

# Test wiedzy w wersji C++ (z odpowiedziami)

XVI OIJ, zawody I stopnia, tura testowa  
6 października 2021



Poniżej znajdują się rozwiązania zadań pierwszego podejścia tury testowej zawodów I stopnia XVI Olimpiady Informatycznej Juniorów (oij.edu.pl).

1. Która instrukcja służy do sprawdzenia warunku i (jednokrotnego) wykonania instrukcji zależnie od tego czy ten warunek jest spełniony?

- condition
- if
- check
- true
- while

2. W jaki sposób zapisać w zmiennej  $x$  długość napisu zapisanego w zmiennej  $y$  typu `string`?

- `x == y->size;`
- `x <- length(y);`
- `x = y.size();`
- `x := y->length;`
- `set(x, len(y));`

**Rozwiązanie:**

Pozostałe instrukcje zawierają co najmniej dwa błędy, w tym co najmniej jeden błąd składniowy (powodujący błąd kompilacji).

3. W jaki sposób obliczyć wynik działania  $a^3$ ?

- `a ^ 3`
- `exp(a, 3)`
- `a * a * a`
- `a * 3`

**Rozwiązanie:**

Operacja `exp` przyjmuje tylko jeden argument i oblicza  $e^x$ . Operator `^` nie oblicza potęgi, a XOR (alternatywę rozłączną) dwóch argumentów.



4. Rozważmy następujący fragment programu:

```
vector<int> elems;
...
int p = 0;
for (int i = 1; i < elems.size(); i++)
    if (elems[i] > elems[p])
        p = i;
```

Co zawiera zmienna `p` po wykonaniu powyższego kodu?

- wartość największego elementu z `elems`
- liczbę elementów `elems`
- liczbę elementów `elems` większych niż `elems[0]`
- pozycję pierwszego największego elementu z `elems`

**Rozwiązanie:**

Elementy `elems` są przeglądane po kolei, a zmienna `p` utrzymuje indeks największego dotychczas przeanalizowanego elementu z `elems` i jest podmieniana na `i`, gdy `elems[i] > elems[p]` czyli gdy przeanalizowany zostanie element, który podwyższa maksimum.

5. Które z poniższych fragmentów obliczają 33 procent liczby zapisanej w zmiennej `x` typu `double`?

- `x / 3`
- `x * 0.33`
- `x % 33`
- `x / 100 * 33`
- `33 % x`
- `x * 33%`

**Rozwiązanie:**

Operator `%` służy do obliczania reszty z dzielenia (a nie procenta z danej wartości). Jeżeli zmienna `x` jest typu `double` to podzielenie przez 100 nie spowoduje zaokrąglenia w dół do najbliższej liczby całkowitej (tak jak stałoby się w przypadku zmiennej typu `int`). Pomnożenie przez 0.33 również jest metodą obliczenia 33% liczby. Podzielenie przez 3 nie jest dokładnie obliczeniem 33 procent ( $\frac{1}{3}$  to  $33\frac{1}{3}\%$ ).

6. Rozważmy poniższą funkcję:

```
int oblicz(int n) {
    int wynik = 0;
    for (int i = 1; i <= n; i++)
        wynik += n / i;
    return wynik;
}
```

Jaki wynik zwróci wywołanie `oblicz(25)`?

**Rozwiązanie:**

Powyższa funkcja oblicza wartość

$$\lfloor \frac{n}{1} \rfloor + \lfloor \frac{n}{2} \rfloor + \lfloor \frac{n}{3} \rfloor + \dots + \lfloor \frac{n}{n} \rfloor$$

Dla  $n = 25$ , kilka pierwszych składników tej sumy to odpowiednio: 25, 12, 8, 6, 5, 4. Można teraz policzyć ile składników jest równych 3, 2 oraz 1:

- wartość 3 jest dla składników  $\lfloor \frac{25}{7} \rfloor$  oraz  $\lfloor \frac{25}{8} \rfloor$ ,
- wartość 2 jest dla składników  $\lfloor \frac{25}{9} \rfloor$ ,  $\lfloor \frac{25}{10} \rfloor$ , ...,  $\lfloor \frac{25}{12} \rfloor$ ,
- pozostałe składniki (dla mianowników 13, 14, ..., 25 są równe 1).

