Styl kodowania

Wstęp

Jest wiele czynników wpływających na jakość programu komputerowego. Kluczowe to poprawność i szybkość. Natomiast cecha programu komputerowego, którą można określić mianem "czytelności", jest bardzo często przez początkujących programistów niedostrzegana. Póki programy są krótkie, faktycznie może się wydawać, że czytelność programu nie ma znaczenia. Okazuje się jednak, że czytelność to krytyczny czynnik, kiedy zachodzi jedna z poniższych sytuacji:

- program pisze się dłużej niż jeden dzień,
- program zajmuje więcej niż kilkaset linii,
- program jest podzielony na więcej niż jeden plik źródłowy,
- program jest pisany przez więcej niż jedną osobę.

W każdej z wymienionych sytuacji zachodzi ciekawe zjawisko: fragmenty kodu źródłowego więcej razy się czyta niż się je pisze (tzn. wpisuje pierwszy raz i potem poprawia). W związku z czym opłaca się spędzić trochę więcej czasu i bardziej czytelnie napisać kawałek kodu, bo później zaoszczędzimy ten czas podczas czytania.

Jak pisać czytelne programy? Generalna zasada jest taka, że należy przestrzegać pewnych konwencji. Konwencje można ustalić samodzielnie lub w grupie, najlepiej jednak dostosować się do takich konwencji, które są stosowane powszechnie na świecie. Dzięki temu nam będzie łatwiej czytać programy innych osób, których możemy w ogóle nie znać, a innym osobom będzie łatwiej czytać nasze programy. Konwencje określają, jak wykonać pewną czynność (np. jak napisać pętlę for) w sytuacji, gdy daną czynność można wykonać na więcej niż jeden sposób. Oprócz tego są jeszcze dobre praktyki, czyli czynności których nie trzeba wykonywać (np. komentowanie kodu), ale być może warto to robić.

Styl kodowania a ocena

Pisanie czytelnych programów to naprawdę ważna umiejętność, którą programista musi posiąść. Praktyka pokazuje, że jeśli programista nie wyrobi w sobie nawyku pisania czytelnego kodu źródłowego na samym początku nauki programowania, to później ciężko będzie mu się przestawić na czytelny styl pisania. W związku z tym czytelność programów będzie brana pod uwagę przy wystawianiu ocen zadań zaliczeniowych. Sposób, w jaki czytelność rozwiązania będzie wpływała na ocenę, będzie podawany oddzielnie dla każdego zadania zaliczeniowego.

Nakazy czy wskazówki?

Jak głosi powiedzenie, "nigdy nie mów nigdy". Nie można powiedzieć, że jakaś konwencja lub praktyka jest jedyna słuszna i zawsze należy ją stosować. Od każdej takiej reguły można znaleźć wyjątek. Dlatego wszystkie opisane w tym dokumencie konwencje i praktyki mają charakter dobrych rad. Jeżeli ktoś nie będzie stosował się do opisanych tutaj wskazówek, to oczywiście nie oznacza to automatycznego obniżenia oceny, ale w takiej sytuacji należy zastosować własne rozwiązanie problemu opisywanego we wskazówce. Kluczem jest konsekwencja - jeżeli np. nie podoba Ci się zaprezentowany przez nas styl pisania pętli for, możesz użyć własnego, ale musisz być w tym konsekwentny.

Jak czytelnie kodować?

Ogólna struktura programu

Program
Typowy program składa się z kilku sekcji. Warto podawać je w następującej kolejności:

1. Instrukcja **`program`**.
2. Załączenie wszystkich niezbędnych modułów za pomocą instrukcji **`uses`**.
3. Definicje wszystkich stałych za pomocą instrukcji **`const`**.
4. Definicje wszystkich typów za pomocą instrukcji **`type`**.
5. Definicje funkcji i procedur.
6. Deklaracje globalnych zmiennych za pomocą instrukcji **`var`**.
7. Treść programu.

Sekcje powinny być od siebie oddzielone co najmniej jedną pustą linią. Tak samo procedury i funkcje powinny być od siebie pooddzielane co najmniej jedną linią.

