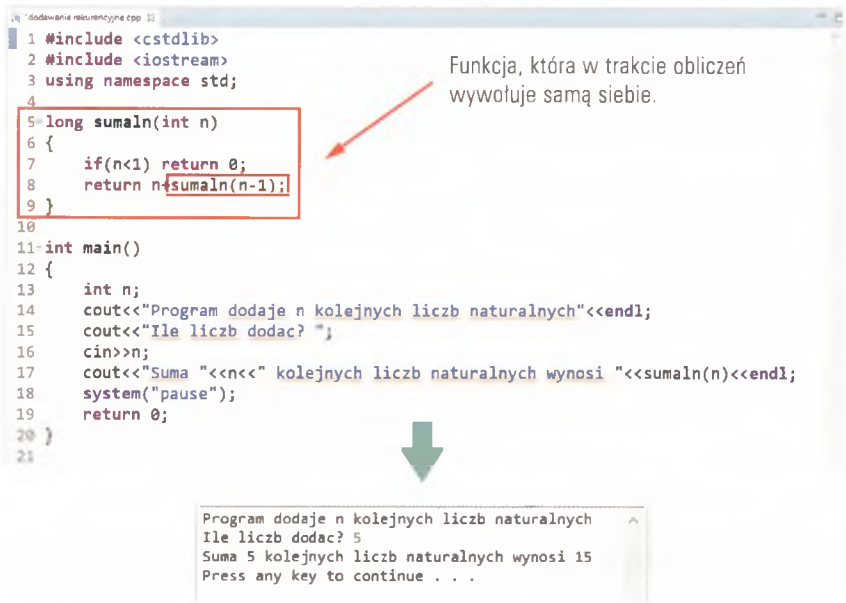
Żeby zrozumieć rekurencję, musisz najpierw zrozumieć rekurencję.

Z pozoru to zdanie nie ma sensu, każe bowiem wyjaśnić zjawisko, korzystając z wyjaśnienia tego zjawiska. Jednak w świecie informatyki nie jest ono pozbawione logiki, gdyż **rekurencja** to wywoływanie funkcji przez samą siebie. Oznacza to, że wewnątrz ciała funkcji może się znajdować odwołanie do niej samej.

Z punktu widzenia informatyki **rekurencją** nazywamy odwoływanie się funkcji lub definicji do samej siebie.

rekurencja

Przykładem zagadnienia, które można programowo rozwiązać metodą rekurencyjną, jest obliczanie sumy pewnej liczby kolejnych liczb naturalnych (rys. 26.4.).



Rys. 26.4. Przykład programu wykorzystującego rekurencję

Program główny to jedynie wprowadzanie danej i wyprowadzenie wyniku poprzez wywołanie funkcji. O wiele ciekawsza jest funkcja `sumaln`, ponieważ w swoim ciele wywołuje samą siebie. Dzięki temu dwie niepozorne linie dokonują obliczeń dla dowolnej liczby liczb ograniczonej wielkością miejsca zarezerwowanego dla wyniku działania funkcji (w tym przypadku `long`, czyli 32 bity).

```
long sumaln(int n)
{
      if(n<1) return 0;
      return n+sumaln(n-1);
}
```

Jak to się dzieje? Otóż funkcja zostaje wywołana po wczytaniu konkretnej liczby *n*. Gdy *n<1*, zwraca wartość *0* i kończy pracę. Kiedy osiągnie *0* (pomijając wprowadzenie przez użytkownika wartości *n=0*)? Wtedy, gdy doda już do siebie wszystkie kolejne liczby, począwszy od *n*-tej, aż do *1*. A jak je dodaje? W pierwszym kroku do *n* dodaje wynik funkcji `sumaln` dla argumentu *n-1*, czyli sumę wszystkich liczb naturalnych od *1* do *n-1*. W drugim kroku (wewnątrz rekurencyjnego wywołania funkcji `sumaln`) liczona jest wartość sumy *n-1* liczb naturalnych poprzez zsumowanie liczby *n-1* z sumą liczb naturalnych od *1* do *n-2*, czyli kolejnym rekurencyjnym wywołaniem funkcji `sumaln`. Tak dzieje się do momentu, gdy *n=0*. Wtedy zwracana jest wartość *0* bez wywoływania kolejnej rekurencji. Komputer składa to w odwrotnej kolejności, czyli do *1* dodaje wynik wywołania funkcji `sumaln(0)`, która przyjmuje wartość *0*. W ten sposób wróciliśmy z ostatniej rekurencji. To stanowi wynik funkcji `sumaln(1)`. Wynik ten jest dodany do liczby *2* i staje się wynikiem kolejnej rekurencji *(2+1=3)*. Wracając z następnej rekurencji, otrzymujemy *3+3=6*. Następnie *4+6=10*.

Prześledźmy to na przykładzie. Załóżmy, że wprowadzono liczbę 5. Jak widać, program podał **wynik 15**.

```
sumaln(5)=5+sumaln(4)=5+(4+sumaln(3))=5+(4+(3+sumaln(2)))=5+(4+(3+(2+(1+0))))=
5+(4+(3+(2+1)))=5+(4+(3+3))=5+(4+6)=5+10=15
```

stosowanie rekurencji i iteracji

W wielu przypadkach problemy informatyczne można rozwiązać, stosując zarówno **rekurencję**, jak i **iterację**. Programy oparte na iteracji zazwyczaj szybciej prowadzą do wyniku z powodu mniejszej liczby wywołań funkcji (mogą także nie zawierać funkcji). Programy z zastosowaniem rekurencji są zazwyczaj prostsze w budowie, a kod jest krótszy. Wybór metody należy dostosować do rozwiązywanego problemu lub algorytmu. **Iteracji** używamy głównie do przeprowadzania obliczeń na danych tablicowych lub gdy musimy wykonać pewną liczbę powtórzeń podobnej czynności. Wyznacznikiem dla użycia **iteracji** jest to, że dane pomiędzy cyklami nie muszą na siebie wpływać. **Rekurencji** używamy, gdy w kolejnych wywołaniach rekurencji używamy wyniku poprzedniego wywołania, na przykład przy liczeniu silni lub w omawianym przykładzie dodawania kolejnych liczb.

ZADANIA DO ROZWIĄZANIA

1. Wykonaj prezentację, w której przedstawisz definicje rekurencji i iteracji w informatyce. Dla każdej z nich podaj jeden przykład problemu i jego rozwiązanie.
2. Przedstaw algorytm z rysunku 26.3. w zapisie pseudokodu. Sprawdź jego działanie, opisując reakcję na polecenia użytkownika.
3. Wykonaj prezentację, w której opiszesz różnice pomiędzy poznanymi rodzajami pętli występujących w języku C++ – **while**, **do...while** i **for**. Podaj przykład problemu, który można rozwiązać przy użyciu każdej z nich.
4. Zmodyfikuj program z rysunku 26.4., aby po podaniu wyniku wyświetlił pytanie: *Czy chcesz powtórzyć obliczenia dla innej liczby liczb?* i reagował na odpowiedź t – tak lub n – nie. W przypadku wybrania t program powinien ponownie zapytać o liczbę liczb i wykonać obliczenia. Natomiast jeśli użytkownik wybierze n, program zakończy działanie i wyświetli zwrot grzecznościowy.

5*. Ułóż program obliczający silnię metodą rekurencyjną. Zastanów się, jakiego typu zmiennej użyjesz do przechowywania wyniku. Uzasadnij decyzję.