Rozdział 5

Dynamiczna alokacja pamięci. Operacje bitowe

5.1 Dynamiczna alokacja pamięci

5.1.1 Operatory new i delete

Wcześniej poznaliśmy tablice statyczne, które cechowały się stałym rozmiarem w czasie trwania programu. Ponadto ich rozmiar musiał być znany już w momencie kompilacji, a nie dopiero gdy program zaczął działać. Niemożliwe więc było np. aby użytkownik po uruchomieniu programu podawał jaki ma być rozmiar tablicy. Obecnie dowiemy się o tzw. dynamicznej alokacji pamięci pozwalającej określać rozmiar tablicy dopiero w trakcie działania programu.

Do dynamicznej alokacji pamięci służy operator `new`. Sama rezerwacja pamięci jest dosyć prosta. Trzeba jednak pamiętać, aby w momencie gdy ta pamięć nie będzie już potrzebna (np. w momencie skończenia programu) to należy ją zwolnić operatorem `delete`. Jeśli tego nie zrobimy nawet po skończeniu programu pamięć będzie zajęta co przy kilkakrotnym uruchamianiu może doprowadzić do skończenia się pamięci RAM i zawieszenia komputera.

Pamięć możemy alokować dla pojedynczych zmiennych jak i tablic. Aby to zrobić musimy mieć wcześniej zdefiniowany wskaźnik do danego typu obiektów. Jest to spowodowane tym, że operator `new` zwraca wskaźnik, więc możemy go przypisać do wskaźnika. Przykład tworzenia zmiennych dynamicznych, a następnie zwalnianie zajętej przez nie pamięci pokazuje listing 5.1. Używając operatora `new` tworzymy obiekt, gdzieś w pamięci komputera, ale nie ma on nazwy. Możemy się do niego dostać jedynie za pomocą wskaźnika. Tworzenie a następnie usuwanie tablicy dynamicznej pokazuje listing 5.2. Zauważ, że przy operatorze `delete` jest znak tablicy `[]`. Używamy go do usuwania tablic.

```
1 double *w1 = nullptr;
2 w1 = new double;
3 int *w2 = new int;
4 *w1 = 23.45;
```

```
*w2 = 2000;
delete w1;
delete w2;
```

Listing 5.1: Przykład dynamicznej alokacji pamięci operatorem new dla pojedynczych zmiennych i następnie zwalnianie pamięci operatorem delete

```
double *w = new double[200];
*w[20] = 23.45;   // wpisanie wartosci do 21 elementu tablicy
delete [] w;
```

Listing 5.2: Przykład dynamicznej alokacji pamięci operatorem new dla tablicy i następnie zwalnianie pamięci operatorem delete [

Przykład

Napisz program, w którym tworzona jest tablica dynamiczna o 20 elementach liczb rzeczywistych. Następnie wyświetlone są elementy tablicy na ekran. W kolejnym kroku do tablicy wpisywane są jakieś liczby i znowu wyświetlane są elementy na ekran.

5.1.2 Inicjalizacja obiektu dynamicznego

Często będziemy chcieli od razu nadać wartość początkową obiektowi tworzonemu operatorem `new`. Możemy to zrobić przez dodanie nawiasów klamrowych jak pokazano w listingu 5.3.

```
double *w = new double{3.14};  // nadanie wartosci 3.14
double *W {new double{3.14}};  // nadanie wartosci 3.14
int *s = new int{};            // nadanie wartosci 0
int *s = new int{};            // nadanie wartosci 0
int *t = new int[1000]{};      // nadanie wartosci 0 kazdemu
                               // elementowi tablicy
int *T {new int[1000]{}};      // nadanie wartosci 0 kazdemu
                               // elementowi tablicy drugim
                               // sposobem
int *i = new int[15]{0,1,2,3}; // nadanie wartosci pierwszym
                               // czterem elementom tablicy
                               // a reszcie wartosci 0
```

Listing 5.3: Inicjalizacja obiektów tworzonych operatorem new

Przykład

Napisz program, który pyta użytkownika o rozmiar tablicy, tworzy taką tablicę i wczytuje do niej dane wprowadzane przez użytkownika z klawiatury.

