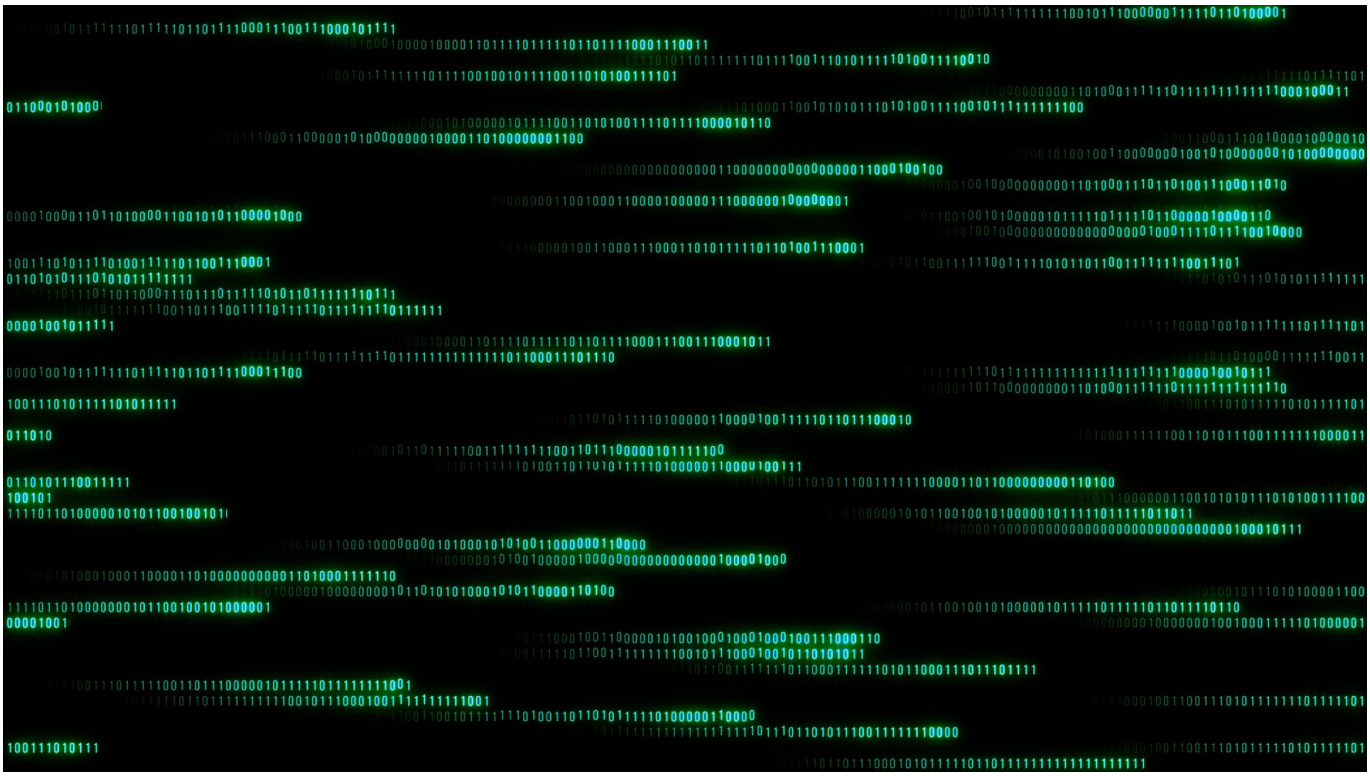
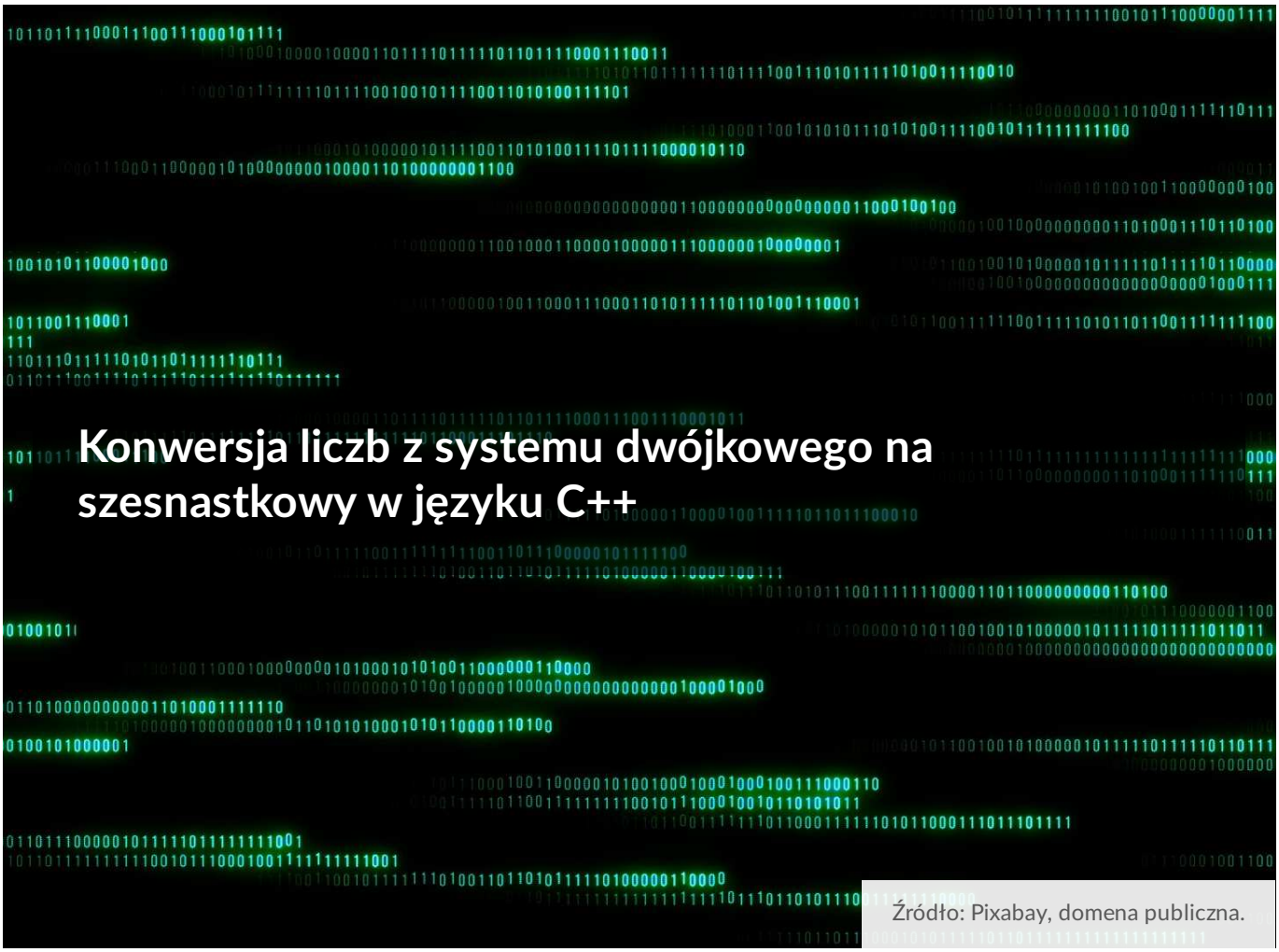