A zatem suma, która będzie obliczona przez funkcję to:  $25 + 12 + 8 + 6 + 5 + 4 + 3 \cdot 2 + 2 \cdot 4 + 1 \cdot 13 = 87$ .

7. Ile gwiazdek wypisze poniższy fragment programu?

```
for (int i = 0; i <= 5; i++)
    for (int j = 0; j <= 5; j++)
        cout << '*';
```

**Rozwiązanie:**

Wewnętrzna pętla wypisuje 6 gwiazdek (dla  $j \in \{0, 1, 2, 3, 4, 5\}$ ). Zewnętrzna pętla wykonuje 6 iteracji (dla  $i \in \{0, 1, 2, 3, 4, 5\}$ ) czyli zostanie wypisane  $6 \cdot 6 = 36$  gwiazdek.

8. Rozważmy poniższą funkcję:

```
string funkcja(const string& napis) {
    string wynik;
    for (int i = 1; i < napis.size(); i += 2)
        wynik.push_back(napis[i]);
    return wynik;
}
```

Jaki będzie wynik wywołania funkcja("olimpiada")?

**Rozwiązanie:**

Powyższa funkcja dodaje kolejne znaki napisu do wyniku, zaczynając od pozycji 1 (przypomnijmy, że pozycje numerowane są od lewej poczynając od 0) i przeskakując co 2 (a więc do wyniku zostaną dodane znaki na pozycjach 1, 3, 5 oraz 7, a więc l, m, i oraz d).

9. Rozważmy poniższą funkcję:

```
int funkcja(int a, int b) {  
    int wynik = 0;  
    while (b > 0) {  
        wynik += a;  
        b--;  
    }  
    return wynik;  
}
```

Zakładając, że zmienne  $a$  oraz  $b$  są całkowite, dodatnie i nie przekraczają 100, który z poniższych kodów jest równoważny wywołaniu funkcji  $funkcja(a, b)$ ?

- $a / b$
- $a - b$
- $a + b$
- $a * b$

**Rozwiązanie:**

Powyższy kod oblicza  $\underbrace{a + a + a + \dots + a}_b = a \cdot b$ .

10. Ile równa jest najmniejsza wspólna wielokrotność liczb  $1, 2, 3, \dots, 10$ ?

2520

**Rozwiązanie:**

Rozważmy rozkłady na czynniki pierwsze liczb od 1 do 10: wśród nich są liczby  $9 = 3^2$ ,  $8 = 2^3$ , 7 oraz 5. Pozostałe liczby zawierają tylko podzbiory tych czynników. A zatem liczba  $9 \cdot 8 \cdot 7 \cdot 5 = 2520$  jest najmniejszą wielokrotnością liczb  $1, 2, 3, \dots, 10$ .

11. Rozważmy poniższą funkcję:

```
int rad(int n) {
    int wynik = 1;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            wynik *= i;
            while (n % i == 0)
                n /= i;
        }
    }
    if (n > 1)
        wynik *= n;
    return wynik;
}
```

Ile jest liczb naturalnych  $n$  mniejszych niż 255, dla których  $\text{rad}(n) == 10$ ?

**Rozwiązanie:**

Powyższa funkcja oblicza *radykał* liczby  $n$ , tzn. iloczyn jej wszystkich czynników pierwszych w rozkładzie, przy czym każdy czynnik liczony jest tylko raz nawet jeśli występuje w większej potędze w liczbie  $n$ . Aby radykał był równy 10 liczba musi być więc wielokrotnością liczby 10 i nie posiadać innych czynników w rozkładzie niż 2 oraz 5. Są to liczby: 10, 20, 40, 50, 80, 100, 160, 200, 250.

12. Rozważmy poniższą funkcję:

```
int js(int n) {
    int wartosc = 1;
    for (int i = 1; ; i++) {
        wartosc *= i;
        if (wartosc > n)
            return i - 1;
    }
}
```

Jaka jest największa wartość parametru  $n$ , dla której  $\text{js}(n) == 4$ ?

**Rozwiązanie:**

Zmienna *wartosc* w powyższej funkcji w kolejnych iteracjach pętli przyjmuje wartość  $i!$ . Pętla przerywana jest, gdy wartość silni przekroczy  $n$  i zwracana jest wartość  $i - 1$ . Aby zatem funkcja zwróciła 4, pętla musi się przerwać dla zmiennej *wartosc* równej  $5! = 120$ . Największa wartość  $n$ , dla której tak jest jest równa 119.

