Podstawy Programowania 1

Wyliczenia i tablice jednowymiarowe

Arkadiusz Chrobot

Katedra Systemów Informatycznych

13 listopada 2024

Plan

1. Abstrakcja danych
2. Wyliczenia
3. Słowo kluczowe `typedef`
4. Tablice jednowymiarowe
5. Tablice a funkcje
6. Inicjacja tablicy
7. 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

Wyliczenie pozwala nadawać nazwy liczbom należącym do określonego podzbioru liczb całkowitych, a także znakom. Innymi słowy, możemy myśleć o nim jako o zbiorze stałych. Musi ono mieć typ, którego wzorzec definicji 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};
```

Wyliczenia

Wyliczenie (zmienna typu wyliczeniowego) może być zadeklarowane jako lokalne (w tym parametr) lub globalne. Wzorzec jego deklaracji jest następujący:

```
enum nazwa_typu_wyliczeniowego nazwa_zmiennej;
```

Tak zadeklarowanemu wyliczeniu możemy przypisać dowolny element jego typu. Wyliczenia mogą pełnić rolę liczników w pętlach `for`, selektorów w instrukcjach `switch` oraz mogą być użyte w warunkach instrukcji `if`. Mogą być także używane w pętlach `while` i `do…while`. Niestety, sposób implementacji wyliczeń w języku C nie jest zbyt wyrafinowany. Stanowią one jedynie ułatwienie dla programisty, którego poprawności kompilator nie weryfikuje, traktując wyliczenia jako zmienne typu `int`, zatem zmiennej tego rodzaju 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`.

Wyliczenia

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;
}
```

Wyliczenia

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 wyliczenia 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 wyliczenia za pomocą funkcji `printf()`. Możliwe jest jednak wyświetlenie ich wartości za pomocą sekwencji konwertującej `%d`.

Wyliczenia

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;
}
```

Wyliczenia

Komentarz do przykładu

Niewielka zmiana w programie (nadanie elementowi `SATURDAY` wartości 9) ujawnia niedoskonałości wyliczeń. 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 wyliczenia, ale jest ona nadal poprawna, bo program traktuje zmienną `day` tak, jakby była typu `int`. Należy pamiętać o takich niuansach używając wyliczeń. Wyliczenie jest w języku C po prostu „pojemnikiem” na stałe.

Słowo kluczowe `typedef`

Pisanie słowa kluczowego **`enum`** przed każdą deklaracją wyliczenia może być nużące i łatwo jest o nim zapomnieć. Celem zniwelowania tej niedogodności można użyć słowa kluczowego **`typedef`**, które pozwala nadać typowi wyliczeniowemu inną (krótszą) nazwę np.:

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

Teraz wyliczenie tego typu można zadeklarować i (opcjonalnie) zainicjować np. tak:

```
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 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

Tablice składają się z *elementów*. Liczby u góry tablicy, napisane mniejszym fontem, to *indeksy*, które pozwalają jednoznacznie ustalić położenie wybranego jej elementu. W języku C indeksy są zawsze liczbami naturalnymi, a pierwszy z nich ma 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. W literaturze tablica nazywana jest niekiedy *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 wykładach. 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, nazywana także *długością tablicy*, określa ile elementów będzie liczyła tablica. Jest ona określana przez *wyrażenie stałe*, czyli takie, którego wartość jest znana w trakcie kompilacji. Najczęściej jednak długość jest podawana jako liczba lub jako stała zdefiniowana za pomocą instrukcji `#define`. Zgodnie ze standardem ISO C99 liczba elementów musi być większa od zera.

