Podstawy Programowania 1

Typy wyliczeniowe i jednowymiarowe tablice

Arkadiusz Chrobot

Katedra Systemów Informatycznych

16 listopada 2020

Plan

1. Abstrakcja danych
2. Typy wyliczeniowe
3. Słowo kluczowe `typedef`
4. Tablice jednowymiarowe
5. Inicjacja tablicy
6. Operacje na tablicach jednowymiarowych

Abstrakcja danych

Abstrakcja może dotyczyć nie tylko operacji (instrukcji) wykonywanych w programach, ale również danych. Język C pozwala programiście na tworzenie własnych typów danych, które są dostosowane do problemu przez niego rozwiązywanego. Przykładem takiego typu danych są typy wyliczeniowe.

Typy wyliczeniowe

Typ wyliczeniowy pozwala nadawać nazwy liczbom należącym do określonego podzbioru liczb całkowitych, a także znakom. Innymi słowy, możemy myśleć o typie wyliczeniowym jako o zbiorze stałych. Ogólny wzorzec definicji typy wyliczeniowego można zapisać następująco:

```
enum nazwa_typu {ELEMENT_1=wartość, …, ELEMENT_N};
```

Jak można zauważyć, nazwy (identyfikatory) poszczególnych elementów typu wyliczeniowego są pisane wielkimi literami, tak jak stałe. Jest to jednak kwestia wyłącznie konwencji, można zastosować każdą pisownię zgodną z regułami dotyczącymi tworzenia identyfikatorów w języku C. Każdemu z elementów wolno nam przypisać dowolną liczbę całkowitą typu `int`. Jeśli opuścimy wszystkie przypisania wartości, to pierwszy element automatycznie otrzyma wartość 0, a kolejne o jeden większą od swojego poprzednika. Język C pozwala również na przypisanie tej samej liczby do więcej niż jednego elementu typu wyliczeniowego oraz na zmianę wartości wybranych elementów lub wybranej grupy elementów w takim typie. Typy wyliczeniowe mogą być definiowane globalnie lub lokalnie.

Typy wyliczeniowe

Przykłady

```
enum names_of_days {MONDAY=0, TUESDAY=1, WEDNESDAY=2, THURSDAY=3,
                    FRIDAY=4, SATURDAY=5, SUNDAY=6};
```

Typ ten nadaje wartości poszczególnym dniom tygodnia, począwszy do zera. To samo można zapisać nieco krócej w ten sposób:

```
enum names_of_days {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
                    SATURDAY, SUNDAY};
```

Jeśli chcemy, aby wartości dni tygodnia zaczynały się od 1, a nie od 0, to możemy to zapisać tak:

```
enum names_of_days {MONDAY=1, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
                    SATURDAY, SUNDAY};
```

Każdy kolejny dzień za poniedziałkiem otrzyma wartość większą od jego poprzednika, czyli `TUESDAY=2`, `WEDNESDAY=3`, itd.

Typy wyliczeniowe

Przykłady

Jeśli chcielibyśmy wyróżnić dni weekendu innymi wartościami, np. 9 i 10, to możemy zapisać to tak:

```
enum names_of_days {MONDAY=1, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
                    SATURDAY=9, SUNDAY};
```

Możemy także uszeregować wartości typu wyliczeniowego w kolejności malejącej:

```
enum directions {NORTH=3, WEST=2, EAST=1, SOUTH=0};
```

Zmienne typu wyliczeniowego

Zmienna typu wyliczeniowego może być zadeklarowana jako lokalna (w tym parametr) lub globalna. Wzorzec jej deklaracji jest następujący:

```
enum nazwa_typu_wyliczeniowego nazwa_zmiennej;
```

Tak zadeklarowanej zmiennej możemy przypisać dowolny element jej typu wyliczeniowego. Zmienne typu wyliczeniowego mogą pełnić rolę liczników w pętlach `for`, zmiennych sterujących (selektorów) w instrukcjach `switch` oraz mogą być użyte w warunkach w instrukcjach warunkowych oraz pozostałych pętlach. Niestety, sposób implementacji typów wyliczeniowych w języku C nie jest zbyt wyrafinowany. Stanowią one jedynie ułatwienie dla programisty, którego poprawności kompilator nie weryfikuje, traktując zmienne typu wyliczeniowego jako zmienne typu `int`, zatem zmiennej tego typu można przypisać dowolną liczbę całkowitą. Może to być przyczyną wielu błędów w programach. Język C umożliwia także tworzenie stałych typu wyliczeniowego z użyciem słowa kluczowego `const`.

