Rozdział 2

Operatory, funkcje, przeładowanie nazw funkcji

2.1 Operatory arytmetyczne

Są to operatory do wyrażeń matematycznych. Oto podstawowe operatory arytmetyczne:

1. operator dodawania +
2. operator odejmowania -
3. operator mnożenia *
4. operator dzielenia /

W przypadku operatora dzielenia `/` warto pamiętać, że gdy wykonamy dzielenie dwóch liczb całkowitych `a` i `b` to domyślnie zostanie wykonane tzw. dzielenie całkowite. Oznacza to, że wynikiem będzie liczba całkowita równa ile razy zmienna `b` mieści się całkowicie w zmiennej `a`. Aby temu zapobiec musielibyśmy zadeklarować te zmienne jako `double`.

Przykład

Napisz program, w którym istnieje zmienna `int a = 11`. Sprawdź jaki wynik dają te dwa rodzaje zapisu dzielenia: 1) `1/a`, 2) `1./a`.

2.1.1 Operator modulo

Operator modulo `%` zwraca jako wynik resztę z dzielenia. Np. $20\%6 = 2$. Operator ten stosujemy tylko dla liczb całkowitych.

Przykład

Napisz program wypisujący liczbę tysięcy, setek, dziesiątek i jedności z liczby 2136.

2.1.2 Operatory inkrementacji i dekrementacji

Są to operatory odpowiednio zwiększania i zmniejszania o 1. Operator inkrementacji zapisujemy jako `++` natomiast dekrementacji `--`. Często używamy je w pętlach. Np. żeby zwiększyć z każdym obrotem pętli zmienną `i` o jeden możemy napisać `i = i + 1` lub `i++`.

Operatory inkrementacji i dekrementacji mogą być napisane:

- przed zmienną np. `--a`. Wtedy zmienna `a` jest najpierw zmniejszana o 1, a potem używana w programie z tą nową zwiększoną wartością.
- za zmienną np. `a--`. Wtedy najpierw w programie używana jest stara wartość zmiennej `a`, a zmniejszenie następuje dopiero po wykonaniu wszystkich instrukcji.

Listing 2.1 pokazuje przykładowe użycie operatora `++`.

```
for(int i=0; i < 10; i++)
{
  cout << i << endl;
}
```

Listing 2.1: Przykład użycia operatora ++ w pętli for

2.1.3 Operatory logiczne

Wynikiem pracy operatorów logicznych jest wyrażenie typu `bool`. Oto lista operatorów logicznych z opisem ich znaczenia:

1. `<` oznacza mniejszy niż
2. `<=` oznacza mniejszy lub równy
3. `>` oznacza większy niż
4. `>=` oznacza większy lub równy
5. `==` oznacza jest równy
6. `!=` oznacza jest różny od
7. `||` oznacza sumę logiczną, czyli operację logiczną LUB
8. `||` oznacza iloczyn logiczny, czyli operację logiczną I

9. `!` oznacza operator negacji; ma on postać `!(wyrazenie)`

Przykład

Napisz program pobierający od użytkownika dwie liczby, a następnie sprawdza czy obie są równe liczbie 11. Jeśli tak to wyświetlany jest napis `Podano dwukrotnie liczbe 11.` Jeśli warunek nie jest spełniony to program ma sprawdzić czy któraś z tych liczb jest ujemna. Jeśli tak to ma wypisać napis: `Jedna z liczb jest ujemna.`, jeśli nie to `Obie liczby są dodatnie.`

2.1.4 Operatory przypisania

W języku C++ wprowadzono kilka dodatkowych operatorów przypisania (oprócz standardowego `=`) w celu skrócenia zapisu kodu. Poniżej częściej wykorzystywane przypadki:

- `a += 3` oznacza `a = a + 3`
- `a -= 3` oznacza `a = a - 3`
- `a *= 3` oznacza `a = a*3`
- `a /= 3` oznacza `a = a/3`
- `a %= 3` oznacza `a = a%3`

Należy pamiętać, że w tych operatorach znak `=` występuje na drugim miejscu.