Tablice jednowymiarowe

Dostęp do elementów

Zakres wartości indeksów dla tablicy jednowymiarowej jest zawsze od `0` do `LICZBA_ELEMENTÓW-1`. Tablica jednowymiarowa jest skonstruowana przez analogię do budowy pamięci operacyjnej. 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 wskaźnikowi. Zatem wspomniane odwołanie może być wykonane na trzy inne 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])
```

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

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

Niestety, jeśli tablica jest parametrem funkcji, to operator `sizeof` podaje rozmiar jej pojedynczego elementu. Usunięcie tej niedogodności jest możliwe i zostanie przedstawione dalej.

Tablice a funkcje

Przekazywanie do funkcji

Funkcje bardzo ułatwiają implementowanie operacji na tablicach. W większej części tego wykładu nie będą prezentowane całe przykłady programów, tylko właśnie pojedyncze funkcje operujące na tablicy. Tablice można i należy przekazywać do funkcji. Parametr przekazujący tablicę do funkcji może mieć taką samą postać jak jej deklaracja, z dokładnością do nazwy. Przykład programu, w którym zastosowano taki parametr jest zamieszczony na następnym slajdzie. Tablica jest przekazywana jako argument do funkcji `print()`, kiedy jest ona wywoływana w funkcji `main()`. Aby określić tablicę jako argument funkcji, wystarczy podać jej nazwę. Funkcja `print()` wypisuje rozmiar parametru tablicowego. Jest on równy rozmiarowi wskaźnika, a nie rozmiarowi tablicy. Informację o tym otrzymamy w trakcie kompilacji od kompilatora, w formie ostrzeżenia.

Tablice a funkcje

Przekazywanie do funkcji

```c
#include<stdio.h>
#define NUMBER_OF_ELEMENTS 10

int array[NUMBER_OF_ELEMENTS];

void print(int array_parameter[NUMBER_OF_ELEMENTS])
{
        printf("Rozmiar: %lu\n", sizeof(array_parameter));
}

int main(void)
{
        print(array);
        return 0;
}
```

Tablice a funkcje

Przekazywanie do funkcji

Liczba elementów tablicy w parametrze jest ignorowana przez kompilator. 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 parametru i argumentu będą się zgadzały. Najczęściej zatem pomija się liczbę elementów, zostawiając w parametrze jedynie puste nawiasy kwadratowe. Na kolejnym slajdzie zamieszczony jest ten sam program co poprzednio, ale bez liczby elementów w parametrze.

Tablice a funkcje

Przekazywanie do funkcji

```c
#include<stdio.h>
#define NUMBER_OF_ELEMENTS 10

int array[NUMBER_OF_ELEMENTS];

void print(int array_parameter[])
{
        printf("Rozmiar: %lu\n", sizeof(array_parameter));
}

int main(void)
{
        print(array);
        return 0;
}
```

Tablice a funkcje

Przekazywanie do funkcji

Tablicę można także przekazać do funkcji przez wskaźnik takiego samego typu, jak typ elementów tablicy lub typu `void *` (wskaźnik na `void`). Ten ostatni jest wskaźnikiem, któremu może być przypisany lub pod który może być podstawiony dowolny inny typ wskaźnikowy, a także dowolna tablica. Następny slajd przedstawia zmodyfikowany program z poprzedniego slajdu, w którym tablica jest przekazywana przez wskaźnik.

Każdy z zaprezentowanych sposobów deklarowania parametru dla tablicy, z punktu widzenia programu, sprowadza się do przekazania jej przez wskaźnik. Dowolna modyfikacja wartości elementów tablicy w funkcji jest trwała, tzn. będzie widoczna po jej zakończeniu.

Tablice a funkcje

Przekazywanie do funkcji

