Podstawy Programowania

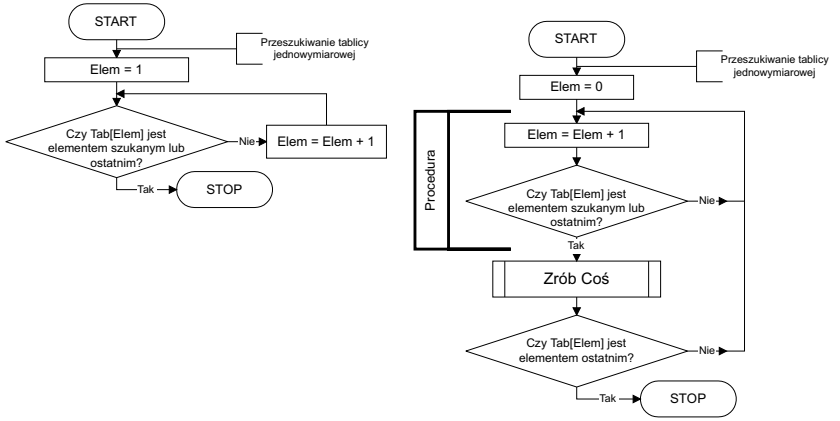
Wykład VI

Tablice i wskaźniki cd., argumenty wywołania programu, reguły stylu programowania, asercje

Robert Muszyński
ZPCiR IIAiR PWr

Zagadnienia: przeszukiwanie tablic, inicjowanie tablic, interfejs użytkownika programu, argumenty wywołania programu, reguły stylu programowania, dokumentacja programu, asercje.

Przeszukiwanie tablic




```
#define NWARTOSCI 30
/*************************************************************/
int ZnajdzNastepna(int tab[], int w, int i){
/*    Wyszukuje w tablicy podana wartosc w zaczynajac od     */
/*    pozycji i + 1, o ile jest wewnatrz tablicy. Zwraca     */
/*    znaleziony indeks lub -1 gdy nie zostal znaleziony.    */
  int znaleziona = 0;
  i++;
  while ((i >= 1) && ( i <= NWARTOSCI) && !znaleziona)
    if (tab[i] == w)
      znaleziona = 1;
    else i++;
  if (znaleziona)
    return (i);
  else return (-1);
}       /* ZnajdzNastepna */
/*************************************************************/
int main(){
  int wartosci[NWARTOSCI];
  int i = 0; int szukana = ... /* Inicjacja wartosci */
  do{
    i = ZnajdzNastepna(tablica, szukana, i);
    if (i != -1)
      printf("Szukana wartosc znaleziona na pozycji %1d\n", i);
  } while (i != -1);
}
```


Tablice wielowymiarowe

- Tablica dwuwymiarowa N×M elementowa

tab[1,1]	tab[1,2]	tab[1,3]	···	tab[1,M]
tab[2,1]	tab[2,2]	tab[2,3]	···	tab[2,M]
⋮	⋮	⋮	⋱	⋮
tab[N,1]	tab[N,2]	tab[N,3]	···	tab[N,M]

tab[i,j] ⇔ prosta zmienna

- Deklaracja tablicy dwuwymiarowej ma postać

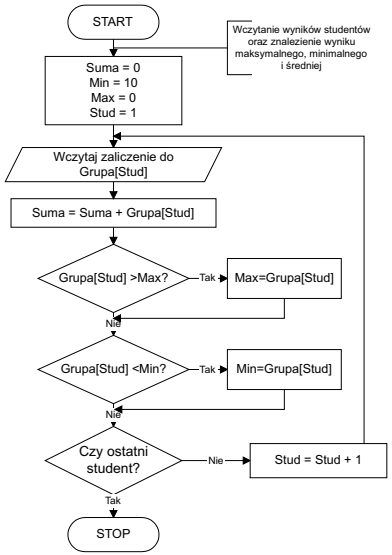
```
int oceny[10][200];  /* [wiersz][kolumna] */
```

- Odwołanie do tablicy dwuwymiarowej:

```
oceny[i][j];  /* a nie oceny[i,j] */
```

- Deklaracja tablicy wielowymiarowej:

```
typ_elementu nazwa_tablicy[wymiar_1][wymiar_2][wymiar_3]...;
```