Konwersja liczb z systemu dwójkowego na szesnastkowy w języku C++



Źródło: Pixabay, domena publiczna.

> W tym e-materiale powtarzamy wiadomości ze szkoły podstawowej.

System dwójkowy ma z punktu widzenia człowieka pewną wadę: przedstawione w nim liczby są niekiedy bardzo długie. Z tego właśnie powodu w pewnych sytuacjach – takich jak choćby podawanie adresów komórek pamięci – wygodniej jest posługiwać się systemem szesnastkowym. Pozwala on czterokrotnie zmniejszyć długość zapisu liczby (w porównaniu z systemem binarnym).

Umiejętność przedstawienia tej samej liczby w systemach o różnych podstawach okazuje się bardzo przydatna. W tym e-materiale skupimy się na konwersji liczb dwójkowych do ich odpowiedników w systemie szesnastkowym. Napiszemy także program w języku C++, który realizuje takie zadanie.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych lekcjach z tej serii:

- Konwersja liczb z systemu dwójkowego na szesnastkowy w języku Java,
- Konwersja liczb z systemu dwójkowego na szesnastkowy w języku Python.

Więcej zadań? Sięgnij do: Konwersja liczb z systemu dwójkowego na szesnastkowy – zadania maturalne.

> **Twoje cele**
>
> - Scharakteryzujesz poznane systemy liczbowe.
> - Poznasz algorytm konwersji liczb dwójkowych do postaci szesnastkowej i zapiszesz go w postaci programu w języku C++.
> - Przeanalizujesz algorytm konwersji części ułamkowej liczby binarnej do postaci szesnastkowej.

Przeczytaj

I metoda konwersji bin → hex

Zanim napiszemy program konwertujący liczbę dwójkową do systemu o podstawie 16 (szesnastkowego), omówimy niezbędne do wykonania czynności. Całą operację przeprowadzimy dwuetapowo: najpierw liczbę dwójkową przedstawimy w systemie dziesiętnym. Następnie zamienimy ją na odpowiednik w systemie szesnastkowym.

Metodę konwersji pokażemy na przykładzie. Ułatwi nam to zrozumienie komputerowej realizacji algorytmu.

> **Ważne!**
>
> Pamiętaj, że w systemie o podstawie 16 oprócz cyfr od 0 do 9 wykorzystywane są symbole literowe z zakresu od A do F. Oto odpowiadające im wartości dziesiętne:

Symbol	Wartość
A	10
B	11
C	12
D	13
E	14
F	15

> **Przykład 1**
>
> Zapiszemy liczbę $10100111_{(2)}$ w postaci szesnastkowej.

$$10100111_{(2)} = 1 \cdot 2^7 + 1 \cdot 2^5 +$$

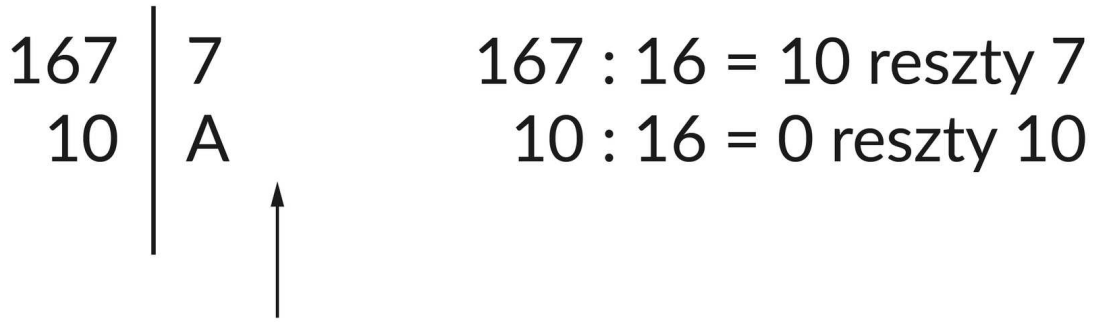
$$+1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 167_{(10)}$$

Najpierw mnożymy wszystkie cyfry liczby dwójkowej przez odpowiadające im wagi, czyli potęgi liczby 2. Po zsumowaniu iloczynów otrzymujemy zapis liczby w postaci dziesiętnej.

Oto wynik pierwszej konwersji:

$$10100111_{(2)} = 167_{(10)}$$

Następnie przedstawimy liczbę dziesiętną w postaci szesnastkowej.



Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

W tym celu wykonujemy następujące czynności:

1. Dzielimy liczbę przez 16 (czyli przez podstawę systemu szesnastkowego).
2. Zapamiętujemy iloraz oraz resztę z dzielenia.
3. Wracamy do punktu 1. Dzielimy przez 16 obliczony wcześniej iloraz. Czynności opisane w punktach 1. i 2. wykonujemy dopóty, dopóki iloraz jest większy niż zero.
4. Rezultatem konwersji są reszty z dzielenia, zapisane w kolejności od ostatniej do pierwszej.

Po zakończeniu całej operacji otrzymujemy wynik:

> $$10100111_{(2)} = 167_{(10)} = \mathrm{A7}_{(16)}$$

Realizacja algorytmu w języku C++

Możemy już przejść do napisania programu, który zamieni liczbę w systemie dwójkowym na system szesnastkowy.

Zaczniemy od napisania funkcji odpowiadającej za konwersję liczby binarnej do systemu dziesiętnego. Nazwiemy ją `konwertujBinDec()`. Jako parametr funkcja przyjmie liczbę zapisaną w systemie dwójkowym. Zwróć uwagę, że podajemy ją w postaci ciągu znaków `liczbaBin` (jest to istotne, ponieważ jej elementy są traktowane jak znaki kodu ASCII):

```
int konwertujBinDec(std::string liczbaBin) {
    int liczbaDec = 0;
    int bit = 0;
    int waga = 1; // ponieważ 2^0 = 1

    for (int i = liczbaBin.length() - 1; i >= 0; i--) {
        bit = liczbaBin[i] - '0';
        liczbaDec = liczbaDec + bit * waga;
        waga *= 2;
    }

    return liczbaDec;
}
```

Omówmy poszczególne linie kodu.

Na początku deklarujemy trzy zmienne typu całkowitego i nadajemy im wartości początkowe. Będą to:

- `bit = 0`, którą wykorzystamy do przechowywania *wartość* aktualnie rozpatrywanego bitu

- `liczbaDec = 0`, która będzie przechowywała nasz rezultat konwersji
- `waga = 1`, która będzie przechowywała *wagę* obecnie rozpatrywanego bitu

Następnie tworzymy pętlę `for`, w której będziemy kolejno przechodzili po znakach naszej liczby binarnej od końca, tj. zaczynając od bitu o najmniejszej wadze. W każdym cyklu pętli `for` wykonamy następujące operacje:

- Zmiennej `bit` przypisujemy wartość `liczbaBin[i] - '0'`. Innymi słowy, nadajemy jej wartość kodu ASCII znaku zapisanego w tablicy `liczbaBin[]`, pomniejszoną o kod ASCII znaku '0'. W rezultacie `bit` przybiera wartość 0 lub 1. Wynika to z faktu, że zmienna `liczbaBin[i]` jest znakiem '0' lub '1'; odpowiadające im wartości w kodzie ASCII wynoszą – kolejno – 48 i 49. Kiedy `liczbaBin[i]` będzie znakiem '1', zostanie wykonane działanie 49-48, a `bit` przyjmie wartość 1. Jeżeli `liczbaBin[i]` to znak '0', po operacji odejmowania (48-48) `bit` wyniesie 0.W każdym cyklu pętli `for` mnożymy ją przez podstawę systemu binarnego, czyli 2.
- Do zmiennej `liczbaDec` przypisujemy wartość `liczbaDec + bit*waga`, czyli obliczony do tej pory wynik powiększymy o `waga` jeżeli `bit` będzie równy 1 – w przeciwnym razie wartość `liczbaDec` pozostanie bez zmian.
- Zmienną `waga` pomnożymy przez podstawę systemu liczbowego, czyli 2

Na koniec zwracamy wynik konwersji, czyli liczbę zapisaną w systemie dziesiętnym. Korzystamy z instrukcji `return liczbaDec`.

> **Już wiesz**
>
> W każdym cyklu pętli `for` podwajamy wartość zmiennej `waga` za pomocą operatora przypisania `*=`.
>
> Ten sam efekt uzyskalibyśmy, wydając polecenie:
>
> ```
> 1 waga = waga * 2;
> ```

> **Ważne!**

> Ponieważ korzystamy z funkcji `std::to_string()`, musimy dołączyć do programu odpowiednią bibliotekę:
>
> ```
> #include <string>
> ```

Pierwsza funkcja jest zatem gotowa. Pora zamienić liczbę dziesiętną na szesnastkową.

Posłużymy się funkcją o nazwie `konwertujDecHex()`. Jej jedynym parametrem będzie liczba dziesiętna, otrzymana w wyniku działania funkcji `konwertujBinDec()`:

```
std::string konwertujDecHex(int liczbaDec) {
    if (liczbaDec == 0) {
        return "0";
    }

    std::string liczbaHex = "";
    int reszta;

    while (liczbaDec > 0) {
        if (liczbaDec % 16 >= 0 && liczbaDec % 16 <= 9) {
            reszta = liczbaDec % 16;
            liczbaHex = std::to_string(reszta) + liczbaHex;
        } else {
            reszta = liczbaDec % 16 - 10 + 'A';
            char znak = reszta;
            liczbaHex = znak + liczbaHex;
        }

        liczbaDec = liczbaDec / 16;
    }

    return liczbaHex;
}
```

Na samym początku należy sprawdzić, czy podany do funkcji argument jest zerem. Jeżeli tak, funkcja od razu powinna zwrócić ciąg znaków „0".

Kolejnym krokiem jest deklaracja zmiennych:

- `std::string liczbaHex` – jest to tablica znaków, w której zapiszemy wynik konwersji.
- `int reszta` – będziemy w niej przechowywać reszty z dzielenia przez 16.

Następnie otwieramy pętlę `while`. Dzięki zapisanym w niej instrukcjom sprawdzimy, czy reszta z dzielenia konwertowanej liczby mieści się w przedziale <0, 9>.