Typy wyliczeniowe

Przykłady

```c
#include <stdio.h>

enum names_of_days {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY};

void print_message(enum names_of_days day)
{
    switch(day) {
    case MONDAY:
    case TUESDAY:
    case WEDNESDAY:
    case THURSDAY:
    case FRIDAY:
        puts("Do pracy!");
        break;
    default:
        puts("Wolne!");
    }
}

int main(void)
{
    enum names_of_days day;
    for(day=MONDAY;day<=SUNDAY;day++)
        print_message(day);
    return 0;
}
```

Typy wyliczeniowe

Komentarz do przykładu

W przykładowym programie zadeklarowano zmienną lokalną typu wyliczeniowego w funkcji `main()` (zmienna `day`) oraz parametr tego samego typu w funkcji `print_message()` (również o nazwie `day`). Ponadto program pokazuje jak takie zmienne mogą być użyte jako licznik pętli `for` oraz jako selektor w instrukcji `switch`. Na uwagę zasługuje również konstrukcja tej ostatniej instrukcji. Okazuje się, że pozostawienie przypadku pustego, nawet bez instrukcji `break` może być użyteczne. W wypadku opisywanego programu skróciło jego zapis. Funkcja `print_message()` po otrzymaniu wartości odpowiadającej dowolnemu przypadkowi, poza przypadkiem domyślnym, wykona instrukcję związaną z piątkiem. Instrukcje dla soboty i niedzieli wykonywane są jako przypadek domyślny. Niestety nie istnieje prosty sposób na wypisanie na ekranie nazw elementów zbioru za pomocą funkcji `printf()`. Możliwe jest jednak wypisanie ich wartości za pomocą ciągu formatującego `"%d"`.

Typy wyliczeniowe

Przykłady

```c
#include <stdio.h>

enum names_of_days {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY=9, SUNDAY};

void print_message(enum names_of_days day)
{
    switch(day) {
    case MONDAY:
    case TUESDAY:
    case WEDNESDAY:
    case THURSDAY:
    case FRIDAY:
        puts("Do pracy!");
        break;
    default:
        puts("Wolne!");
    }
}

int main(void)
{
    enum names_of_days day;
    for(day=MONDAY;day<=SUNDAY;day++)
        print_message(day);
    return 0;
}
```

Typy wyliczeniowe

Komentarz do przykładu

Niewielka zmiana w programie (nadanie elementowi SATURDAY wartości 9) ujawnia niedoskonałości typów wyliczeniowych. Po uruchomieniu przekonamy się, że tydzień teraz ma aż 11 dni, z czego większość jest wolna. Problemem jest zastosowanie zmiennej `day` jako licznika pętli `for`. Kiedy licznik ten osiągnie wartość 4 odpowiadającą elementowi FRIDAY, to w następnej iteracji zostanie zwiększony o jeden. Wartości 5 nie odpowiada żaden element typu wyliczeniowego, ale jest ona nadal poprawna, bo program traktuje zmienną `day` tak, jakby miała wartość typu `int`. Należy pamiętać o takich niuansach używając typów wyliczeniowych. Typ wyliczeniowy jest w języku C po prostu „pojemnikiem" na stałe.

Słowo kluczowe typedef

Pisanie słowa kluczowego enum przed każdą deklaracją zmiennej typu wyliczeniowego może być nużące i łatwo jest o nim zapomnieć. Celem zniwelowania tej niedogodności można użyć słowa kluczowego typedef, które pozwala nadać całej definicji typu krótszą nazwę np.:

```
typedef enum names_of_days {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY=9, SUNDAY} days;
```

Teraz zmienną tego typu można zadeklarować jako np.:

```
days day = MONDAY;
```

Słowo kluczowe typedef pozwala nadać nazwy także innym typom danych, nawet będącym częścią standardu języka. Należy więc stosować je z rozsądkiem. Wiele osób programujących w języku C zaleca w ogóle rezygnację z jego używania, bo, w ich opinii, degraduje ono czytelność kodu, ukrywając prawdziwy typ zmiennej.