2.1.5 Operator uzyskiwania adresu &

Do uzyskiwania adresu danej zmiennej wykorzystujemy operator `&`, który stawiamy przed zmienną. Np. `&a`.

Przykład

Napisz program wyświetlający wartość adresu danej zmiennej.

2.1.6 Wyrażenie warunkowe

Wyrażenie to ma formę:

```
warunek ? wyrazenie_tak : wyrazenie_nie
```

Listing 2.2: Forma wyrażenia warunkowego

Np. wyrażenie `(a == b) ? 100 : 20` ma wartość 100 jeśli jest spełniony warunek `(a == b)` natomiast w przeciwnym przypadku ma wartość 20.

Przykład

Napisz program wpisujący do zmiennej wartość 1000, jeśli użytkownik poda liczbę dodatnią, a w przeciwnym przypadku wartość 0. Wykorzystaj wyrażenie warunkowe w celu skrócenia kodu.

2.2 Funkcje

2.2.1 Schemat funkcji

Funkcje są to jakby małe podprogramy służące do wykonywania pewnych zadań, np. do obliczania sinusa kąta. Funkcje wywołuje się wpisując jej nazwę oraz argumenty jakie ona przyjmuje, które umieszczamy w nawiasach. Ogólny schemat funkcji podany jest poniżej w Listingu 2.3.

```
wartosc_zwracana nazwa_funkcji(argument1, argument2, ...)
{
  // cialo funkcji, czyli instrukcje jakie wykonuje
  ...
  return rezultat;
}
```

Listing 2.3: Schemat funkcji w C++

`wartosc_zwracana` oznacza jakiego typu zmienną zwraca funkcja. Jeśli funkcja ma nic nie zwracać to w tym miejscu wpisujemy słówko `void`. Wartością zwracaną może być np. `double`. `nazwa_funkcji` to nazwa jaką nadajemy naszej funkcji. Między nawiasami okrągłymi znajdują się tzw. argumenty, czyli zmienne jakie są wysyłane do naszej funkcji. Funkcja może nie przyjmować żadnych argumentów i wtedy po prostu wpisujemy same nawiasy `nazwa_funkcji()`. W ciele funkcji wykonywane są instrukcje, które zaprogramujemy. Jeśli funkcja zwraca jakiś obiekt (nie jest typu `void`) to musimy na końcu postawić słówko `return` i po nim wyrażenie bądź zmienną jakie ma zwracać funkcja. Jeśli nasza funkcja jako `wartosc_zwracana` ma wpisane `double` to po słowie `return` powinna być zmienna tego właśnie typu. W Listingu 2.4 pokazana jest przykładowa funkcja przyjmująca jako argument promień koła i zwracająca jako wynik pole koła.

```

// deklaracja funkcji
double poleKola(double promien);

int main()
{
  ...
  cout << "Podaj promien kola: " << endl;
```

```
9   double r;
10  cin >> r;
11  cout << "Pole kola wynosi: " << poleKola(r) << endl;
12 }
13 // definicja funkcji
14 double poleKola(double promien)
15 {
16  return 3.14*promien*promien;
17 }
```

Listing 2.4: Przykładowa funkcja w C++ obliczająca pole koła i wywołanie w funkcji main

Przykład

Napisz program obliczający objętość prostopadłościanu. Wywołaj tą funkcję w funkcji main.

2.2.2 Przesyłanie argumentów do funkcji

Mamy dwa podstawowe sposoby przesyłania argumentów do funkcji:

1. Przez wartość - argumenty przesłane do funkcji są kopiowane wewnątrz funkcji i jakiekolwiek działania na nich nie zmieniają oryginałów. Domyślnie przesyłanie jest przez wartość.