- Jeżeli tak jest, przypisujemy zmiennej `reszta` wynik operacji `liczbaDec % 16`. Następnie, korzystając z funkcji `std::to_string()`, zamieniamy wartość całkowitą na znakową i dopisujemy („doklejamy") ją do zmiennej `liczbaHex`.
- Jeżeli reszta z dzielenia jest większa niż 9, w zmiennej `reszta` zapisujemy wynik operacji `liczbaDec % 16 - 10 + 'A'`. Odjęcie liczby 10, a następnie dodanie wartości odpowiadającej w tablicy ASCII literze 'A' spowoduje, że zmienna `reszta` przybierze wartości odpowiadające pozycjom znaków od 'A' do 'F' w tablicy ASCII. Przykładowo, jeżeli wynik działania `liczbaDec % 16` będzie równy 14 (w notacji szesnastkowej odpowiada mu znak 'E'), to zostanie wykonana instrukcja `reszta = 14 - 10 + 65`. Jej wynikiem jest 69, czyli liczbowy odpowiednik znaku 'E' w tablicy ASCII. Przypisując zmiennej `znak` wartość 69 otrzymujemy literę 'E'. Następnie dopisujemy ją do zmiennej `liczbaHex`.

Musimy pamiętać, aby w każdym cyklu pętli podzielić iloraz `liczbaDec` przez podstawę systemu szesnastkowego. W innym przypadku pętla będzie działać w nieskończoność:

```
1 liczbaDec = liczbaDec / 16;
```

Na końcu zwracamy wynik konwersji, czyli zmienną `liczbaHex`.

Niezbędne funkcje mamy już napisane.

Dodajmy jeszcze instrukcje, dzięki którym użytkownik poda liczbę do przekształcenia. Odpowiedni kod umieszczamy w funkcji głównej `int main()`:

```
std::string liczbaTest;

std::cout << "Podaj liczbę w systemie binarnym, która ma zostać p
std::cin >> liczbaTest;
```

Ostatnim etapem jest wywołanie funkcji dla liczby podanej jako argument. Zwróć uwagę, że musimy zdefiniować zmienną `liczbaDec`, w której zapiszemy wynik pierwszego etapu konwersji (bin → dec). Wynika to z faktu, że zastosowany algorytm nie dokonuje bezpośredniej konwersji między systemami bin → hex.

```
int liczbaDec = konwertujBinDec(liczbaTest);

std::cout << konwertujDecHex(liczbaDec);
```

Oto cały kod programu:

```
#include <iostream>
#include <string>

int konwertujBinDec(std::string liczbaBin) {
    int liczbaDec = 0;
    int bit = 0;
    int waga = 1; // dlatego że 2 ^ 0 = 1

    for (int i = liczbaBin.length() - 1; i >= 0; i--) {
        bit = liczbaBin[i] - '0';
        liczbaDec = liczbaDec + bit * waga;
        waga *= 2;
    }

    return liczbaDec;
```

```cpp
}

std::string konwertujDecHex(int liczbaDec) {
    if (liczbaDec == 0) {
        return "0";
    }

    std::string liczbaHex = "";
    int reszta;

    while (liczbaDec > 0) {
        if (liczbaDec % 16 >= 0 && liczbaDec % 16 <= 9) {
            reszta = liczbaDec % 16;
            liczbaHex = std::to_string(reszta) + liczbaHex;
        } else {
            reszta = liczbaDec % 16 - 10 + 'A';
            char znak = reszta;
            liczbaHex = znak + liczbaHex;
        }

        liczbaDec = liczbaDec / 16;
    }

    return liczbaHex;
}

int main() {
    std::string liczbaTest;

    std::cout << "Podaj liczbę w systemie binarnym, która ma zost
    std::cin >> liczbaTest;

    int liczbaDec = konwertujBinDec(liczbaTest);

    std::cout << konwertujDecHex(liczbaDec);

```

```
52      return 0;
53  }
```

II metoda konwersji bin → hex

Przekształcenia liczby dwójkowej w szesnastkową możemy dokonać także wykorzystując fakt, że po podniesieniu liczby 2 (czyli podstawy systemu binarnego) do czwartej potęgi uzyskamy liczbę 16 (czyli podstawę systemu szesnastkowego).

$$2^4 = 16$$

Każdy znak systemu szesnastkowego da się więc zapisać za pomocą czterech bitów.

Przeanalizujmy metodę konwersji:

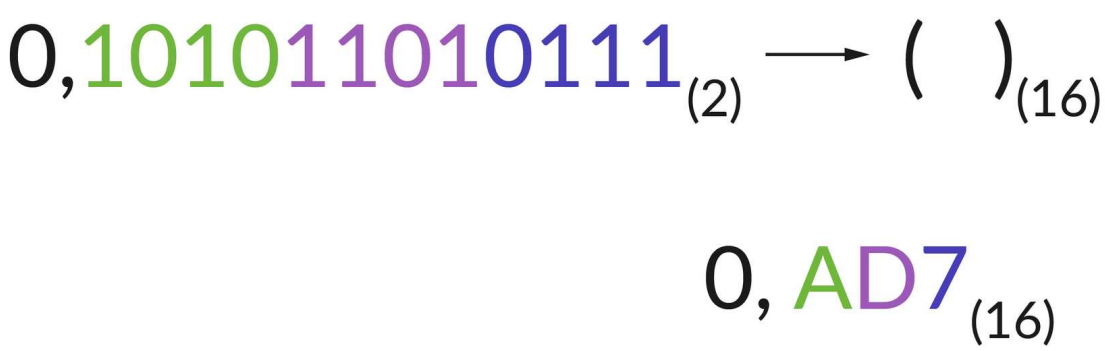
> **Przykład 2**
>
> Chcemy zapisać liczbę $11010101_{(2)}$ w postaci szesnastkowej. Wykorzystamy fakt, że oba systemy mają bazy skojarzone.
>
> $$0101_{(2)} = 5_{(16)}$$
>
> $$1101_{(2)} = \mathrm{D}_{(16)}$$
>
> A zatem otrzymujemy wynik:
>
> $$11010101_{(2)} = \mathrm{D5}_{(16)}$$

Konwersja części ułamkowej

Korzystając z właściwości baz skojarzonych możemy także konwertować część ułamkową liczby. Oto przykład:

> **Przykład 3**

Zapiszemy liczbę $0,101011010111_{(2)}$ w systemie szesnastkowym.