Procedury i funkcje
Każda procedura i funkcja powinna mieć strukturę podobną do całego programu. Ponadto żadna procedura czy funkcja nie powinna być zbyt długa - jeżeli nie mieści się w całości na ekranie, to jest to znak, że być może należy podzielić ją na więcej procedur lub funkcji. Pomocna bywa również następująca zasada: Jeżeli procedura lub funkcja zawiera więcej niż 10-15 instrukcji sterujących (tzn. **`if`**, **`for`**, **`while`**, **`repeat`**), to jest to znak, że należy ją podzielić na mniejsze procedury lub funkcje.

Kiedy tylko zachodzi potrzeba użycia podobnego kawałka kodu w więcej niż jednym miejscu, należy taki kawałek kodu wydzielić i

stworzyć z niego procedurę lub funkcję. Na pewno tyczy się to identycznych kawałków kodu pojawiających się w wielu miejscach w programie. Czasami jednak podobieństwo i możliwość wydzielenia procedury lub funkcji nie są oczywiste.

Program komputerowy realizuje pewne zadanie. Takie zadanie składa się z mniejszych podzadań, a te podzadania znów mogą składać się z jeszcze mniejszych podzadań. Podział na procedury i funkcje powinien odzwierciedlać podział na zadania, to znaczy każda procedura i funkcja powinny realizować pewien konkretny cel, który jest oddzielony od reszty programu. Idealnie, każda procedura czy funkcja powinny być tak napisane, żeby mogły być przeniesione do innego programu, który ich potrzebuje, i działać bez konieczności wprowadzania jakichkolwiek poprawek.

Instrukcje złożone
Instrukcje złożone nie powinny być zbyt długie. Jeżeli instrukcja złożona ma więcej niż kilkanaście linii, to być może warto zamienić ją na wywołanie procedury.

Zmienne globalne
Zmienne globalne celowo powinny znaleźć się dopiero tuż przed treścią programu, żeby procedury i funkcje nie mogły z nich korzystać. Jeżeli procedury lub funkcje chcą się komunikować ze sobą lub z programem poprzez zmienną globalną, to coś jest nie tak.

Stałe
Gdy w programie pojawiają się stałe liczbowe inne niż 0 lub 1, to na ogół oznacza to konieczność wprowadzenia stałej i używania jej zamiast liczby. Liczby w programie mają to do siebie, że często się zmieniają, a jeśli dana liczba występuje w paru miejscach i oznacza tę samą wielkość, to wprowadzenie stałej zdecydowanie ułatwia operację zmiany tej wielkości. Stałe napisowe to szeroki temat, zwłaszcza w kontekście tzw. internacjonalizacji.

Prostota wyrażania się
Przed napisaniem kawałka kodu zawsze warto "wyprostować" swoje myśli i zamiast bezpośrednio przekształcać je w kod, najpierw spróbować poszukać prostszej, bardziej czytelnej formy. Ilustruje to poniższy przykład. Drugie wyrażenie ma bardziej przystępną formę, niż pierwsze, a trzecie jest najprostsze, najkrótsze i najbardziej czytelne.

```
not ((x < 3) or (x > 5))
(x >= 3) and (x <= 5)
x in [3..5]
```

Instrukcja goto
ZABRONIONA!

Komentarze

Komentarze należy stosować w następujących sytuacjach:

Miejsce w kodzie	Co powinien zawierać komentarz
początek programu	ogólny opis programu: autor, data powstania, co program robi, jak go uruchamiać
typ definiowany przez użytkownika	do czego służy, co oznaczają poszczególne składowe
procedura lub funkcja	co robi, jak szybko to robi, informacje nt. przekazywanych argumentów, wymagane warunki wstępne, zapewniane warunki końcowe
dowolny niebanalny kawałek kodu	jak działa i dlaczego jest zapisany akurat w ten sposób

Komentowanie we wszystkich wyżej wymienionych sytuacjach to czasami przesada. Często dzięki dobremu nazewnictwu kawałek kodu nie wymaga komentarza, np. poniższa funkcja:

```
function factorial(n : Integer) : Integer;
begin
    if n = 0 then begin
        factorial := 1;
    end else begin
        factorial := n * factorial(n - 1);
    end;
end;
```