2. Przez referencję - referencja to inna nazwa tej samej zmiennej. Jeśli przesyłamy argumenty przez referencję to wewnątrz funkcji pracujemy na oryginałach i jeśli w funkcji zmienimy wysłany do niej argument to zmienimy oryginał. W celu przesłania przez referencję przed danym argumentem stawia się znaczek `&`. Np. `double fun(double &arg)`

W Listingu 2.5 pokazano przykład przesyłania argumentów przez wartość i przez referencję. Przesyłanie przez referencje stosuje się gdy obiekt wysyłany do funkcji jest duży (zajmuje dużo pamięci). Wtedy przy przesyłaniu przez wartość wewnątrz funkcji ten obiekt musiałby być skopiowany co przy dużych rozmiarach mogłoby spowalniać program. Efekt ten potęgowałby się gdyby funkcja była wysyłana wielokrotnie. W takich sytuacjach lepiej przesyłać przez referencję. Trzeba jednak uważać, bo możemy przypadkowo zmienić zmienną przesyłaną, a nie zawsze jest to pożądane. Jeśli jesteśmy pewni, że przesyłany przez referencję obiekt nie powinien być zmieniany możemy to zagwarantować przez dodanie słowa `const` w deklaracji funkcji np. `double fun(const double &arg)`.

```
1
2 // deklaracje funkcji
3 void zmienPrzezWartosc(int liczba);
4 void zmienPrzezReferencje(int &liczba);
5
6 int main()
7 {
```

```
8   int l = 10;
9   cout << "Zmienna l przed zmiana przez wartosc: " << l << endl;
10  zmienPrzezWartosc(l);
11  cout << "Zmienna l po zmianie przez wartosc: " << l << endl;
12  zmienPrzezReferencje(l);
13  cout << "Zmienna l po zmianie przez referencje: " << l << endl;
14 }
15 // definicje funkcji
16 void zmienPrzezWartosc(int liczba)
17 {
18   liczba += 100;
19 }
20 void zmienPrzezReferencje(int &liczba)
21 {
22   liczba += 100;
23 }
```

Listing 2.5: Przesyłanie argumentów przez wartość oraz przez referencję

2.2.3 Argumenty domniemane

Jeśli nasza funkcja przyjmuje jakieś argumenty to w programie musimy ją wywołać z tymi argumentami. Może się jednak zdarzyć, że wartości tych argumentów często się powtarzają, czyli np. często wywołujemy funkcję z argumentem równym 10, a tylko czasami z jakąś inną wartością. Możemy wtedy zastosować tzw. argumenty domniemane i naszą funkcję wywoływać prościej bez podawania za każdym razem wartości tego argumentu. Wtedy kompilator uzna, że ma uruchomić funkcję z wartością domyślną argumentu. Listing 2.6 pokazuje przykład użycia argumentów domniemanych.

```
1
2 // deklaracja funkcji
3 void fun(double &liczba, int x = 0, int y = 0);
4
5 int main()
6 {
7   double l;
8   fun(l, 10, 20);
9   fun(l);  // domniemane wywolanie fun(l,0,0)
10 }
11 // definicja funkcji
12 void fun(double &liczba, int x, int y)
13 {
14   liczba = liczba + x - y;
15 }
```

Listing 2.6: Przykład zastosowania arguemntów domniemanych funkcji

2.2.4 Przeładowanie nazw funkcji

W języku C++ nie może być dwóch tych samych nazw, np. zmiennych. Istnieje jednak mechanizm pozwalający na definiowanie wielu funkcji o tej samej nazwie.

Funkcje te muszą się wtedy różnić od siebie liczbą lub typem argumentów. Jest to tzw. przeciążanie nazw funkcji. Może się to przydać w sytuacjach gdy dana funkcja powinna inaczej się wykonywać w zależności od przyjmowanych argumentów. Nie trzeba wtedy tworzyć dodatkowych nazw dla tych różnych funkcji. Np. funkcja dodająca dwie liczby typu `int` będzie działała nieco inaczej dla liczb typu `double` dlatego trzeba napisać dwie wersje tej funkcji. Dzięki możliwości przeciążania nie trzeba wymyślać nowej nazwy np. `dodaj_int` i `dodaj_double`, ale wpisać dwie funkcje o tej samej nazwie, a różniące się typem zwracanym i typem argumentów `double dodaj(double a, double b)` oraz `int dodaj(int a, int b)`. Przy wywołaniu funkcji kompilator zorientuje się, którą wersję funkcji `dodaj` wybrać po rodzaju argumentów z jakimi ją wywołamy.