Konwersja liczby z systemu binarnego na system szesnastkowy

W kolejnej sekcji zaimplementujemy ten właśnie algorytm.

Słownik

bazy skojarzone

bazy systemów liczbowych, w przypadku których podstawa jednego systemu jest potęgą bazy systemu drugiego

parametr funkcji

element składni w określonym języku programowania; umożliwia komunikację pomiędzy podprogramem (funkcją) wywołanym a programem wywołującym; parametry określa się w nagłówku podprogramu (przy jego definicji)

argument funkcji

element składni w określonym języku programowania, który w wyniku wywołania podprogramu zostaje utożsamiony (skojarzony) z określonym parametrem podprogramu

Symulacja interaktywna

Polecenie 1

Przeanalizuj algorytm konwersji części ułamkowej liczby binarnej do postaci zapisanej w systemie szesnastkowym.

1

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Napiszemy program konwertujący część ułamkową liczby zapisanej w systemie binarnym do postaci szesnastkowej.

Wykorzystamy zdobytą już wiedzę oraz zagadnienia omówione w poprzedniej sekcji.

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Zaczniemy od zadeklarowania głównej funkcji programu – `main()`:

```
#include <iostream>

int main() {
    return 0;
}
```

Materiał audio dostępny pod adresem:

3

https://zpe.gov.pl/b/PRhZ2HybS

Następnie zbudujemy szkielet funkcji `konwertujBinDecUlamek()`. Zwróci ona zmienną typu `int`, czyli liczbę zapisaną w systemie dziesiętnym:

```
#include <iostream>

int
konwertujBinDecUlamek() {

}

int main() {
    return 0;
}
```

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Funkcja będzie przyjmować jeden parametr – łańcuch znaków `std::string liczbaBin`.

Pamiętajmy o dodaniu odpowiedniej biblioteki (`<string>`):

```
#include <iostream>
#include <string>

int
konwertujBinDecUlamek(std:
:string liczbaBin) {

}

int main() {
    return 0;
}
```

5

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Wykorzystamy znany ci już algorytm konwersji liczby z systemu binarnego do dziesiętnego. Omawialiśmy go w poprzedniej sekcji, pisząc równocześnie kod w języku C++:

```
#include <iostream>
#include <string>

int konwertujBinDecUlamek(std::string liczbaBin) {
    int wynik = 0;
    int waga = 1;
    int bit = 0;

    for (int i = liczbaBin.length() - 1; i >= 0; i--) {
        bit = liczbaBin[i] - '0';
        wynik = wynik + bit * waga;
        waga = waga * 2;
    }

    return wynik;
}

int main() {
    return 0;
}
```

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Następnie zadeklarujemy drugą funkcję o nazwie `konwertujBinHexUlamek()`, która będzie zamieniać część ułamkową z postaci dwójkowej do szesnastkowej. Zwróci ona łańcuch znaków `std::string`:

```
#include <iostream>
#include <string>

int konwertujBinDecUlamek(std::string liczbaBin) {
    int wynik = 0;
    int waga = 1;
    int bit = 0;

    for (int i = liczbaBin.length() - 1; i >= 0; i--) {
        bit = liczbaBin[i] - '0';
        wynik = wynik + bit * waga;
        waga = waga * 2;
    }

    return wynik;
}

std::string konwertujBinHexUlamek() {

}

int main() {
    return 0;
}
```

https://zpe.gov.pl/b/PRhZ2HybS

Parametrem funkcji `konwertujBinHexUlamek()` będzie tablica znaków `std::string liczbaBin`, czyli część ułamkowa zapisana w systemie dwójkowym. Zamierzamy ją przekształcić do postaci szesnastkowej:

```
#include <iostream>
#include <string>

int konwertujBinDecUlamek(std::string liczbaBin) {
    int wynik = 0;
    int waga = 1;
    int bit = 0;

    for (int i = liczbaBin.length() - 1; i >= 0; i--) {
        bit = liczbaBin[i] - '0';
        wynik = wynik + bit * waga;
        waga = waga * 2;
    }

    return wynik;
}

std::string konwertujBinHexUlamek(std::string liczbaBin) {

}

int main() {
    return 0;
}
```

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Zgodnie z omówionym wcześniej algorytmem, powinniśmy odczytywać 4-bitowe części ułamka, przekształcać je do postaci dziesiętnej, a następnie liczbę dziesiętną zapisywać w systemie szesnastkowym.

9

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Aby to zrobić, wykorzystamy następujące zmienne:

- `bool koniecLiczby`
- `std::string liczbaHex`
- `char hex[16]`
- `std::string bity`
- `int i`

Zadeklarujemy je wewnątrz funkcji `konwertujBinHexUlamek()`:

```
#include <iostream>
#include <string>

int
konwertujBinDecUlamek(std:
:string liczbaBin) {
    int wynik = 0;
    int waga = 1;
    int bit = 0;

    for (int i =
liczbaBin.length() - 1; i
```

```
>= 0; i--) {
10          bit = liczbaBin[i]
- '0';
11          wynik = wynik +
bit * waga;
12          waga = waga * 2;
13      }
14
15      return wynik;
16  }
17
18  std::string
konwertujBinHexUlamek(std:
:string liczbaBin) {
19      bool koniecLiczby =
false;
20      std::string liczbaHex
= "0,";
21      char hex[16] =
{'0','1','2','3','4','5','
6','7','8','9','A','B','C'
,'D','E','F'};
22      std::string bity = "";
23      int i = 2
24  }
25
26  int main() {
27      return 0;
28  }
```