Tablice jednowymiarowe

Jednowymiarowe tablice są przykładem *struktur danych*, czyli zmiennych, które potrafią przechowywać więcej niż jedną wartość w określonym porządku. W przypadku tablic są to dane tego samego typu. Ilustracja zamieszczona poniżej obrazuje tablicę jednowymiarową, która pozwala przechowywać do 8 liczb całkowitych.

0	1	2	3	4	5	6	7
-7	7	10	-3	0	7	15	0

Liczby u góry tablicy, napisane mniejszym fontem, to indeksy tablicy, które pozwalają jednoznacznie określić położenie wybranego elementu tablicy. W języku C indeksy są zawsze liczbami naturalnymi, a pierwszy indeks tablicy ma zawsze wartość 0. Element to pojedyncze miejsce w tablicy, które służy do przechowywania pojedynczej danej. O ile indeksy są niepowtarzalne i uszeregowane rosnąco, to wartości mogą być nieuporządkowane i wielokrotnie się powtarzać. Tablica nazywana jest niekiedy, niezbyt poprawnie, *wektorem*.

Tablice jednowymiarowe

Deklaracja

Tak jak inne zmienne, tablica musi zostać zadeklarowana przed jej użyciem. Tablice mogą być zarówno zmiennymi lokalnymi, jak i globalnymi. W drugim przypadku wartości ich elementów wynoszą domyślnie zero, w pierwszym są nieokreślone. Ogólny schemat deklaracji tablicy można przedstawić następująco:

```
typ_danych nazwa_tablicy[LICZBA_ELEMENTÓW];
```

Elementy tablicy mogą mieć dowolny z typów, które do tej pory zostały przedstawione na wykładzie. Tablice elementów typu `char` mają specjalne znaczenie i będą omawiane na odrębnym wykładzie. Nazwa tablicy jest identyfikatorem budowanym zgodnie z regułami języka C. Liczba elementów określa ile elementów będzie liczyła tablica. Najczęściej jest ona podawana wprost jako liczba, lub jest stałą zdefiniowaną z pomocą instrukcji `#define`. Zgodnie ze standardem ISO C99 liczba elementów musi być większa od zera. Zakres indeksów dla tablicy jednowymiarowej jest zawsze od 0 do `LICZBA_ELEMENTÓW-1`.

Tablice jednowymiarowe

Dostęp do elementów

Warto zauważyć, że tablica została skonstruowana przez analogię do budowy pamięci operacyjnej, o czym można się przekonać porównując uproszczony model pamięci przedstawiony na poprzednim wykładzie z ilustracją z jednego z poprzednich slajdów. Aby zapisać lub odczytać wartość z komórki pamięci procesor musi podać jej adres. Podobnie programista, który chce uzyskać dostęp do elementu tablicy musi podać jego indeks. Schemat takiego odwołania można przedstawić następująco: `nazwa_tablicy[indeks]`
W języku C nazwa tablicy jest równoznaczna (w większości przypadków) wskaźnikowi. Zatem to samo odwołanie może być wykonane na trzy sposoby:

```
*(nazwa_tablicy+indeks)
*(indeks+nazwa_tablicy)
 indeks[nazwa_tablicy]
```

Dwa pierwsze oparte są na tzw. arytmetyce wskaźników. Z czterech przedstawionych tu sposobów najczęściej stosowany jest pierwszy, czasem można spotkać drugi. Ostatnie dwa są stosowane rzadko z uwagi na ich małą czytelność.

Tablice jednowymiarowe

Rozmiar tablicy i liczba elementów

Rozmiar tablicy, czyli liczbę bajtów, które ona zajmuje w pamięci operacyjnej, można określić stosując operator `sizeof`. Liczbę jej elementów można określić stosując wyrażenie:

```
sizeof(nazwa_tablicy)/sizeof(nazwa_tablicy[0])
```

Stosują arytmetykę wskaźników, zaprezentowaną na poprzednim slajdzie można to wyrażenie zapisać także w następującej postaci:

```
sizeof(nazwa_tablicy)/sizeof(*nazwa_tablicy)
```

Niestety, te wyrażenia, ani operator `sizeof` nie działają, jeśli tablica jest parametrem funkcji.