Funkcja zwraca największą liczbę  $k$ , dla której  $k! \leq n$ .

13. Rozważmy poniższą funkcję:

```
int mx(const vector<int>& tab, int x) {
    int wynik = 0;
    for (int y : tab)
        wynik = max(wynik, x ^ y);
    return wynik;
}
```

Niech  $tab = \text{vector<int>}(\{11, 10, 24, 26\})$ . Jaka najmniejsza wartość może zwrócić wywołanie  $mx(tab, x)$  dla odpowiednio dobranej nieujemnej całkowitej wartości  $x$ ?

**Rozwiązanie:**

Powyższa funkcja dla ustalonego  $tab$  zwraca maksimum z  $11 \oplus x$ ,  $10 \oplus x$ ,  $24 \oplus x$ ,  $26 \oplus x$ , gdzie  $\oplus$  to operacja alternatywy rozłącznej (XOR).

Rozważmy zapisy liczb 11, 10, 24 oraz 26 w systemie dwójkowym. Są to:  $1011_2$ ,  $1010_2$ ,  $11000_2$ ,  $11010_2$ . Niezależnie od tego czy w liczbie  $x$  na piątej od końca pozycji bitowej będzie jedynka czy zero, w wyniku na tej pozycji będzie jedynka (obliczymy alternatywę rozłączną  $x$  z jakąś liczbą, która ma na tej pozycji bitowej zero oraz z jakąś liczbą, która ma tam jedynkę). Wynik więc jest równy co najmniej 16. Możliwe jest uzyskanie zera na drugiej, trzeciej i czwartej od końca pozycji bitowej jeżeli  $x = 10$  lub  $x = 11$ . Jednak niezależnie od tego wyboru na ostatniej pozycji bitowej uzyskamy wtedy jedynkę. A zatem wynikiem zadania jest 17 (osiągnane dla  $x \in \{10, 11\}$ ).

14. Rozważmy poniższą funkcję:

```
bool ws(const vector<int>& tab, int p, int x) {
    if (p >= tab.size())
        return false;
    if (tab[p] == x)
        return true;
    return ws(tab, p + 1, x);
}
```

Dla jakich zawartości  $tab$  wywołanie  $ws(tab, 2, 7)$  zwróci true?

- $\text{vector<int>}(\{3, 7, 1, 7, 5, 2\})$
- $\text{vector<int>}(\{4, 9, 4, 1\})$
- $\text{vector<int>}(\{1, 2, 7\})$
- $\text{vector<int>}(\{5, 8, 0, 2, 7\})$
- $\text{vector<int>}(\{2\})$
- $\text{vector<int>}(\{7, 1, 2, 3, 4\})$

**Rozwiązanie:**

Powyższa funkcja rekurencyjnie poszukuje  $x$  w  $tab$  zaczynając od pozycji  $p$  i szukając w prawo. A zatem należy zaznaczyć odpowiedzi, w których liczba 7 występuje chociaż raz na pozycji 2 lub późniejszej (przypominamy, że indeksacja elementów jest od 0).

15. Celem poniższej funkcji jest zmodyfikować `tab`, aby odwrócić kolejność jej elementów. Jak można uzupełnić warunek w `...`, aby to osiągnąć?

```
void odwracaj(vector<int>& tab) {  
    int n = tab.size();  
    for (int i = 0; ...; i++)  
        swap(tab[i], tab[n - 1 - i]);  
}
```

- `i < n`  
 `i < n / 2`  
 `i <= n / 2`  
 `i < n - 1 - i`  
 `i <= n`

**Rozwiązanie:**

Funkcja zamienia miejscami pary elementów pierwszy z ostatnim, drugi z przedostatnim itd. Jeżeli warunek byłby `i < n` to funkcja każdą parę elementów zamieniłaby dwukrotnie (czyli w efekcie każdy element byłby na swoim miejscu). Jeżeli warunek byłby `i <= n` to w funkcji odwołałbyśmy się do elementu na pozycji `n` lub `-1` (poza tablicą). Gdyby warunkiem było `i <= n / 2` to dla parzystej długości funkcja zamieniłaby środkowe elementy dwukrotnie.