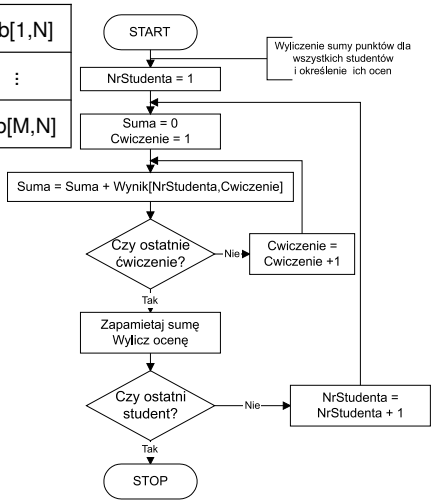



Tablica dwuwymiarowa — przykład

Ćwiczenie →

NrStudenta ↓

tab[1,1]	· · ·	tab[1,N]
⋮	⋱	⋮
tab[M,1]	· · ·	tab[M,N]




```
#define NSTUDENTOW 200            /* calkowita liczba studentow */
#define NCWICZEN 8       /* liczba cwicz. do wykon. w semestrze */

int main() {
  int Zal1[NSTUDENTOW], Zal2[NSTUDENTOW];
  int Wyn1[NSTUDENTOW][NCWICZEN+1], Wyn2[NSTUDENTOW][NCWICZEN+1];
  int stud, cwicz, suma;

/*****************************************************************/
  for (stud = 0; stud < NSTUDENTOW; stud++)
  {
    Suma = 0;
    for (cwicz = 1; cwicz <= NCWICZEN; cwicz++)
      Suma += Wyn1[stud][cwicz];
    Wyn1[stud][0] = Suma;
    Zal1[stud] = 2 + (3 * Suma) / (NCWICZEN * MAXPUNKT);
  }
}
```


Zakres indeksów tablic

W języku C nie jest sprawdzana poprawność (zakres) indeksów!

Przykładowo tablica

```
int macierz[4][2]  /* dwuwymiarowa: 4 wiersze po 2 kolumny */
```

ma postać

[0][0]	[0][1]
[1][0]	[1][1]
[2][0]	[2][1]
[3][0]	[3][1]

[1][2]

i reprezentację w pamięci

[0][0]	[0][1]	[1][0]	[1][1]	[2][0]	[2][1]	[3][0]	[3][1]

Odwołanie

```
macierz[1][2]
```

wskaże element `[2][0]`.

Inicjowanie tablic

Przy definiowaniu zmiennych można je zainicjować

```
int i = 10;
char c = 'a';
```

Dotyczy to także tablic

```
int tab[10] = {20, -3, 12, 1, 0, 7, -5, 100, 2, 5};

float mac_A[3][2] = {{1,1}, {3.5, 7.0}, {-15, 100}};
float mac_B[3][2] = {1, 1, 3.5, 7.0, -15, 100};  /* rownowazne */

char znaki[5] = {'a', 'B', '\n', '1', '\0'};        /* lub tez */
char znaki[5] = "napi"; /* automatycznie dodawany jest znak \0 */
```

Gdy lista inicjatorów jest krótsza niż liczba elementów tablicy zostaje uzupełniona zerami lub wskaźnikami `NULL`

```
int tab[10] = {20, -3, 12, 1};                    /* rownowazne */
int tab[10] = {20, -3, 12, 1, 0, 0, 0, 0, 0, 0};

float mac[3][2] = {{1}, {3.5, 7.0}};              /* rownowazne */
float mac[3][2] = {{1, 0}, {3.5, 7.0}, {0, 0}};
float mac[3][2] = {1, 0, 3.5, 7.0, 0, 0};
```


Wskaźniki a tablice wielowymiarowe

Po definicjach:

```
int a[10][20];
int *b[10];
```

odwołania typu

```
a[3][4];
b[3][4];
```

są poprawne. Jednakże `a` jest dwuwymiarową, 200-elementową tablicą liczb całkowitych — do znalezienia elementu `a[wiersz][kolumna]` stosuje się tradycyjny wzór `20*wiersz+kolumna`. Natomiast definicja `b` przydziela jedynie 10 miejsc pamięci na wskaźniki i nie inicjuje ich — nadanie wartości początkowych musi być zrobione jawnie: statycznie lub programowo. Dzięki temu wiersze tablicy `b` mogą mieć różną długość.

Porównaj definicje:

```
char nazwy[][15] = {"Bledny miesiac", "Sty", "Lut", "Mar"};
char *nazwy[]    = {"Bledny miesiac", "Sty", "Lut", "Mar"};
```


Funkcje operujące na tablicach cd.