Tablice jednowymiarowe

Przekazywanie do funkcji

Tablice można i należy przekazywać do funkcji. Parametr przekazujący tablicę do funkcji może mieć taką samą postać jak jej deklaracja, z dokładnością do nazwy. Liczba elementów jest ignorowana przez kompilator, co oznacza, że pod parametr tablicowy, który deklaruje, że ma np. 10 elementów można podstawić tablicę o dowolnej liczbie elementów i kompilator nie będzie w żaden sposób protestował. Jeśli typy elementów nie będą się zgadzały, to wystosuje jedynie ostrzeżenie. Najczęściej zatem pomija się liczbę elementów, zostawiając jedynie w parametrze puste nawiasy kwadratowe, co jest drugim sposobem przekazania tablicy przez parametr. Trzecim sposobem jest zadeklarowanie parametru jako wskaźnika takiego samego typu jak typ elementów tablicy lub typu wskaźnik na `void` (`void *`). Ta ostatnia deklaracja nie oznacza wskaźnika pustego, ale wskaźnik, któremu może być przypisany lub pod który może być podstawiony dowolny typ wskaźnikowy. Przekazanie tablicy działa zawsze jak przekazanie przez wskaźnik.

Inicjacja tablicy

Tablica zainicjowana

Tablice można zadeklarować jako zainicjowaną. Wystarczy po zamykającym nawiasie kwadratowym dodać instrukcję przypisania i wymienić wartości elementów tablicy w nawisach klamrowych, rozdzielając je przecinkami. W takim przypadku można pominąć liczbę elementów tablicy, zostawiając puste nawiasy. Zostanie ona ustalona na podstawie liczby wymienionych w nawiasach klamrowych wartości. Jeśli jednak zdecydujemy się ją podać, a w nawiasach klamrowych umieścimy mniej wartości, to część elementów nie zostanie zainicjowana. Przykład ilustruje taki sposób deklaracji tablicy:

```
int main(void)
{
        double fractions[] = {0.1, 0.2, 0.3};
        double fractions_2[3] = {0.1, 0.2, 0.3};
        return 0;
}
```

Inicjacja tablicy

Inicjacja przez użytkownika

Wartości elementów tablicy może podać również użytkownik za pomocą klawiatury. Tak określone dane należy wprowadzić w pętli do każdego elementu z osobna, używając funkcji `scanf()`. Ponieważ element jest pojedynczą zmienną, to przekazując go jako argument wywołania `scanf()` powinniśmy go poprzedzić operatorem wyłuskania adresu. Ilustruje to przykład:

```
int main(void)
{
        int array[5];
        unsigned int i;

        for(i=0;i<5;i++)
                scanf("%d",&array[i]);

        return 0;
}
```

Inicjacja tablicy

Inicjacja z wykorzystaniem indeksów

W przypadku dużych tablic trudno byłoby stworzyć zainicjowaną tablicę lub prosić użytkownika o jej wypełnienie. Jeśli elementy tej tablicy są typu `int` lub kompatybilnego, to można wykorzystać zmienną indeksującą do ich zainicjowania:

```
int main(void)
{
        int array[1000];
        unsigned int i;
        for(i=0;i<1000;i++)
                array[i] = i;
        return 0;
}
```

W ten sposób elementy uzyskają wartości swoich indeksów. Choć sposób ten jest prosty w realizacji, to wstępne uszeregowanie wartości elementów tablicy nie zawsze musi być pożądane.

Generator liczb pseudolosowych

Tablice można zainicjować za pomocą generatora liczb pseudolosowych. Ten mechanizm za pomocą określonej wartości początkowej i pewnych wzorów matematycznych wylicza wartości, które wydają się być losowe. Niestety, badania statystyczne wykazują, że tak nie jest, stąd generator ten określany jest mianem pseudolosowego. Dla potrzeb tego wykładu losowość, jaką daje ten mechanizm jest całkowicie wystarczająca, nie nadaje się ona jednak do poważniejszych zastosowań, takich jak kryptografia.