Nazewnictwo

Czynnikiem, dzięki któremu można pominąć dużą część komentarzy, jest sposób nazywania różnych bytów w programie: zmiennych, procedur, funkcji, typów. Nazwa powinna odzwierciedlać przeznaczenie bytu. Nazwy zmiennych, typów i funkcji są na ogół rzeczownikami, nazwy procedur są na ogół czasownikami z okolicznikami. Zachęcamy do stosowania angielskich nazw - dzięki temu program będzie zrozumiały dla większej ilości osób. Nazwy powinny być długie. Stosowanie skrótów to duży grzech - ciężko się domyślić, do czego służy zmienna *a*, *ilObj* itd. Wyjątkiem są tu nazwy odpowiadające konwencjom matematycznym, czyli np. *i*, *j* oraz *k* dla indeksów lub *x*, *y* oraz *z* dla współrzędnych. Użycie nazw *m* lub *n* na oznaczenie ilości pewnego rodzaju obiektów bywa już kłopotliwe.

Jeżeli nazwa składa się z kilku wyrazów, to jest parę konwencji zapisania jej w programie:

1. *laczenie_wyrazow_srednikiem_male_litery*
2. *LACZENIE_WYRAZOW_SREDNIKIEM_DUZE_LITERY*
3. *KonwencjaZwanaCamelCase*
4. *camelCasePisanaMalaLitera*

Proponujemy stosować powyższe konwencje następująco:

- nazwy zmiennych, funkcji i procedur - konwencja pierwsza,
- nazwy stałych - konwencja druga,
- nazwy typów - konwencja trzecia.

Formatowanie fragmentów kodu

Jeśli porównać pisanie programów do pisania wypracowań, to powyższe punkty można porównać do sposobu układania zdań i słownictwa. Pozostaje jeszcze charakter pisma, co w przypadku pisania programów przekłada się na sposób rozmieszczenia białych znaków w programie.

Szerokość linii
Dobrym zwyczajem jest pisanie linii nie dłuższych niż 80 znaków. Terminale tekstowe wciąż są w powszechnym użyciu, a nawet w środowisku graficznym, gdy źródła nie są szersze niż 80 znaków, można wyświetlić kilka plików obok siebie. Jeżeli instrukcja jest dłuższa niż 80 znaków, trzeba ją jakoś podzielić na wiele linii. Np. wywołanie funkcji można przepisać w taki sposób, żeby kolejne argumenty znalazły się w oddzielnych liniach.

Wcięcia
Najważniejszą zasadą jest stosowanie wcięć. Każda instrukcja złożona (czyli ciąg instrukcji pomiędzy słowami kluczowymi **`begin`** i **`end`**) powinna być wcięta o jeden raz więcej, niż otaczający ją kod. Najlepsze wcięcie to cztery spacje. Jeśli ktoś chce używać znaków tabulacji, to musi mieć na uwadze, że różne edytory różnie je wyświetlają. Absolutnie niedopuszczalne jest mieszanie różnego rodzaju wcięć.

Białe znaki na końcu linii
Bardzo nieładny zwyczaj.

Spacje w kodzie
Stosować m.in. w następujących sytuacjach:

- dookoła dwukropka przy definiowaniu zmiennych,
- dookoła operatora przypisania,
- dookoła operatorów arytmetycznych i porównujących,
- po przecinku oddzielającym argumenty w wywołaniu procedury lub funkcji.

Nie stosować m.in. w następujących sytuacjach:

- dookoła nawiasu otwierającego przy wywołaniu procedury lub funkcji,
- przed nawiasem zamykającym przy wywołaniu procedury lub funkcji,
- przed średnikiem kończącym instrukcję,
- za nawiasem kwadratowym otwierającym i przed nawiasem kwadratowym zamykającym,
- dookoła kropki.

Instrukcje sterujące
Poniższe kawałki kodu obrazują, jak powinny być sformatowane poszczególne instrukcje sterujące.

```
if warunek then
    instrukcja
else
    instrukcja;

if warunek then begin
    instrukcja;
    instrukcja;
end else begin
    instrukcja;
    instrukcja;
end;

for i := 1 to 10 do
    instrukcja;

for i := 1 to 10 do begin
    instrukcja;
    instrukcja;
end

while warunek do
    instrukcja;
```

```
while warunek do begin
    instrukcja;
    instrukcja;
end;

repeat
    instrukcja;
    instrukcja;
until warunek;
```

Ostatnia modyfikacja: Saturday, 8 March 2014, 21:23
Moodle, wersja 2.2.1 | moodle@mimuw.edu.pl
▲ Back to Top