10

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Następnie utworzymy pętlę `do-while`, dzięki której będziemy przesuwać się co 4 bity. Zmienna `i` jest zmienną sterującą. Jej wartość ustalamy początkowo na 2, ponieważ `liczbaBin[0]` i `liczbaBin[1]` wynoszą – odpowiednio – "0" i "1". Znaki rozpoczynające część ułamkową nie podlegają konwersji:

```cpp
#include <iostream>
#include <string>

int konwertujBinDecUlamek(std::string liczbaBin) {
    int wynik = 0;
    int waga = 1;
    int bit = 0;

    for (int i = liczbaBin.length() - 1; i >= 0; i--) {
        bit = liczbaBin[i] - '0';
        wynik = wynik + bit * waga;
        waga = waga * 2;
    }

    return wynik;
}

std::string konwertujBinHexUlamek(std::string liczbaBin) {
    bool koniecLiczby = false;
    std::string liczbaHex = "0,";
    char hex[16] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
    std::string bity = "";
    int i = 2;

    do {
        bity = "";
        i = i + 4;
    } while (koniecLiczby == false);
}

```

```
int main() {
    return 0;
}
```

Na początku każdego cyklu pętli przypisujemy zmiennej `bity` pusty ciąg. Postępujemy tak, ponieważ za każdym razem w zmiennej `bity` zapisujemy kolejne 4 znaki. Gdybyśmy nie zerowali zmiennej, po kilku powtórzeniach zawierałaby ona więcej niż 4 znaki i program nie działałby poprawnie.

Zmienna licznikowa `i` po każdym cyklu pętli zwiększa się o 4.

Pętla kończy się, gdy nie będzie już więcej cyfr w części ułamkowej, czyli gdy zmienna `koniecLiczby` przyjmie wartość `true`.

Grupowanie po cztery cyfry zaczynamy od najbardziej znaczących pozycji, a zerami uzupełniamy ułamek na końcu, czyli na pozycjach najmniej znaczących.

11

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Napiszmy pętlę `for` która będzie zapisywać 4 kolejne znaki w zmiennej `bity`:

```
#include <iostream>
#include <string>

int konwertujBinDecUlamek(std::string liczbaBin) {
    int wynik = 0;
    int waga = 1;
    int bit = 0;

    for (int i = liczbaBin.length() - 1; i
```

```
>= 0; i--) {
10          bit = liczbaBin[i]
- '0';
11          wynik = wynik +
bit * waga;
12          waga = waga * 2;
13      }
14
15      return wynik;
16  }
17
18  std::string
konwertujBinHexUlamek(std:
:string liczbaBin) {
19      bool koniecLiczby =
false;
20      std::string liczbaHex
= "0,";
21      char hex[16] =
{'0','1','2','3','4','5','
6','7','8','9','A','B','C'
,'D','E','F'};
22      std::string bity = "";
23      int i = 2;
24
25      do {
26          bity = "";
27
28          for (int j = i; j
< i + 4; j++) {
29              bity = bity +
liczbaBin[j];
30          }
31
32          i = i + 4;
33      } while (koniecLiczby
== false);
34  }
35
36  int main() {
37      return 0;
38  }
```

12

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Co zrobić w przypadku, gdy liczba bitów w części ułamkowej nie będzie wielokrotnością liczby 4? Przypisywalibyśmy wtedy zmiennej `bity` wartości z tablicy znaków `liczbaBin`, które nie istnieją. Musimy uporać się z takim problemem.

Napiszemy instrukcję warunkową, która sprawdza czy odczytywany element tablicy nie zawiera znaku pustego (znaku NULL). Jeżeli nie, to element zostanie dopisany do zmiennej `bity`. Jeżeli tak, do zmiennej `bity` zostanie dopisany znak '0', a zmienna logiczna `koniecLiczby` przybierze wartość `true`.

Dopisanie zera nie wpłynie na wynik konwersji.

Pierwsza instrukcja warunkowa sprawdza, czy pierwszy bit z kolejnej "czwórki" jest znakiem NULL. Gdy okaże się, że jest, to od razu zwracamy wynik. Dzięki temu unikamy wyświetlania dodatkowego znaku "0":

```
1  #include <iostream>
2  #include <string>
3
4  int
   konwertujBinDecUlamek(std:
   :string liczbaBin) {
5      int wynik = 0;
6      int waga = 1;
7      int bit = 0;
8
9      for (int i =
   liczbaBin.length() - 1; i
   >= 0; i--) {
10         bit = liczbaBin[i]
   - '0';
11         wynik = wynik +
   bit * waga;
```

```
        waga = waga * 2;
    }

    return wynik;
}

std::string konwertujBinHexUlamek(std::string liczbaBin) {
    bool koniecLiczby = false;
    std::string liczbaHex = "0,";
    char hex[16] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
    std::string bity = "";
    int i = 2;

    do {
        bity = "";

        for (int j = i; j < i + 4; j++) {
            if (liczbaBin[j] == NULL && i == j) {

koniecLiczby = true;

                return liczbaHex;
            } else if (liczbaBin[j] != NULL) {
                bity = bity + liczbaBin[j];
            } else {
                bity = bity + "0";

koniecLiczby = true;
            }
        }

```

```
        i = i + 4;
    } while (koniecLiczby
== false);
}

int main() {
    return 0;
}
```