Generator liczb pseudolosowych

Korzystanie z generatora w języku C

W języku C dostępne są dwie funkcje zadeklarowane w pliku nagłówkowym `stdlib.h`, umożliwiające korzystanie z generatora liczb pseudolosowych. Są to `srand()` i `rand()`. Pierwsza przyjmuje jeden argument wywołania typu `unsigned int` i ustawia go jako pierwszą wartość ciągu liczb pseudolosowych (tzw. ziarno). Druga nie przyjmuje żadnych argumentów, ale zwraca liczbę pseudolosową typu `int` z zakresu od 0 do `RAND_MAX`. Funkcja `srand()` **musi być wywoływana poza wszelkimi pętlami**. Zazwyczaj jako argument przekazuje się do niej wartość zwróconą przez funkcję `time()` zadeklarowaną w pliku nagłówkowym `time.h`. Ta funkcja zwraca bieżący czas w postaci liczb całkowitej. Jako jej argument wywołania przekazuje się 0 lub `NULL`.

Generator liczb pseudolosowych

Sposób użycia

Generator liczb pseudolosowych generuje liczby naturalne. Jeśli chcielibyśmy, aby została wylosowana liczba z zakresu od 0 do 9, to możemy użyć następującego wyrażenia:

```
int x = rand()%10;
```

Aby zmienić zakres na od 1 do 10 należy zastosować takie wyrażenie:

```
int x = 1+rand()%10;
```

Jeśli interesuje nas zakres na od -10 do 10, to wyrażenie musi mieć postać:

```
int x = -10+rand()%21;
```

Jeśli chcemy, aby poprzedni zakres zmienić na przedział [-10,11), to musimy dodać część ułamkową:

```
double x = -10+rand()%21+rand()/(RAND_MAX+1.0);
```

Generator liczb pseudolosowych

Sposób użycia

Aby wylosować małą literę spośród 26 innych należ zastosować takie wyrażenie:

```
char x = 'a'+rand()%26;
```

Inicjacja tablicy

Inicjacja tablicy z użyciem liczb pseudolosowych

Poniższa funkcja wypełnia tablice o NUMBER_OF_ELEMENTS elementach liczbami pseudolosowymi z zakresu od 0 do 199:

```
void fill_array_with_random_numbers(int array[])
{
    srand(time(0));
    int i;
    for(i=0; i<NUMBER_OF_ELEMENTS; i++)
        array[i]=rand()%200;
}
```

Należy zauważyć, że tak losowane wartości mogą się powtarzać.

Inicjacja tablicy

Przestawianie

Korzystając z indeksów możemy uzyskać tablicę, której wartości elementów są uporządkowane rosnąco i nie powtarzają się. Możemy ją zamienić w tablicę, której wartości elementów się nie powtarzają, ale tworzą nieuporządkowany ciąg stosując algorytm przestawiania (ang. *shuffle*). Polega on na przeglądaniu kolejnych elementów takiej tablicy i zamianie ich wartości z losowo wybranymi elementami, spośród tych, które nie zostały jeszcze odwiedzone (włączając bieżąco badany). Algorytm ten został zaimplementowany za pomocą kilku funkcji, które będą przedstawione na kolejnych slajdach.

Inicjacja tablicy

Przestawianie - zamiana wartości elementów

Poniższa funkcja zamienia miejscami wartość dwóch zmiennych, które zostaną przekazane jako jej argumenty wywołania:

```
void swap(int *first, int *second)
{
        int tmp;
        tmp = *first;
        *first = *second;
        *second = tmp;
}
```

Inicjacja tablicy

Przestawianie - losowanie elementu

Poniższa funkcja losuje indeks elementu tablicy począwszy od bieżącego (`from`), a skończywszy na ostatnim (`ARRAY_LENGTH-1`):

```
int choose(int from)
{
        return from+rand()%(ARRAY_LENGTH-from);
}
```