Przekazywanie funkcji tablicy dwuwymiarowej:

```
f(int tablica[2][13]) {...}
f(int tablica[][13])  {...}
f(int (*tablica)[13]) {...}
```

- Ogólnie, tylko pierwszy wymiar tablicy jest dowolny — wszystkie pozostałe muszą być jawnie określone.
- Ostatnia deklaracja mówi, że argument jest wskaźnikiem do tablicy 13 liczb całkowitych. Nawiasy okrągłe są konieczne gdyż deklaracja

  ```
  int *tablica[13]
  ```

  wprowadza tablicę 13 wskaźników do obiektów całkowitych.

Inerfejs użytkownika programu – menu

```
#include <stdio.h>        /* proste menu tekstowe uzytkownika */
int main (){
  char wybor[2] = " ";    /* dowolne byle nie "0" - koniec */

  while (wybor[0] != '0') {
    printf("Opcje menu:\n");
    printf("  1 - Wczytaj plik\n");
    printf("  2 - Negatywuj\n");
    printf("  0 - Zakoncz\n");
    printf("Twoj wybor: ");

    scanf("%1s", wybor);

    switch (wybor[0]) {
    case '1': wczytaj(...); break;
    case '2': negatyw(...); break;
    case '0': break;
    default : printf("\t\tnierozpoznane\n"); break;
    }
  } /* while */
} /* main */
```

Argumenty wywołania programu

By uruchomić program znajdujący się w pliku `prog` należy wykonać polecenie

```
prog                    /* alternatywnie ./prog */
```

Do uruchamianego programu można przekazać parametry, czyli argumenty wywołania

```
prog moj_plik.wej 17 tak
```

które dostępne są w funkcji `main` poprzez jej standardowo definiowane argumenty

```
main(int argc, char *argv[])
```

gdzie `argc` jest liczbą argumentów, z jakimi wywołano program, zaś `argv` jest wskaźnikiem do tablicy zawierającej argumenty, każdy jako osobny tekst.

W powyższym przykładzie `argc` jest równe 4, a `argv[0]`, `argv[1]`, `argv[2]`, `argv[3]` wskazują odpowiednio napisy `"prog"`, `"moj_plik.wej"`, `"17"`, `"tak"`.

Pierwszym „prawdziwym” argumentem jest `argv[1]` a ostatnim `argv[argc-1]`.

Argumenty wywołania programu — przykład

```
#include <stdio.h>
/* echo argumentow wywolania programu  -  wersja nr 1 */
int main (int argc, char *argv[]) {
  int i;
  for (i = 1; i < argc; i++)
    printf("%s%s", argv[i], (i < argc - 1) ? " ": "");
  printf("\n");
  return 0;
}
```

```
#include <stdio.h>
/* echo argumentow wywolania programu  -  wersja nr 2 */
int main (int argc, char *argv[]) {
  while (--argc > 0)
    printf("%s%s", *++argv, (argc > 1) ? " ": "");
  printf("\n");
  return 0;
}
```


Argumenty wywołania programu — konwencje

- Dla programów w systemie UNIX argument wywołania rozpoczynający się znakiem minus wprowadza nieobowiązkowy sygnalizator lub parametr programu, nazywany opcją, np.
  `find -x -n wzorzec`
- W programie należy przewidzieć możliwość podawania opcji w dowolnej kolejności.
- Dużą wygodą dla użytkowników jest możliwość grupowania opcji, np.
  `find -nx wzorzec`
- Można także przewidzieć istnienie opcji z dodatkowymi parametrami, np.
  `find -n 5 -x wzorzec`
- Przy potrzebie uzyskania wartości numerycznej argumentu można posłużyć się funckją `atoi` z nagłówka `stdlib.h`.
- Wypasioną obsługę argumentów wywołania programu można uzyskać z wykorzystaniem funkcji `getopt` z nagłówka `stdio.h` lub `unistd.h`.