5.2 Operacje bitowe

Oprócz zwykłych operacji matematycznych możliwe są także operacje na bitach co wiąże się z tym, że informacje w komputerze przechowywane są w systemie binarnym. Może się to przydać np. przy programowaniu różnego rodzaju sterowników. Pamiętamy, że bit to pojedyncza wartość 0 lub 1. Bity łączymy w bajty. Bajt to 8 bitów. Wyróżniamy następujące operatory bitowe:

1. `|` bitowa suma logiczna
2. `&` bitowy iloczyn logiczny
3. `^` bitowa różnica symetryczna
4. `~` bitowa negacja
5. `<<` przesunięcie w lewo
6. `>>` przesunięcie w prawo

5.2.1 Bitowa suma logiczna |

Jest to operator, który z lewej i prawej strony ma liczby całkowite. Powoduje zastosowanie sumy logicznej dla dwóch bitów. Czyli jak mamy 0 z lewej i 0 z prawej to wynik jest 0. 0 i 1 to wynik 1, 1 i 0 to wynik 1, 1 i 1 to wynik 1. Np. liczby 10 i 5 w zapisie binarnym mają postać 1010 i 0101. Stosując sumę bitową 10 | 5 otrzymamy `10 | 5 = 1111` .

5.2.2 Bitowy iloczyn logiczny &

Jest to operator, który z lewej i prawej strony ma liczby całkowite. Powoduje zastosowanie iloczynu logicznego dla dwóch bitów. Czyli jak mamy 0 z lewej i 0 z prawej to wynik jest 0. 0 i 1 to wynik 0, 1 i 0 to wynik 0, 1 i 1 to wynik 1. Np. liczby 10 i 5 w zapisie binarnym mają postać 1010 i 0101. Stosując iloczyn bitowy 10 & 5 otrzymamy `10 & 5 = 0000` .

Operator iloczynu bitowego możemy wykorzystać do sprawdzenia czy liczba jest parzysta czy nie. Wystarczy sprawdzić czy najmniej znaczący bit (bit leżący najbardziej z prawej strony) ma wartość 0 (parzysta) czy 1 (nieparzysta). Aby to zrobić należy sprawdzić czy wynik działania `(zmiennaInt & 1)` daje wartość `true` (1) czy `false` (0).

5.2.3 Bitowa różnica symetryczna

Jest to operator, który z lewej i prawej strony ma liczby całkowite. Jeśli dwa odpowiadające sobie bity są różne od siebie to wynik jest 1, jak takie same to 0. Czyli jak mamy 0 z lewej i 0 z prawej to wynik jest 0. 0 i 1 to wynik 1, 1 i 0 to wynik 1, 1 i 1 to wynik 0. Np. liczby 10 i 5 w zapisie binarnym mają postać 1010 i 0101. Stosując bitową różnicę symetryczną otrzymamy `10 ^ 5 = 1111` .

5.2.4 Bitowa negacja

Jest to operator, który z lewej nie ma nic i z prawej strony ma liczbę całkowitą (operator jednoargumentowy). Powoduje, że wszystkie bity zamieniane są na przeciwne (0 na 1, 1 na 0). Np. liczba 6 w zapisie binarnym ma postać 0110 (reszta to zera z lewej strony). Stosując negację binarną 6 otrzymamy 1001 (teraz z prawej strony pojawią się jedynki w miejsce zer).

Operator bitowej negacji w połączeniu z operatorami przesunięcia w lewo i iloczynu logicznego możemy wykorzystać do wyzerowania n-tego bitu danej zmiennej. Algorytm wygląda tak:

1. Przesuń bitowo w lewo 1 o n pozycji, czyli `1 << n`
2. Wykonaj na wyniku bitową negację. Dzięki temu n-ty bit zostaje ustawiony na 0, a reszta na 1, czyli `~(1 << n)`
3. Wykonaj bitowy iloczyn logiczny danej zmiennej i otrzymanego wyniku z poprzedniego kroku. Te trzy kroki możemy zapisać w jednej linii jako `zmienna & (~(1 << n))`

5.2.5 Przesunięcie w lewo «

Jest to operator, który z lewej i prawej strony ma liczby całkowite. Wywołujemy go w ten sposób: `a << b` . Powoduje on przesunięcie w lewo o `b` bitów każdy z bitów w `a`. Np. mając liczbę 10 zapisaną jako `0000 0000 0000 1010` to zastosowanie przesunięcia bitowego w lewo o 3 da wynik `0000 0000 0101 0000`. Jak widać nowo powstałe bity po prawej stronie zostały uzupełnione zerami. Jeśli byłyby jakieś jedynki po lewej skrajnej stronie to po przesunięciu zostałyby utracone. Operacja przesunięcia w lewo o `b` bitów odpowiada pomnożeniu przez 2^b.

Operator bitowego przesunięcia w lewo możemy wykorzystać do ustawienia wartości 1 dla n-tego bitu w danej zmiennej. Robimy to w ten sposób, że przesuwamy bitowo w lewo 1 o n pozycji i z wyniku bierzemy bitową sumę logiczną z daną zmienną. W praktyce możemy się tego dowiedzieć wyświetlając wynik komendy `(1 >> n) | zmienna` .