Inicjacja tablicy

Przestawienie - implementacja

Poniższa funkcja dokonuje właściwego przestawienia elementów. Proszę zwrócić uwagę, że wybór drugiego z elementów tablicy, z którym należy wymienić wartość bieżącego elementu dokonywany jest za pomocą funkcji `choose()`:

```
void shuffle(int array[])
{
        srand(time(0));
        unsigned int i;
        for(i=0;i<ARRAY_LENGTH-1;i++)
                swap(&array[i],&array[choose(i)]);
}
```

Kopiowanie tablic

Dwie tablice o takich samych typach elementów można skopiować przy pomocy dowolnej instrukcji iteracyjnej. Jednakże bardziej efektywnym sposobem wykonania tej operacji jest użycie funkcji `memcpy()` zadeklarowanej w pliku nagłówkowym `string.h`. Ta funkcja przyjmuje trzy argumenty wywołania. Jako pierwszy przekazywana jest nazwa tablicy, do której kopiowane będą wartości, jako drugi nazwa tablicy z której będą kopiowane wartości, a jako trzeci rozmiar kopiowanej tablicy. Należy zadbać, aby tablica kopiowana była równa lub mniejsza rozmiarem od tablicy, do której następuje kopiowanie. Wartość zwracana przez funkcję `memcpy()` jest najczęściej ignorowana. Z tego samego pliku nagłówkowego, co `memcpy()` pochodzi również funkcja `memset()`. Przyjmuje ona trzy argumenty. Pozwala ona wielokrotnie skopiować ustaloną wartość do tablicy. Może ona być wykorzystana do inicjowania lub zerowania tablic. Ta funkcja przyjmuje trzy argumenty: nazwę tablicy, na której ma być wykonana operacja, liczbę typu `int`, która ma być do niej wpisana i rozmiar tablicy. Wartość zwracana przez funkcję jest zazwyczaj również ignorowana.

Wypisanie wartości elementów tablicy na ekranie

Do indeksowania poszczególnych elementów tablicy można użyć dowolnej instrukcji iteracyjnej, np. pętli `for`. Zaprezentowana poniżej funkcja wypisuje na ekran zawartość tablicy o 100 elementach typu `int`:

```
void print_array(int array[])
{
    unsigned int i;
    for(i=0; i<100; i++)
        printf("array[%u]: %d ",i,array[i]);
}
```

Sposób wypisania można też uprościć:

```
void print_array(int array[])
{
    unsigned int i;
    for(i=0; i<100; i++)
        printf(" %d ",array[i]);
}
```

Wyszukiwanie wartości minimalnej

Algorytm

W niektórych zagadnieniach należy znaleźć wartość minimalną w nieuporządkowanej tablicy. Algorytm jej wyszukiwania jest stosunkowo prosty:

1. Zapamiętaj w osobnej zmiennej wartość pierwszego elementu tablicy, jako wartość minimalną.
2. Przeglądaj kolejne elementy tablicy; jeśli któryś z nich ma wartość mniejszą od tej w zmiennej, to ją w niej umieść; teraz to będzie wartość najmniejsza.
3. Jeśli odwiedzone zostały wszystkie elementy w tablicy, to wartość najmniejsza znajduje się w zmiennej.

Wyszukiwanie wartości minimalnej

Implementacja

Poniższa funkcja implementuje ten algorytm dla tablicy o liczbie elementów określonej wartością stałej `NUMBER_OF_ELEMENTS`:

```
int find_min(int *array)
{
    int min;
    unsigned int i;
    min=array[0];
    for(i=1; i<NUMBER_OF_ELEMENTS; i++)
        if(min>array[i])
            min=array[i];
    return min;
}
```

Wyszukiwanie wartości maksymalnej

Implementacja

Algorytm wyszukiwania wartości maksymalnej w tablicy jest bardzo podobny do wyszukiwania wartości minimalnej - wystarczy tylko zmienić kilka wyrazów. Kod funkcji go implementującej różni się od kodu funkcji szukającej wartości minimalnej nazwą funkcji, zmiennej oraz znakiem w warunku instrukcji warunkowej:

```
int find_max(int *array)
{
    int max;
    unsigned int i;
    max=array[0];
    for(i=1; i<NUMBER_OF_ELEMENTS; i++)
        if(max<array[i])
            max=array[i];
    return max;
}
```