Warunki `i < n / 2` oraz `i < n - 1 - i` są równoważne i ich dodanie powoduje, że niezależnie od parzystości długości tablicy, odpowiednie pary elementów zamieniane są dokładnie raz.

16. Ile zer ma liczba  $20^{50} \cdot 50^{20} \cdot 90^{90}$  na końcu zapisu dziesiętnego?

180

**Rozwiązanie:**

$$\begin{aligned} 20^{50} \cdot 50^{20} \cdot 90^{90} &= (2 \cdot 10)^{50} \cdot (5 \cdot 10)^{20} \cdot (3^2 \cdot 10)^{90} \\ &= 2^{50} \cdot 5^{20} \cdot 3^{180} \cdot 10^{50} \cdot 10^{20} \cdot 10^{90} \\ &= (2 \cdot 5)^{20} \cdot 2^{30} \cdot 3^{180} \cdot 10^{160} \\ &= 2^{30} \cdot 3^{180} \cdot 10^{180} \end{aligned}$$

Powyższa liczba jest podzielna przez  $10^{180}$ , ale nie jest podzielna przez  $10^{181}$ , a zatem kończy się dokładnie 180 zerami w zapisie dziesiętnym.

17. Poprawnym nawiasowaniem nazywamy napis, który może powstać z wyrażenia arytmetycznego przez opuszczenie wszystkiego poza znakami nawiasów. Na przykład napis  $()(())$  jest poprawnym nawiasowaniem, mógł powstać na przykład z wyrażenia  $(2 + 2) * (1 + (2 + 3) * 4)$ . Celem poniższej funkcji jest zbadać czy przekazany napis jest poprawnym nawiasowaniem.

```
bool czy_poprawne_nawiasowanie(const string& naw) {
    int min_balans = 0;
    int balans = 0;
    for (char x : naw) {
        if (x == '(')
            balans++;
        else if (x == ')')
            balans--;
        else
            return false;
        min_balans = min(min_balans, balans);
    }
    return ...;
}
```

Jak należy uzupełnić ..., aby to osiągnąć?

- balans == 0
- (min\_balans < balans) && (balans >= 0)
- min\_balans == 0
- (min\_balans >= 0) && (balans == 0)
- min\_balans <= balans

**Rozwiązanie:**

Balans nawiasów (różnica między liczbą nawiasów otwartych a zamkniętych) musi być równy 0, dodatkowo w żadnym momencie balans nie może być ujemny (dla każdego początkowego fragmentu nawiasowania liczba nawiasów otwierających musi być większa lub równa liczbie nawiasów zamykających).

18. Jaka jest najmniejsza liczba naturalna  $n$ , której zapis w systemie rzymskim ma dokładnie 10 znaków? Odpowiedź podaj w systemie dziesiętkowym.

**Rozwiązanie:**

Ustalmy najpierw długość liczby  $n$  w cyfrach arabskich: każda cyfra arabska zapisywana jest w systemie rzymskim z użyciem co najwyżej czterech znaków. A zatem  $n$  musi być co najmniej trzycyfrowa ( $\lfloor \frac{10}{4} \rfloor = 3$ ), przy czym pierwsza cyfra musi być zapisana z użyciem co najmniej dwóch znaków w systemie rzymskim (reszta z dzielenia 10 przez 4 to 2). A zatem cyfra setek szukanej liczby to 2, a pozostałe dwie cyfry muszą być zapisane z użyciem dokładnie czterech znaków rzymskich. Tak dzieje się tylko dla cyfry 8, a zatem  $n = 288$ .



19. Ile zmiennych typu `string` przechowujących napisy długości 1 000 000 znaków można zadeklarować w programie, aby zużycie pamięci (związane z przechowywaniem tych zmiennych) wynosiło około 50 MB?

- około pięćset
- około pięciu
- kilka tysięcy
- około pięćdziesiąt
- jedną

**Rozwiązanie:**

Pojedynczy znak napisu zajmuje bajt, a więc zużycie pamięci związane z przechowaniem miliona znaków jest co najmniej rzędu miliona bajtów czyli około jednego megabajta. Dodatkowe, stałe zużycie pamięci na pusty napis jest pomijalnie małe przy tak długim napisie. Strategia alokowania pamięci w przypadku różnych kompilatorów może być różna, ale najpopularniejsze kompilatory nie powinny domyślnie zaalokować więcej niż niewiele ponad milion bajtów. A zatem możliwe jest utworzenie około pięćdziesięciu napisów tej długości.

