Literały

- Program może zawierać zmienne, które na pierwszy rzut oka nie wyglądają jak zmienne, i nie są deklarować. Na przykład w programie mogą się pojawić wyrażenia:
  - `cout << "Hello world";`
  - `i*=2;`
  - `double r = 10.53;`
  - `if( m > 12 )`
- To są tak zwane stałe zdefiniowane (literały): `"Hello world"`, `2`, `10.53`, `12`. Każda taka stała musi być w programie wykonywalnym gdzieś zapisana, i jest jego częścią
- Literały zapisywane są tak samo jak zmienne, z tym, że nie mają nazw, nie można się do nich odnosić (z wyjątkiem samej wartości) i nie można zmieniać ich wartości. Kompilator sam wybiera typ takiej zmiennej, starając się dobrać typ najbardziej oszczędny pod względem zajętej pamięci

Literały

- Na przykład, `"Hello world"` zostanie zapisany jako tak zwany string. O stringach będziemy mówili wkrótce. String to tablica `char`'ów, typ pochodny. 2 i 12 zostaną zapisane przypuszczalnie jako inty, a 10.53 jako float.
- Kompilator sam dopasowuje typ zmiennej do przechowywania literału. Można jednak na to wpłynąć używając sufiksów np. 0x1AuL to liczba 26 ale przechowywana w typie `unsigned long`. 0b1101f to liczba 13, ale zapisana jako typ `float`
- Stałe będące liczbami całkowitymi można zapisywać w różnych systemach:
  - bez prefiksu, np. 15 - system dziesiętny
  - prefiks 0b, np 0b1101 - system dwójkowy 0b1101=13
  - prefiks 0, np. 016 - system ósemkowy 016=14
  - prefiks 0x, np 0xFA - system szesnastkowy 0xFA=250
- Pamiętajmy, w systemie szesnastkowym A=10, B=11, C=12, D=13, E=14, F=15. Litery mogą być duże lub małe.
- Literały zmiennoprzecinkowe: $8e2 = 8 \cdot 10^2$, $10.4e8 = 10.4 \cdot 10^8$ itp.

Literały

- W poniższym przykładzie powstaną 2 zmienne typu double (`pole` i `prom`), oraz dwa literały typu `float` (1.7 i 3.14). Powstanę też dwa literały typu string (napis): „Pole koła o promieniu” oraz „ wynosi ”. Pamiętajmy, to wszystko będzie częścią programu, wpłynie na jego rozmiar i miejsce zajmowane w pamięci w czasie wykonania

```
double pole, prom = 1.7;
pole = prom * prom * 3.14;
cout << "\nPole koła o promieniu " << prom
     << " wynosi " << pole;
```

Literały znakowe

- Stałe znakowe (literały typu `char`): 'A', 'ń'. Pojedyncze znaki zapisujemy przy pomocy pojedynczych cudzysłowów. Wiele stałych znakowych niedrukowalnych ma specjalne nazwy. Często stosowane są:

  '\n' - nowa linia,
  '\0' - znak NULL,
  '\t' - tabulator poziomy,
  '\r' - powrót na początek linii.

- Literały typu `string` (napis): "Hello world", "A". Napisy zapisujemy przy pomocy podwójnego cudzysłowu. Literały "A"i 'A' są różne. Pierwszy jest napisem długości 2 bajtów (bo każdy napis ma na końcu dodany znak NULL), natomiast drugi jest typu `char` i ma długość jednego bajta. Pojedyncze znaki zawsze zapisujemy w pojedynczym cudzysłowiu

Operatory

- Dwuargumentowe operatory arytmetyczne zapisujemy jako +, -, *, /. Dodatkowo często używany jest operator modulo: %. Na przykład:

```
cout << 12 % 5 << endl;
```

  zwróci 2. Priorytet operatorów jest zwykły: %, potem *, / potem +, -. Dodatkowo od lewej do prawej. Warto jednak stosować nawiasy, nie polegać tylko na priorytetach
- Operatory w C++ mogą być *przeładowane*. Oznacza to, że mogą zachowywać się różnie w zależności od tego, z jakim zmiennymi są wywoływane. Do zagadnienia przeładowania funkcji i operatorów wrócimy w przyszłości. Teraz zwróćmy tylko uwagę, że jeżeli operator dzielenia / zastosujemy do zmiennych typu całkowitego, to rezultat też będzie całkowity. Część ułamkowa zostanie odrzucona.

Operatory

- Na przykład

```
cout << 12 / 5 << endl;
```

  zwróci 2. Nawet jeżeli wynik zostanie przypisany zmiennej zmiennoprzecinkowej, na przykład

```
double x;
x = 12 / 5;
cout << x << endl;
```

  też zwróci 2. Trzeba na to uważać.