13

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Gdy w zmiennej `bity` zostaną zapisane 4 znaki, należy je przekształcić do postaci dziesiętnej. Zrobimy to za pomocą wcześniej napisanej funkcji `konwertujBinDecUlamek()`. Wynik konwersji zapisujemy w zmiennej `liczbaDec`:

```
#include <iostream>
#include <string>

int
konwertujBinDecUlamek(std:
:string liczbaBin) {
    int wynik = 0;
    int waga = 1;
    int bit = 0;

    for (int i =
liczbaBin.length() - 1; i
>= 0; i--) {
        bit = liczbaBin[i]
- '0';
        wynik = wynik +
bit * waga;
        waga = waga * 2;
    }

    return wynik;
}
```

```
17
18  std::string
    konwertujBinHexUlamek(std:
    :string liczbaBin) {
19      bool koniecLiczby =
    false;
20      std::string liczbaHex
    = "0,";
21      char hex[16] =
    {'0','1','2','3','4','5','
    6','7','8','9','A','B','C'
    ,'D','E','F'};
22      std::string bity = "";
23      int i = 2;
24
25      do {
26          bity = "";
27
28          for (int j = i; j
    < i + 4; j++) {
29              if
    (liczbaBin[j] == NULL && i
    == j) {
30
    koniecLiczby = true;
31
32                  return
    liczbaHex;
33              } else if
    (liczbaBin[j] != NULL) {
34                  bity =
    bity + liczbaBin[j];
35              } else {
36                  bity =
    bity + "0";
37
    koniecLiczby = true;
38              }
39          }
40
41          int liczbaDec =
    konwertujBinDecUlamek(bity
    );
42          i = i + 4;
```

```
    } while (koniecLiczby == false);
}

int main() {
    return 0;
}
```

14

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Następnie zapisujemy do zmiennej wynikowej `liczbaHex` odpowiednik szesnastkowy przekształconej do postaci szesnastkowej zmiennej `liczbaDec`:

```
#include <iostream>
#include <string>

int konwertujBinDecUlamek(std::string liczbaBin) {
    int wynik = 0;
    int waga = 1;
    int bit = 0;

    for (int i = liczbaBin.length() - 1; i >= 0; i--) {
        bit = liczbaBin[i] - '0';
        wynik = wynik + bit * waga;
        waga = waga * 2;
    }

    return wynik;
}
```

```
std::string konwertujBinHexUlamek(std::string liczbaBin) {
    bool koniecLiczby = false;
    std::string liczbaHex = "0,";
    char hex[16] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
    std::string bity = "";
    int i = 2;

    do {
        bity = "";

        for (int j = i; j < i + 4; j++) {
            if (liczbaBin[j] == NULL && i == j) {

koniecLiczby = true;

                return liczbaHex;
            } else if (liczbaBin[j] != NULL) {
                bity = bity + liczbaBin[j];
            } else {
                bity = bity + "0";

koniecLiczby = true;
            }
        }

        int liczbaDec = konwertujBinDecUlamek(bity);
        liczbaHex = liczbaHex + hex[liczbaDec];
```

```
        i = i + 4;
    } while (koniecLiczby
== false);
}

int main() {
    return 0;
}
```

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Program jest już prawie gotowy. Do kodu dodamy jeszcze instrukcje, dzięki którym użytkownik poda część ułamkową liczby w systemie binarnym przeznaczoną do zapisania w postaci szesnastkowej. Najpierw deklarujemy zmienną `liczbaTest`, a później pobieramy ją poleceniem `cin`:

```
#include <iostream>
#include <string>

int
konwertujBinDecUlamek(std:
:string liczbaBin) {
    int wynik = 0;
    int waga = 1;
    int bit = 0;

    for (int i =
liczbaBin.length() - 1; i
>= 0; i--) {
        bit = liczbaBin[i]
- '0';
        wynik = wynik +
bit * waga;
        waga = waga * 2;
    }

```

```
    return wynik;
}

std::string konwertujBinHexUlamek(std::string liczbaBin) {
    bool koniecLiczby = false;
    std::string liczbaHex = "0,";
    char hex[16] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
    std::string bity = "";
    int i = 2;

    do {
        bity = "";

        for (int j = i; j < i + 4; j++) {
            if (liczbaBin[j] == NULL && i == j) {

koniecLiczby = true;

                return liczbaHex;
            } else if (liczbaBin[j] != NULL) {
                bity = bity + liczbaBin[j];
            } else {
                bity = bity + "0";

koniecLiczby = true;
            }
        }

        int liczbaDec = konwertujBinDecUlamek(bity);
```

```
42          liczbaHex =
liczbaHex +
hex[liczbaDec];
43          i = i + 4;
44      } while (koniecLiczby
== false);
45
46      return liczbaHex;
47  }
48
49  int main() {
50      std::string
liczbaTest;
51      std::cout << "Podaj
część ułamkową do
przekonwertowania" <<
std::endl;
52      std::cin >>
liczbaTest;
53
54      return 0;
55  }
```

Określamy również, co funkcja ma zwracać. Jest to `liczbaHex`, czyli przekształcona do systemu szesnastkowego liczba zapisana jako łańcuch znaków.

16

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PRhZ2HybS

Na koniec dodajemy wywołanie funkcji `konwertujBinHexUlamek()` dla argumentu `liczbaTest` i wyświetlamy wynik.

Oto kod programu:

```
1  #include <iostream>
2  #include <string>
3
```

```
4  int
   konwertujBinDecUlamek(std:
   :string liczbaBin) {
5      int wynik = 0;
6      int waga = 1;
7      int bit = 0;
8
9      for (int i =
   liczbaBin.length() - 1; i
   >= 0; i--) {
10         bit = liczbaBin[i]
   - '0';
11         wynik = wynik +
   bit * waga;
12         waga = waga * 2;
13     }
14
15     return wynik;
16 }
17
18 std::string
   konwertujBinHexUlamek(std:
   :string liczbaBin) {
19     bool koniecLiczby =
   false;
20     std::string liczbaHex
   = "0,";
21     char hex[16] =
   {'0','1','2','3','4','5','
   6','7','8','9','A','B','C'
   ,'D','E','F'};
22     std::string bity = "";
23     int i = 2;
24
25     do {
26         bity = "";
27
28         for (int j = i; j
   < i + 4; j++) {
29             if
   (liczbaBin[j] == NULL && i
   == j) {
30
   koniecLiczby = true;
31
```

```
32                  return
liczbaHex;
33              } else if
(liczbaBin[j] != NULL) {
34                  bity =
bity + liczbaBin[j];
35              } else {
36                  bity =
bity + "0";
37
koniecLiczby = true;
38              }
39          }
40
41          int liczbaDec =
konwertujBinDecUlamek(bity
);
42          liczbaHex =
liczbaHex +
hex[liczbaDec];
43          i = i + 4;
44      } while (koniecLiczby
== false);
45
46      return liczbaHex;
47  }
48
49  int main() {
50      std::string
liczbaTest;
51      std::cout << "Podaj
część ułamkową do
przekonwertowania" <<
std::endl;
52      std::cin >>
liczbaTest;
53      std::cout <<
konwertujBinHexUlamek(licz
baTest);
54
55      return 0;
56  }
```

Polecenie 2

Na poniższym schemacie interaktywnym przedstawiono konwersję z systemu dwójkowego na szesnastkowy. Zastanów się, w jaki sposób wyglądałaby konwersja do systemu ósemkowego.