Zużycie pamięci w C++ można sprawdzić z użyciem `sizeof`.

20. Rozważmy poniższy fragment programu:

```
for (int i = 1; i <= n; i++)
  for (int j = 1; j * j <= n; j++)
    cout << '*';
```

Jaką liczbę gwiazdek wypisaną przez program można osiągnąć, odpowiednio dobierając  $n$ ?

- 400
- 512
- 350
- 256
- 150

**Rozwiązanie:**

Program wypisuje  $n \lfloor \sqrt{n} \rfloor$  gwiazdek. Pytanie zatem sprowadza się do tego, czy istnieje naturalne rozwiązanie równania  $n \cdot \lfloor \sqrt{n} \rfloor = M$  (dla  $M \in \{150, 256, 350, 400, 512\}$ ). Lewa strona równości jest równa w przybliżeniu  $n\sqrt{n}$ , możliwe jest więc zgadnięcie ile powinno wynosić  $n$  i sprawdzenie czy jest ono całkowite.

Dane zostały dobrane tak, żeby chociaż część obliczeń była łatwa do wykonania w pamięci:  $256 = 2^8$  nie jest rozwiązaniem (bo  $2^{16/3}$  nie jest całkowite), ale  $512 = 2^9$  jest (bo  $2^{18/3} = 2^6$  jest całkowite).



21. Rozważmy poniższą funkcję:

```
void wykonuj(int n) {
    for (int i = 1; i <= n; i++) {
        int j = i;
        while (j % 2 == 0) {
            j /= 2;
            cout << "*";
        }
    }
}
```

Zakładamy, że operacje arytmetyczne oraz porównania na zmiennych typu `int` zajmują czas stały. Jaka jest pesymistyczna złożoność obliczeniowa funkcji `wykonuj`?

- $\Theta(n \log n)$
- $\Theta(n)$
- $\Theta(n^2)$
- $\Theta(\sqrt{n})$
- $\Theta(\log n)$
- $\Theta(n\sqrt{n})$

**Rozwiązanie:**

Na pierwszy rzut oka widać, że złożoność powyższej funkcji jest  $O(n \log n)$  (jest  $O(\log n)$  obrotów pętli wewnętrznej dla każdego obrotu pętli zewnętrznej). Jednak można zauważyć, że tylko co drugi obrót pętli `for` powoduje co najmniej jeden obrót pętli `while`, tylko co czwarty obrót pętli `for` powoduje co najmniej dwa obroty pętli `while` itd. A zatem łącznie wypisanych gwiazdek (i jednocześnie operacji elementarnych) jest rzędu  $n + \frac{n}{2} + \frac{n}{4} + \dots = \Theta(n)$ .

Powyższy kod jest analogią do prostego licznika binarnego.

22. Które z poniższych zbiorów liczb można podzielić na dwa zbiory o równej sumie? Każdy element powinien trafić do dokładnie jednego zbioru.

- {2, 3, 4, 5, 6, 7, 8}
- {1, 2, 3, 5, 8, 13}
- {1, 2, 3, 4, 5, 6, 7}
- {1, 2, 4, 8, 16, 32, 65}
- {2, 3, 5, 7, 11}

**Rozwiązanie:**

Suma elementów w zbiorze {2, 3, 4, 5, 6, 7, 8} jest nieparzysta, więc na pewno nie jest możliwe dokonanie żądanego podziału.

Zbiór {1, 2, 3, 5, 8, 13} to ciąg sześciu kolejnych liczb Fibonacciego, można więc przypisać pierwszą, drugą, czwartą i piątą liczbę do jednego zbioru a trzecią i szóstą do drugiego zbioru otrzymując  $1 + 2 + 5 + 8 = 3 + 13$ .

Zbiór {1, 2, 3, 4, 5, 6, 7} można podzielić np. tak:  $7 + 6 + 1 = 2 + 3 + 4 + 5$  (jest wiele sensownych metod podziału).

Zbiór {1, 2, 4, 8, 16, 32, 65} nie jest możliwy do podzielenia ponieważ suma elementów innych niż 65 jest równa 63.