Przykład

Napisz program, w którym wywoływana jest funkcja o nazwie `pole` obliczająca pole koła, prostokąta i trapezu. Wykorzystaj przeładowanie nazw funkcji.

2.3 Zadania do rozwiązania

Zadania na ocenę 3,0

Zad. 1. Napisz program, który potrafi odszukać miejsca zerowe dowolnej prostej zadanej w postaci kierunkowej $y = ax+b$ (przy założeniu $a \neq 0$). Program powinien odczytywać a i b i na tej podstawie odszukiwać miejsce zerowe.

Zad. 2. Napisz program, który: a) pobiera dwie liczby całkowite a i b; b) wypisuje ile razy liczba a mieści się (całkowicie) w b; c) w kolejnym wierszu wypisuje resztę z dzielenia b przez a; d) w kolejnym wierszu wypisuje wartość rzeczywistą a/b.

Zad. 3. Napisz program, który pobiera od użytkownika liczbę całkowitą i wyświetla kolejno cyfrę setek, dziesiątek i jedności.

Zadania na ocenę 3,5-4,0

Zad. 4. Napisz program, który odczytuje dwie liczby (np. oceny z kolokwium) i oblicza ich średnią. Jeśli średnia jest mniejsza niż 3,0, program wypisuje komunikat „brak zaliczenia”, w przeciwnym przypadku program wypisuje „przedmiot zaliczony”. Jeśli średnia przekracza 4,0, to dopisywane jest dodatkowo „ i to na wysokim poziomie”.

Zad. 5. Napisz program, który pobiera od użytkownika liczbę z zakresu $0 \div 9$. Jeśli użytkownik poda liczbę 2, na ekranie wypisany jest napis „Kod poprawny. Masz dostęp do tajnych danych”. W przeciwnym przypadku program wypisuje

informację „Zly kod”.

Zad. 6. Napisz program, który rozwiązuje równanie $ax + b = 0$, czyli pobiera od użytkownika a i b (rzeczywiste) oraz wypisuje informacje o rozwiązaniach.

Zad. 7. Napisz program, który pobiera od użytkownika trzy liczby rzeczywiste i wypisuje największą z nich.

Zadania na ocenę 4,5-5,0

Zad. 8. Napisz prosty kalkulator. Program umożliwia podanie znaku, który oznacza wykonywane później operacje. Ustala się następujące znaki: „+”, „-”, „*”, „/”, oznaczające kolejno dodawanie, odejmowanie, mnożenie i dzielenie. W przypadku podania wymienionych znaków program umożliwia wprowadzenie dwu liczb rzeczywistych, wykonuje na nich odpowiednie działanie i wyświetla wynik. Jeśli zostanie podany inny znak, program wypisuje „nieznana operacja”.

Zad. 9. Napisz program, który pobiera trzy długości odcinków (dodatnie liczby rzeczywiste) i sprawdza, czy z tych odcinków można zbudować trójkąt, czyli suma długości dwu boków musi być w każdym przypadku większa od długości trzeciego.

Zad. 10. Napisz program, który sprawdza czy podany rok jest przestępny. Rok przestępny, to taki, który dzieli się przez 4, ale nie dzieli się przez 100 - chyba, że dzieli się przez 400. Po wprowadzeniu roku program informuje czy jest on przestępny czy nie.

Zadania nieobowiązkowe

Zad. 11. Napisz program wywołujący funkcję, która oblicza wartość cos(x). Argument x powinien być wczytywany z klawiatury. Do obliczeń wykorzystać szereg nieskończony opisany poniższym równaniem:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} \dots \tag{2.1}$$