- Jeżeli chcemy, żeby wynik był zmiennoprzecinkowy, co najmniej jeden z argumentów musi taki być. Na przykład

```
x = 12. / 5;
x = 12 / 5.d;
```

Operatory

- Dobrym zwyczajem jest zostawianie spacji po obu stronach operatora. Zwiększa to czytelność. Operator - można stosować jako jednoargumentowy, bez lewego składnika: `i = -n`. W tym przypadku nie dajemy spacji.
- Mamy jednoargumentowe operatory inkrementacji i dekrementacji: `i++`, `i--`. Taki operator zwiększa lub zmniejsza zmienną o jedną jednostkę. Jaka to jest jednostka, to zależy od typu zmiennej. Dla zmiennych całkowitych tą jednostką jest 1. W przyszłości poznamy inne jednostki, na przykład dla wskaźników.

Operatory

- Operatory te występują w wersji pre- i post-: `++i`, `i++`. Różnica leży w momencie inkrementacji (dekrementacji). W pierwszym przypadku zmienna jest zwiększona, po czym podstawiona do wzoru. W drugim przypadku najpierw jest podstawiona do wzoru, a potem zwiększona.

```
int a, b=1 ,c=1;
a = ++b;
cout << a << endl;
a=c++;
cout << a << endl;
cout << b << c << endl;
```

Po wykonaniu powyższych instrukcji zmienne b i c będą miały wartość 2. Ale zmienna a w pierwszym wypadku będzie miała wartość 2, a w drugim wypadku wartość 1.

Operatory

- Operator przypisania =. To jest operator dwuargumentowy, oblicza wartość po prawej stronie i przypisuje zmiennej po lewej stronie.
- Jest jeden szczegół związany z operatorem przypisania, na który trzeba uważać. Przypisanie samo w sobie też ma swoją wartość, i tą wartością jest wartość przypisywana. Na przykład

```
double x = 1.5;
cout << ( x = 5 ) << endl;
```

zwróci 5. Oczywiście samo `x` też po tej instrukcji będzie miało wartość 5. Najczęstszym problemem z tym związanym jest taki błąd:

```
if ( x = 5 ) instrukcja;
```

Błąd polega na tym, że w wyrażeniu warunkowym zamiast operatora logicznego == napisaliśmy podstawienie =.

Operatory

- Wbrew pozorom to jest częsty błąd, szczególnie, jeżeli piszemy programy w różnych językach. Błędu nie wychwyci kompilator, i taki błąd trudno będzie znaleźć. W nawiasie jest wyrażenie, które ma wartość 5 i jest interpretowane jako `true` (dowolna wartość niezerowa jest interpretowana jako logiczne `true`). `instrukcja` będzie więc zawsze wykonywana, niezależnie od wejściowej wartości zmiennej `x`.
- Omawiając operator dzielenia / widzieliśmy, że może się zdarzyć, że obliczona z prawej strony wartość jest innego typu niż zmienna po lewe stronie. Jeżeli typy są ze sobą zgodne (to znaczy wartość po prawej stronie w jakikolwiek sposób można zrozumieć w kontekście typu zmiennej po lewej), to nastąpi tak zwane niejawne rzutowanie. Wartość zostanie przypisana i nie zostanie zgłoszony żaden błąd.

Operatory

- Na przykład

```
double x;
x = 5;
```

  x będzie miało wartość 5 i będzie typu `double`.

- Podobnie

```
int x;
x = 5.6;
```

  x będzie miało wartość 5 i będzie typu `int`. Zauważmy, że część ułamkowa została zgubiona.

- Jeszcze jeden przykład:

```
unsigned int x;
x = -5;
```

  x będzie typu `unsigned int` i będzie miało wartość 4294967291.

Operatory

- Nie jest to żadna niespodzianka, jeżeli pamiętamy, jak zapisywane są liczby całkowite ujemne. $2^{32} - 5 = 4294967291$. Niejawnego rzutowania należy unikać. Jest częstym źródłem błędów, które trudno znaleźć. Jeżeli potrzebujemy zmienić typ wartości przy podstawieniu, używamy rzutowania jawnego.

```
unsigned int x;
x = (unsigned int) -5;
```

  Rezultat jest ten sam, ale cała operacja jest świadoma i łatwo ją zauważyć, czytając kod. Wrócimy do tego tematu w przyszłości.

Zadanie domowe (do oddania 5.04.)

Napisz program który prosi o podanie liczby naturalnej n, a następnie zwraca pierwszą i ostatnią cyfrę. Dopracuj menu programu, czyli czy użytkownik chce kontynuować, czy zakończyć. Program (podobnie jak projekty) proszę przesyłać prowadzącym ćwiczenia.