Zbiór {2, 3, 5, 7, 11} można podzielić np. tak:  $3 + 11 = 2 + 5 + 7$ .

23. Która z podanych liczb jest największa?

- $FF_{16}$
- 200
- $1111111_2$
- $A0_{11}$

**Rozwiązanie:**

$FF_{16} = 255$ ,  $1111111_2 = 127$ , a  $A0_{11} = 110$ .

24. Ile jest całkowitych nieujemnych kwot, których nie można wydać nominałami 7 oraz 11?

30

**Rozwiązanie:**

Zauważmy, że jeśli możliwe jest wydanie kwoty  $x$  to jest również możliwe wydanie kwoty  $x + 7$ . Dla każdej reszty z dzielenia kwoty przez 7 mamy zatem granicę poniżej, której nie jest możliwe wydanie żadnej kwoty o tej reszcie z dzielenia, a po osiągnięciu której jest już to możliwe. Aby tę granicę wyznaczyć, wystarczy dokładać kolejne nominały 11.

I tak:

- jest możliwe wydanie kwot 0, 7, 14, 21, ... ,
- nie jest możliwe wydanie kwoty 4, ale jest możliwe wydanie kwot 11, 18, 25, ... ,
- nie jest możliwe wydanie kwot 1, 8, 15, ale jest możliwe wydanie kwot 22, 29, 36, ... ,
- nie jest możliwe wydanie kwot 5, 12, 19, 26, ale jest możliwe wydanie kwot 33, 40, 47, ... ,
- nie jest możliwe wydanie kwot 2, 9, 16, 23, 30, 37, ale jest możliwe wydanie kwot 44, 51, 58, ... ,
- nie jest możliwe wydanie kwot 6, 13, 20, 27, 34, 41, 48, ale jest możliwe wydanie kwot 55, 62, 69, ... ,
- nie jest możliwe wydanie kwot 3, 10, 17, 24, 31, 38, 45, 52, 59, ale jest możliwe wydanie kwot 66, 73, 80, ... .

25. Dla jakich wartości parametru  $n$  poniższa funkcja poprawnie kończy swoje działanie (tzn. nie powoduje błędu wykonania)?

```
void wykonuj(int n) {  
    if (n == 100)  
        return;  
    wykonuj(n + 2);  
}
```

- 1000
- 10
- 100
- 1

**Rozwiązanie:**

Funkcja kończy swoje działanie dla wartości  $n = 100$  oraz wszystkich wartości parzystych mniejszych od 100.

26. Ile tablic rozmiaru 1 000 000 składających się z liczb naturalnych można posortować z użyciem funkcji `std::sort` na przeciętnym komputerze z roku 2020, aby czas wykonania był równy około pół sekundy? Zakładamy, że sortowania wykonywane są jedno po drugim bez współbieżności.
- kilka tysięcy
  - nie można wykonać nawet jednego
  - kilka
  - kilkaset

**Rozwiązanie:**

Funkcja `sort` działa w czasie  $\Theta(n \log n)$  czyli można założyć, że potrzebuje około 20 milionów porównań i ewentualnych przestawień elementów do posortowania jednej tablicy. Rozsądnym jest założenie (przy obecnej szybkości przeciętnych procesorów i optymalizacjach w kompilatorach C++), że 100 milionów porównań i przestawień zajęłoby około jednej sekundy. Stąd można przyjąć, że jedno posortowanie tablicy kosztuje dziesiątą części sekundy.

27. 60 dzieci ustawiło się w kółku i zaczęło odliczać do dwóch (tzn. pierwsze dziecko mówi 1, drugie dziecko mówi 2, trzecie 1, czwarte 2, piąte 1 itd.). Każde dziecko, które mówi 2 natychmiast wypada z kółka, a wyliczanka jest dalej kontynuowana. Po pełnym okrążeniu przeprowadzana jest analogiczna wyliczanka tylko do trzech (tzn. dzieci liczą 1, 2, 3 i każde dziecko, które mówi 3 wypada). W kolejnych okrążeniach następują analogiczne wyliczanki, ale do 4, 5, 6, ... W pewnym momencie wyliczanka jest do większej liczby niż liczba dzieci i nikt już dalej nie odpadnie. Ile dzieci pozostanie w kółku?