```
#include<stdio.h>
#define NUMBER_OF_ELEMENTS 10

int array[NUMBER_OF_ELEMENTS];

void print(int *array_parameter)
{
        printf("Rozmiar: %lu\n", sizeof(array_parameter));
}

int main(void)
{
        print(array);
        return 0;
}
```

Tablice a funkcje

Przekazywanie do funkcji

Aby móc wyznaczyć w funkcji liczbę elementów tablicy przy użyciu podanego wcześniej sposobu należy w programie najpierw zdefiniować typ tablicowy przy użyciu słowa kluczowego `typedef`, a następnie zadeklarować w funkcji parametr tego samego typu. Wówczas, w ciele funkcji można wyznaczyć rozmiar tablicy przy pomocy operatora `sizeof`, podając jako jego argument ten typ. To rozwiązanie zastosowane jest w programie znajdującym się na następnym slajdzie.

Tablice a funkcje

Przekazywanie do funkcji

```c
#include<stdio.h>
typedef int int_array_type[10];
int_array_type array;

void print(int_array_type array_parameter)
{
    printf("%lu\n",
    sizeof(int_array_type)/sizeof(*array_parameter));
}

int main(void)
{
    print(array);
    return 0;
}
```

Tablice a funkcje

Zwracanie tablicy przez funkcję

Tablice nie mogą być bezpośrednio zwracane przez funkcje. Można jednak zwrócić ich adres, deklarując typ wartość zwracanej przez funkcję, jako odpowiedni typ wskaźnikowy. **Nigdy nie należy zwracać z funkcji adresu tablicy, ani żadnej innej zmiennej, która jest w niej zadeklarowana lokalnie. Po zakończeniu tej funkcji ta zmienna przestaje istnieć.** Wyjątkiem od tej reguły są zmienne lokalne zadeklarowane z użyciem słowa kluczowego `static`.

Tablice a funkcje

Tablice o zmiennej liczbie elementów

Standard ISO C99 dopuszcza możliwość deklarowania tablic lokalnych, których liczba elementów jest określona za pomocą zmiennej, czyli *tablic o zmiennej liczbie elementów*. Choć takie zmienne wydają się być sporym udogodnieniem, to mogą też sprawiać wiele problemów. Jeśli bowiem wartość wspomnianej zmiennej będzie nieprawidłowa, to program nie będzie działał prawidłowo. W szczególności, jeśli zmienna będzie określała zbyt dużą liczbę elementów, to zostanie przekroczony rozmiar stosu, co będzie skutkowało zakończeniem programu w trybie awaryjnym. Innym problemem dotyczącym tablic o zmiennej liczbie elementów jest to, że nie można ich zainicjować w miejscu deklaracji, bo przed wykonaniem programu ich długość nie jest znana. Z tych powodów w standardzie ISO C11 tego rodzaju tablice są opcjonalne. Przykład programu tworzącego taką tablicę znajduje się na następnym slajdzie. Z uwagi, że ta tablica nie jest w nim używana, to w takcie jego kompilacji pojawia się ostrzeżenie.

Tablice a funkcje

Tablice o zmiennej liczbie elementów

```
void create_vla(unsigned int number_of_elements)
{
        if(number_of_elements>0 && number_of_elements<1000) {
                int array[number_of_elements];
        }
}

int main(void)
{
        create_vla(100);
        return 0;
}
```

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 ostatnie elementy będą zainicjowane zerami. 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 oznaczona

Najprostszy sposób na zainicjowanie wszystkich elementów tablicy lokalnej prezentuje fragment kodu zamieszczony niżej.

```
int main(void)
{
        int array[100]={0};
        return 0;
}
```

Począwszy od standardu ISO C99 język C oferuje również *inicjację oznaczoną* (ang. *designated initializers*), która pozwala wskazać, którym elementom tablicy należy przypisać określone wartości:

```
int main(void)
{
        int array[100]={[5]=2,[3]=1};
        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 (`length`-1):

```
int choose(int from, const unsigned int length)
{
        return from+rand()%(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[], const unsigned int length)
{
        srand(time(0));
        unsigned int i;
        for(i=0;i<length-1;i++)
                swap(&array[i],&array[choose(i, length)]);
}
```

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 `length` elementach typu `int`:

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

Sposób wypisania można też uprościć:

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

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.