Argumenty wywołania programu — analiza opcji

```
#include <stdio.h>        /* przyklad analizy opcji wywolania */
int main (int argc, char *argv[]){
  int c, except = 0, number = 0, found = 0;
  while (--argc > 0 && (*++argv)[0] == '-')
    while (c = *++argv[0])
      switch (c) {
      case 'x':
        except = 1; break;
      case 'n':
        number = 1; break;
      default:
        fprintf(stderr, "find: nieznana opcja %c\n", c);
        argc = 0; found = -1; break;
      }
  if (argc != 1)
    fprintf(stderr, "Format wywolania: find -x -n wzorzec\n");
  else ...       /* tu znajduje sie zasadnicza czesc programu */
  return found;
}  /* main */
```

Reguły stylu programowania

(a) poprawny podział programu na podprogramy, zgodnie ze strukturą opracowywanego zagadnienia,

(b) umiejętny dobór nazw tak, by wyjaśniały rolę zmiennych, stałych, procedur itp., i sugerowały czego te obiekty tyczą,

(c) postać graficzna programu (oddzielenia pionowe i poziome, wcięcia) tak zaplanowane, aby ukazać jego strukturę i zwiększyć czytelność,

(d) dokumentacja programowa (komentarze), dostarczająca informacji do dalszej pracy nad programem, wyjaśniająca wykorzystane rozwiązania, itp.,

(e) elementy formalnie dokumentujące utworzone oprogramowanie odróżnione od zawartych w nim komentarzy dla programistów,

(f) asercje, pozwalające zweryfikować poprawność wywołań funkcji,

(g) zasada lokalności, która mówi, że elementy powiązane ze sobą powinny występować razem, a niezwiązane — oddzielnie.

Dokumentacja programu

- komentarze „marginesowe”

```
#define   SygnalAmp 10        /* amplituda sygnalu wejsciowego  */
#define   KoniecDanych 99     /* sygnalizuje koniec pracy prog  */
#define   GrupaDanych 20      /* wielk.grupy danych do zliczania*/
#define   XZeroMin 5          /* minimalna ilosc przeciec zera  */
#define   XZeroMax 10         /* maksymalna ilosc przeciec zera */
#define   Sprawdz TRUE        /* czy sprawdz.poprawn.wart.sygn. */

  int  sygnal;       /* pobrana wartosc sygnalu         */
  char bylplus;      /* czy poprzedni sygnal byl dodat. */
  int  nsygn;        /* licznik wczytanych wart. sygn.  */
  int  nxzero;       /* licznik przeciec zera w grupie  */
```



```
#define  StopnieNaRadiany 0.017453292        /* dla przeliczen */
#define  SumaKatow 180.0                     /* ...w trojkacie */
```

```
  scanf("%d", &sygnal);     /*pierwsza wart. dla porown. w petli*/
  nsygn = 1;
  bylplus = (sygnal == abs(sygnal));      /* tylko wtedy sygn + */
```

```
  if (((sygnal > 0) && !bylplus) ||
      ((sygnal < 0) &&  bylplus))
  {                       /* wykryte przeciec. zera - zapamietac! */
    nxzero++;
    bylplus = (sygnal == ABS(sygnal));    /* tylko wtedy sygn + */
  }
```


- komentarze „in-line”

```
  if ((b*b - 4.0*a*c) >= 0.0)
  {
    /* wiemy ze istnieja dwa pierwiastki rzeczywiste,  */
    /* pozostaje je wyliczyc i wyswietlic na terminalu */
    sqdelta = sqrt(b*b - 4.0*a*c);
    x1 = (-b - sqdelta) / (2.0 * a);
    x2 = (-b + sqdelta) / (2.0 * a);
    /* ... */
```

- komentarze „blokowe”

```
/*   +-------+-------+-------+-------+-------+------+   */
/*   |   UWAGA: ponizszy fragment programu zostal   |   */
/*   |   przeniesiony  spod  Turbo  C  i  nie  ma   |   */
/*   |   pewnosci co do jego poprawnego dzialania.  |   */
/*   |   Uzywaj tylko na wlasna odpowiedzialnosc!   |   */
/*   +-------+-------+-------+-------+-------+------+   */
```