8

**Rozwiązanie:**

Jeśli przed  $i$ -tym odliczaniem było  $n$  dzieci, to po  $i$ -tym odliczaniu jest  $\lceil n \cdot \frac{i}{i+1} \rceil$  dzieci (odpada  $\lfloor n \cdot \frac{1}{i} \rfloor$  dzieci). A zatem, w kolejnych odliczaniach pozostaje: 30, 20, 15, 12, 10, 9, 8 dzieci. W kolejnych odliczaniach nie odpada już żadne dziecko.

28. Rozważmy następującą funkcję:

```
int oblicz(const string& operacje) {
    int wynik = 0;
    for (char x : operacje) {
        if (x == '+')
            wynik++;
        if (x == '*')
            wynik *= 2;
    }
    return wynik;
}
```

Podaj napis złożony z czterech znaków + oraz czterech znaków \*, który należy przekazać jako parametr operacje, aby `oblicz(operacje) == 35`?

++\*\*\*+\*

**Rozwiązanie:**

Celem zadania jest uzyskanie liczby 35 z liczby 0 w skutek procesu dodawania jedynki lub mnożenia przez 2. Jest to możliwe w ośmiu operacjach jedynie w następujący sposób:

$0 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 17 \rightarrow 34 \rightarrow 35$

29. Rozważmy poniższą funkcję:

```
int oblicz(int n, int k) {
    int wyniki[n + 1][k + 1];
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= k; j++)
            wyniki[i][j] = 0;

    for (int i = 0; i <= n; i++) {
        wyniki[i][0] = 1;
        for (int j = 1; j <= min(k, i); j++) {
            wyniki[i][j] += wyniki[i - 1][j];
            wyniki[i][j] += wyniki[i - 1][j - 1];
        }
    }
    return wyniki[n][k];
}
```

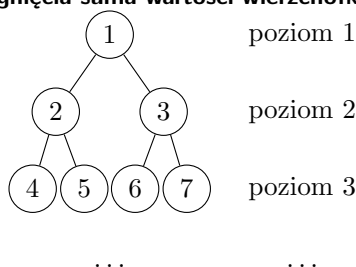
Jaki jest wynik wywołania `oblicz(10, 5)`?

**Rozwiązanie:**

Przedstawiony kod powyżej oblicza trójkąt Pascala, zawierający w kolejnych wierszach kolejne symbole Newtona  $\binom{n}{k}$  korzystając z tożsamości  $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ . A zatem zwrócony wynik to:

$$\binom{10}{5} = \frac{10!}{5! \cdot 5!} = \frac{6 \cdot 7 \cdot 8 \cdot 9 \cdot 10}{2 \cdot 3 \cdot 4 \cdot 5} = 7 \cdot 2 \cdot 9 \cdot 2 = 252$$

30. Na poniższym rysunku wierzchołki są na trzech poziomach, każdy wierzchołek na poziomie innym niż ostatni jest połączony z dwoma na poziomie kolejnym. Wierzchołki są numerowane od 1 kolejno poziomami, a na każdym poziomie od lewej do prawej. Rozważmy podobny rysunek, ale o sześciu poziomach. Ile wynosi największa możliwa do osiągnięcia suma wartości wierzchołków na ciągu kolejno połączonych wierzchołków bez zawracania?



Fragment pełnego drzewa binarnego (do trzech poziomów).

**Rozwiązanie:**

Najlepiej jest aby ścieżka była od skrajnie prawego dolnego wierzchołka, poruszała się w górę drzewa do pewnego wierzchołka i potem, po jednym skręcie do lewego syna biegła już tylko w prawo i w dół do innego skrajnie dolnego wierzchołka. Wierzchołków w drzewie jest  $2^6 - 1 = 63$ , taki jest więc numer wierzchołka startowego. Rodzicem wierzchołka  $x$  jest wierzchołek  $\lfloor \frac{x}{2} \rfloor$ , lewym synem wierzchołka  $x$  jest wierzchołek  $2x$ , a prawym synem wierzchołka  $x$  jest wierzchołek  $2x + 1$ .

Optymalna ścieżka przebiega kolejno po wierzchołkach:

$$63 \rightarrow 31 \rightarrow 15 \rightarrow 7 \rightarrow 3 \rightarrow 6 \rightarrow 13 \rightarrow 27 \rightarrow 55$$