Zasób interaktywny dostępny pod adresem https://zpe.gov.pl/a/DWA2dDXF4

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Do konwersji liczb między systemem dwójkowym a szesnastkowym możemy skorzystać z faktu, że do zapisania jednej cyfry w systemie szesnastkowym potrzebujemy dokładnie 4 bitów. Przykładowo, $1111_{(2)} = f_{(16)}$. Nic nie stoi na przeszkodzie, aby użyć tej właściwości dla liczb złożonych z większej ilości bitów. Dla przykładu: $10011100_{(2)} = 9c_{(16)}$.

Uzupełnij poniższy kod tak, aby dokonywał on konwersji liczby dodatniej całkowitej zapisanej w systemie dwójkowym na system szesnastkowy. Skorzystaj w tym celu z grupowania liczby binarnej po cztery bity. Możesz przyjąć, że długość liczby binarnej to wielokrotność liczby 4.

Twoje zadania

1. Kod przekształca dodatnią liczbę całkowitą z systemu dwójkowego na system szesnastkowy.

Ćwiczenie 2

Uzupełnij poniższy kod tak, aby dokonywał on konwersji liczby naturalnej zapisanej w systemie binarnym na system szesnastkowy. Twój kod powinien obsłużyć przypadek, w którym długość podanej liczby nie jest wielokrotnością cyfry 4.

Twoje zadania

1. Program dokonuje konwersji dowolnej liczby naturalnej zapisanej w systemie binarnym na system szesnastkowy.

Ćwiczenie 3

Korzystając z przygotowanych fragmentów kodu, napisz program, który przekształci dodatnią liczbę całkowitą zapisaną w systemie binarnym do postaci szesnastkowej. W celu sprawdzenia czy program działa poprawnie, dokonaj konwersji podanej liczby zapisanej w łańcuchu znaków `liczba_test`. Do konwersji z systemu binarny na dziesiętny użyj schematu Hornera.

Twoje zadania

1. Program powinien zapisać dowolną dodatnią liczbę całkowitą binarną w systemie szesnastkowym.

Dla nauczyciela

Autor: Maurycy Gast

Przedmiot: Informatyka

Temat: Konwersja liczb z systemu dwójkowego na szesnastkowy w języku C++

Grupa docelowa:

Liceum ogólnokształcące i technikum, liceum ogólnokształcące, technikum, zakres podstawowy i rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres podstawowy. Uczeń:

2) stosuje przy rozwiązywaniu problemów z różnych dziedzin algorytmy poznane w szkole podstawowej oraz algorytmy:

a) na liczbach: badania pierwszości liczby, zamiany reprezentacji liczb między pozycyjnymi systemami liczbowymi, działań na ułamkach z wykorzystaniem NWD i NWW,

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Scharakteryzujesz poznane systemy liczbowe.
- Poznasz algorytm konwersji liczb dwójkowych do postaci szesnastkowej i zapiszesz go w postaci programu w języku C++.
- Przeanalizujesz algorytm konwersji części ułamkowej liczby binarnej do postaci szesnastkowej.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne;
- metody aktywizujące.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka C++, w tym kompilator GCC/G++ 4.5 (lub nowszej wersji) i Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Konwersja liczb z systemu dwójkowego na szesnastkowy w języku C++". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytania dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu, np.
   – jakie litery są wykorzystywane w systemie o podstawie 16 oprócz cyfr od 0 do 9?
   – czym charakteryzują się bazy skojarzone?
   Chętni lub wybrani uczniowie udzielają na nie odpowiedzi.

Faza realizacyjna:

1. Uczniowie analizują przykład z sekcji „Przeczytaj" i powtarzają zaprezentowane rozwiązanie na swoim komputerze.
2. **Praca z multimedium.** Uczniowie zapoznają się z materiałami z sekcji „Symulacja interaktywna". Analizują algorytm konwersji części ułamkowej liczby binarnej do postaci zapisanej w systemie szesnastkowym oraz zapoznają się ze schematem

interaktywnym przedstawiającym konwersję z systemu dwójkowego na szesnastkowy. Zastanawiają się, w jaki sposób wyglądałaby konwersja do systemu ósemkowego. Wymyślają pytanie na kartkówkę związane z tematem materiału.

3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenia nr 1 i 2. Po wykonaniu każdego z nich następuje omówienie rozwiązania przez nauczyciela.

Faza podsumowująca:

1. Na koniec zajęć nauczyciel prosi uczniów o rozwinięcie zdania: „Na dzisiejszych zajęciach nauczyłam/łem się jak…”.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 3 z sekcji „Sprawdź się”.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka C++.
- Oficjalna dokumentacja techniczna dla kompilatora GCC/G++ 4.5 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Wskazówki metodyczne:

- Nauczyciel może wykorzystać multimedium w sekcji „Symulacja interaktywna” do pracy przed lekcją. Uczniowie zapoznają się z jego treścią i przygotowują do pracy na zajęciach w ten sposób, żeby móc samodzielnie rozwiązać zadania dołączone do e-materiału „Konwersja liczb z systemu dwójkowego na szesnastkowy w języku C++”.