- komentarze opisujące działanie funkcji

```
void DodajDni( int  Dzien1, int  Miesiac1, int  Rok1, int NDni,
               int *Dzien2, int *Miesiac2, int *Rok2);
/*********************************************************
 *  Funkcja dodaje okreslona ilosc dni do podanej       *
 *  daty i oblicza nowa date                            *
 *********************************************************
 *  Parametry:                                          *
 *       Dzien1   - dzien daty wyjsciowej (1-31)        *
 *       Miesiac1 - miesiac daty wyjsciowej (1-12)      *
 *       Rok1     - rok daty wyjsciowej (pelny,np.1996) *
 *       NDni     - liczba dni do dodania (+ lub -)     *
 *       Dzien2, Miesiac2, Rok2 - nowo wyliczona data   *
 *********************************************************
 *  PRE: Dzien1,Miesiac1,Rok1 musza okreslac poprawna   *
 *       date nie wczesniejsza niz 1 stycznia 1970      *
 * POST: Dzien2,Miesiac2,Rok2 otrzymuja date            *
 *       pozniejsza (lub wczesniejsza gdy NDni < 0)     *
 *       o NDni od Dzien1,Miesiac1,Rok1                 *
 *********************************************************/
```


Asercje

Dodatkowe instrukcje, sprawdzające spełnienie warunków PRE

```
void Pierwiastki(float A, float B, float C,
                 float *X1, float *X2)
/*  PRE: parametry A,B,C musza spelniac warunki          */
/*       istnienia rozwiazan rzeczywistych rownania      */
/*       kwadratowego: B*B-4*A*C>=0.0 oraz A<>0.0        */
/* POST: parametry X1,X2 przyjmuja wartosci pierwiastkow */
/*       rownania kwadratowego o wspolczynnikach A,B,C   */
{
  float PierwDelta;
  if ((B*B-4.0*A*C<0.0) || (A=0.0))
    printf("Blad asercji: procedura Pierwiastki\n");
  else {
    PierwDelta = sqrt(B * B-4.0 * A *C);
    *X1 = (-B - PierwDelta) / (2.0 * A);
    *X2 = (-B + PierwDelta) / (2.0 * A);
  }
}    /* Pierwiastki */
```


Asercje — dalsze przykłady

```
void Assert(int a, char *komunikat);
/*********************************************************/
/*  PRE: zadne                                           */
/*  POST: w przypadku niespelnionego warunku a wyswietla */
/*        na ekranie komunikat i zatrzymuje program      */
/*********************************************************/
{
  if (!a)
  {
    printf("Blad asercji: %s\n", komunikat);
    printf("Program nie moze kontynuowac!\n");
    exit (-1);             /* funkcja z naglowka stdlib.h */
     /* sa zdefiniowane stale EXIT_SUCCESS i EXIT_FAILURE */
     /*      jest dostepna takze funkcja void abort(void) */
  }
}   /* Assert */
```

W nagłówku `assert.h` dostępne jest makro `assert`.


```
void Pierwiastki(float A, float B, float C,
                 float *X1, float *X2)
/*  PRE: parametry A,B,C musza spelniac warunki          */
/*       istnienia rozwiazan rzeczywistych rownania      */
/*       kwadratowego: B*B-4*A*C>=0.0 oraz A<>0.0        */
/* POST: parametry X1,X2 przyjmuja wartosci pierwiastkow */
/*       rownania kwadratowego o wspolczynnikach A,B,C   */
{
  float PierwDelta;
  Assert(((B*B-4.0*A*C>=0.0) && (A!=0.0)),
         "Pierwiastki: niepopr.wspolcz.rown.kwadrat.");
  PierwDelta = sqrt(B * B-4.0 * A *C);
  *X1 = (-B - PierwDelta) / (2.0 * A);
  *X2 = (-B + PierwDelta) / (2.0 * A);
}    /* Pierwiastki */
```

Podsumowanie

- Zagadnienia podstawowe

1. Jak wygląda deklaracja, inicjowanie i odwołanie do tablicy dwuwymiarowej? A trójwymiarowej?
2. Czy przy deklaracji tablicy `int tab[5][5]` poprawne jest odwołanie `tab[0][10]`?
3. Czy możliwa jest deklaracja zmiennej `char tab[3] = "ala";`?
4. Czy do przeglądania tablic dwuwymiarowych trzeba koniecznie używać dwóch pętli?
5. Czy stosowanie dwóch pętli do przeglądania tablic dwuwymiarowych nie jest konieczne, ale ułatwia pracę z nimi?
6. Czy dla kwadratowej tablicy dwuwymiarowej `int tablica[MAX][MAX]` odwołania `tablica[i][j]` i `tablica[j][i]` wskazują na ten sam element?
7. Czy pętle `for(i=0,i<X_MAX,i++) for(j=0,j<Y_MAX,j++) tablica[i][j]=0` oraz `for(i=0,i` spowodują wyzerowanie wszystkich elementów tablicy `int tablica[X_MAX][Y_MAX]`? Czy „odwiedzą” elementy tablicy w tej samej kolejności?
8. Czy można przekazać do funkcji jako jej argument tablicę wielowymiarową? Jeśli tak – w jaki sposób?
9. Czy zmienne lokalne są potrzebne w programie? Jeśli tak, to w jakim celu?
10. Wyjaśnij znaczenie argumentów `argc` i `argv`.
11. Jak należy skonstruować program, aby móc wczytywać argumenty wywołania programu?