5.2.6 Przesunięcie w prawo »

Jest to operator, który z lewej i prawej strony ma liczby całkowite. Wywołujemy go w ten sposób: `c >> d` . Powoduje on przesunięcie w prawo o `d` bitów każdy z bitów w `a`. Np. mając liczbę 10 zapisaną jako `0000 0000 0000 1010` to zastosowanie przesunięcia bitowego w prawo o 3 da wynik `0000 0000 0000 0001`. Jak widać nowo powstałe bity po lewej stronie zostały uzupełnione zerami. Bity, które znalazły się po przesunięciu poza prawą skrajną stroną po przesunięciu zostały utracone. Operacja przesunięcia w prawo o `d` bitów odpowiada podzieleniu przez 2^d.

Operator bitowego przesunięcia w prawo możemy wykorzystać do sprawdzania jaka jest wartość n-tego bitu w danej zmiennej. Robimy to w ten sposób, że przesuwamy bitowo w prawo o n pozycji i z wyniku bierzemy bitowy iloczyn logiczny z 1. W praktyce możemy się tego dowiedzieć wyświetlając wynik komendy `(zmienna >> n) & 1` .

5.3 Zadania do rozwiązania

Zadania na ocenę 3,0

Zad. 1. Napisz program, w którym tworzone są w sposób dynamiczny: liczba 3.14, liczba 1000, napis "Imie i nazwisko", tablica o 1000 elementów wypełniona zerami. Zadbaj o dealokację pamięci.

Zad. 2. Napisz program, w którym są dwie cyfry całkowite o wartościach 14 i 544 i wyświetlane są wyniki:

1. ich sumy algebraicznej i bitowej
2. ich iloczynu algebraicznego i bitowego
3. ich bitowej różnicy symetrycznej
4. negacji logicznej (!) i bitowej każdej z nich
5. przesunięcia w prawo i w lewo każdej z nich

Zadania na ocenę 3,5-4,0

Zad. 3. Napisz program, który dynamicznie zaalokuje tablicę typu int i wypełni ją kolejnymi liczbami, a na koniec zwolni pamięć.

Zad. 4. Napisz program, w którym jest zdefiniowana liczba całkowita i wykonywane jest:

1. mnożenie przez 128 zapisane za pomocą operacji na bitach
2. dzielenie przez 1024 zapisane za pomocą operacji na bitach

Wyniki wyświetl na ekran i sprawdź ich poprawność przez dodanie analogicznych instrukcji zapisanymi operatorami algebraicznymi.

Zadania na ocenę 4,5-5,0

Zad. 5. Napisz funkcję, która tworzy tablicę liczb całkowitych o zadanej liczbie elementów. Liczba elementów powinna być przyjmowana przez funkcję jako

argument. Następnie napisz program, w którym tworzone są 2 tablice za pomocą tej funkcji.

Zad. 6. Napisz program, który przekształca podaną przez użytkownika liczbę całkowitą na liczbę zapisaną w systemie binarnym i wyświetla wynik. Do tego celu wykorzystaj operacje bitowe.

Zadania nieobowiązkowe

Zad. 7. Korzystając z operacji bitowych napisz program, który:

1. zwraca n-ty bit liczby,
2. sprawdza, czy liczba jest parzysta,
3. ustawia n-ty bit liczby na 1,
4. ustawia n-ty bit liczby na 0.

Zad. 8. Napisz program tworzący dwie tablice o rozmiarze podawanym przez użytkownika podczas pracy programu. Rozmiar tablic powinien być duży, np. rzędu 10^4 lub większy. Następnie tablice są inicjalizowane kolejnymi liczbami naturalnymi od wartości 1. W programie powinny być dwie funkcje przyjmujące jako parametr tablicę i jej rozmiar. Pierwsza funkcja wykonuje przemnożenie każdego elementu tablicy przez 128 w sposób algebraiczny, a druga w sposób bitowy. Czy jest jakaś zauważalna różnica w czasie wykonania tych funkcji? (Aby dokładnie to sprawdzić możesz spróbować sposobów pokazanych w internecie, np. [4]).

Bibliografia

[1] cplusplus.com. Variables and types. https://www.cplusplus.com/doc/tutorial/variables/.

[2] cpp0x.pl/ Serwis programistyczny C++. Biblioteka <math.h>. https://cpp0x.pl/kursy/Kurs-C++/Dodatkowe-materialy/Biblioteka-math-h/322.

[3] cpp0x.pl/ Serwis programistyczny C++. Pseudolosowe liczby całkowite. https://cpp0x.pl/kursy/Kurs-C++/Poziom-2/Pseudolosowe-liczby-calkowite/290.

[4] GeeksforGeeks. Measure execution time of a function in c++. https://www.geeksforgeeks.org/measure-execution-time-function-cpp/.

[5] J. Grębosz. *Opus magnum C++11. Programowanie w języku C++*. Helion, 2019.