Wyszukiwanie ekstremów

Łatwo zauważyć, że w przypadku, gdy potrzebujemy zarówno wartości minimalnej, jak i maksymalnej, a nie tylko jednej z nich, korzystniej będzie szukać ich obu przeglądając tablicę tylko raz, a oba wyniki zwracać przez parametry funkcji:

```
void find_exterme_values(int array[], int *min, int *max)
{
    unsigned int i;
    *max = *min = array[0];
    for(i=1; i<NUMBER_OF_ELEMENTS; i++) {
        if(*min>array[i])
            *min=array[i];
        if(*max<array[i])
            *max=array[i];
    }
}
```

Wyszukiwanie określonej wartości w tablicy nieuporządkowanej

Algorytm

Często spotykanym problemem jest lokalizacja wartości w nieuporządkowanej tablicy. Jest wiele odmian tego problemu. Nas będzie interesowała ta, w której należy podać indeks pierwszego elementu od początku tablicy, który zawiera szukaną wartość. Algorytm rozwiązania tego problemu jest prosty: należy przeglądać kolejne elementy tablicy, aż do napotkania takiego, który zawiera szukaną wartość. Wówczas należy zwrócić wartość jego indeksu. Alternatywnie, jeśli żaden element tablicy nie zawiera takiej wartości, to należy zwrócić określoną wartość, która ten fakt zasygnalizuje np. `-1`.

Wyszukiwanie określonej wartość w tablicy nieuporządkowanej

Implementacja

Poniższa funkcja implementuje przedstawiony algorytm:

```
int find_value_index(int array[], int value)
{
    unsigned int i;
    for(i=0; i<NUMBER_OF_ELEMENTS; i++) {
        if(array[i]==value)
            return i;
    }
    return -1;
}
```

Wyszukiwanie wartości w tablicy - druga wersja

Algorytm

Jeśli przyjrzymy się poprzedniej funkcji, to zauważymy, że w każdej iteracji pętli `for` sprawdzane są dwa warunki: czy nie został osiągnięty ostatni element w tablicy oraz czy nie została znaleziona poszukiwana wartość. Okazuje się, że tę pętlę można uprościć. Choć rozwiązanie jest dosyć zaskakujące. Należy bowiem umieścić wartości wszystkich sprawdzanych elementów w tablicy o jeden większej niż wynosi ich liczba, a w nadmiarowym, ostatnim elemencie tablicy należy umieścić ...szukaną wartość! Takie rozwiązanie pozwala sprawdzać w pętli jedynie to, czy już nie znaleziono szukanej wartości. Nie ma konieczności sprawdzania, czy nie osiągnięto końca tablicy. Po zakończeniu pętli wystarczy zbadać, czy wartość zmiennej indeksującej jest równa indeksowi ostatniego elementu. Jeśli nie, to znaczy że szukana wartość wystąpiła wcześniej w tablicy, w elemencie, który określa zmienna indeksująca. Jeśli tak, to wartość nie wystąpiła wcześniej i należy zasygnalizować jej brak zwracając liczbę `-1`.

Wyszukiwanie wartości w tablicy - druga wersja

Implementacja

Poniższa funkcja implementuje opisany algorytm:

```
int faster_find_value_index(int *array, int value)
{
    int larger_array[NUMBER_OF_ELEMENTS+1];

    memcpy(larger_array,array,sizeof(array[0])*NUMBER_OF_ELEMENTS);
    larger_array[NUMBER_OF_ELEMENTS]=value;

    unsigned int i = 0;
    while(larger_array[i]!=value)
        i++;
    return (i!=NUMBER_OF_ELEMENTS)?i:-1;
}
```

Proszę zwrócić uwagę, że tablica oryginalna jest kopiowana przy pomocy `memcpy()`, a z uwagi na to, że jest ona przekazywana przez wskaźnik, to jej rozmiar jest określany jako iloczyn rozmiaru jej pierwszego elementu i stałej określającej liczbę jej elementów.

Podziękowania

Składam podziękowania dla dra inż. Grzegorza Łukawskiego i mgra inż. Leszka Ciopińskiego za udostępnienie materiałów, których fragmenty zostały wykorzystane w tym wykładzie.

Pytania

?

KONIEC

Dziękuję Państwu za uwagę.