12. W jaki sposób można rozpoznać i przetworzyć opcje przekazane do programu?
13. Do czego służy i jaka jest składnia instrukcji `switch`?
14. Czy instrukcję `switch` można zastosować do wyrażenia typu `float`?
15. Czy napisane dotychczas przez Ciebie programy są zgodne z regułami stylu programowania? Co należałoby poprawić?
16. Jaka jest rola asercji w programie i jak się je definiuje?

- Zagadnienia rozszerzające

1. Wyjaśnij pojęcie tablicy o zmiennym rozmiarze (VLA, ang. variable-length array).
2. Poczytaj o zastosowaniach w odniesieniu do tablic i zmiennych wskaźnikowych słowa kluczowego `const`.
3. W jaki sposób można wykorzystać konstrukcję `int *b[10];`?
4. W jaki sposób powłoka systemowa przetwarza argumenty wywołania przed przekazaniem ich do programu (sprawdź np. argumenty zawierające *, spację)? W jaki sposób uniknąć tego przetwarzania?
5. W jaki sposób można korzystać z biblioteki `getopt`?
6. Co musi zawierać specyfikacja funkcji, by była kompletna? Jakie inne sposoby, poza pokazanym na wykładzie a bazującym na warunkach PRE i POST, pozwalają na podanie specyfikacji funkcji?
7. W jaki sposób wykorzystywać zasady dobrego stylu przy wykorzystaniu narzędzi wspomagających tworzenie dokumentacji, takich jak `doxygen`?

- Zadania

1. Dopisz asercje do dotychczasowo napisanych programów.
2. Zapisz funkcję `int replace(int *tab, int a, int b)` zamieniającą każdy element tablicy `tab` o wartości `a` na `b`.
3. Napisz program, który zapełni tablicę dwuwymiarową wartościami funkcji `f(x,y) = sin(x/10`
4. Zamiast deklarowania tablicy wielowymiarowej można korzystać z tablicy jednowymiarowej odpowiednio przeliczając indeksy. Napisz funkcje, które umożliwią przeliczanie wartości między indeksami tablicy wielowymiarowej i odpowiadającej jej – jednowymiarowej.
5. Zapisz program wypisujący argumenty z jakim został wywołany w odwrotnej kolejności, począwszy od ostatniego.
6. Napisz program, który jako argumenty wywołania przyjmuje dowolny ciąg liczb naturalnych a zwraca ich sumę, następnie zmodyfikuj go tak, by argumentami mogły być liczby rzeczywiste.
7. Napisz program, który wykorzystując swoje parametry wywołania będzie wypisywać na ekranie zawartość dowolnie wskazanego pliku tekstowego, przykładowe wywołanie programu: `\.a.out dane.txt`.
8. Napisz program, który prosi kolejno użytkownika o podanie danych: imienia, nazwiska i nr telefonu. Podawane dane zapisuje do ustalonego pliku tekstowego w postaci:
   ```
   Jan
   Kowalski
   2345672
   ```



   ```
   Waldemar
   Nowak
   3345223
   ```
9. Wykorzystując parametry wywołania programu napisać prosty kalkulator dwuargumentowy. Program powinien przyjmować 3 parametry: liczbę, liczbę i operację, która ma być wykonana, np. `\.a.out 2 4 *`
10. Napisz program, który inicjalizuje tablicę, a następnie kopiuje jej zawartość do dwóch innych tablic (wszystkie trzy tablice powinny być zadeklarowane w zmienne lokalne funkcji `main`). Do wykonania pierwszej kopii użyj funkcji wykorzystującej notację tablicową. Do wykonania drugiej kopii użyj funkcji wykorzystjącej zapis wskaźnikowy i zwiększanie wskaźników. Każda funkcja powinna przyjmować jako argumenty nazwę tablicy źródłowej, nazwę tablicy docelowej oraz rozmiar tablic.