Tablice i struktury

Wstęp do Informatyki i Programowania

Maciek Gębala

24 października 2024

Notatki

Złożone typy danych

Tablica

Przechowuje elementy jednego typu indeksowane typem porządkowym (najczęściej liczbami).

Struktura jednorodna, homogeniczna.

Struktura

Przechowuje elementy różnych typów identyfikowane przez nazwy.

Struktura niejednorodna, heterogeniczna.

Notatki

Język C

Tablice są zawsze indeksowane od 0, operator dostępu to `[]`, rozmiar podawany przy tworzeniu (nie musi być stały), brak kontroli zakresów i informacji o wielkości.

W języku C tablice do funkcji są przekazywane przez nazwę (adres). Nie są więc kopiowane i są zmieniane globalnie.

Struktury to kolekcja danych z nazwami - ułatwiają organizację danych. Mogą być nazwane i traktowane jak pojedynczy typ prosty (`typedef`).

Notatki

Przykład w języku C

example.c

```c
#include <stdio.h>
#include <math.h>
typedef struct point {
  float x;
  float y;
} point;
typedef point segment[2];
float lengthp(point a, point b) {
  return sqrt( (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y) );
}
float lengths(segment s) {
  return sqrt( (s[0].x-s[1].x)*(s[0].x-s[1].x)+
               (s[0].y-s[1].y)*(s[0].y-s[1].y) );
}
int main() {
  point a, b;
  segment c;
  a.x = -1.0;  a.y = -2.0;
  b.x = 3.0;   b.y = 1.0;
  c[0] = a;    c[1] = b;
  printf("%f\n%f\n", lengthp(a, b), lengths(c));
  return 0;
}
```


Notatki

Język Ada

Tablice są indeksowane dowolnie, operator dostępu to `()`, rozmiar podawany przy tworzeniu (nie musi być stały, może zależeć od parametrów wejściowych podprogramu), pełna kontrola zakresów, można pobrać zakres i długość tablicy.

Struktury są tworzone słowem `record`.

Szablony struktur i tablic mogą być nazwane i traktowane jak pojedynczy typ prosty (`type`).

Notatki

Przykład w języku Ada

example.adb

```
1  with Ada.Text_IO;
2  use Ada.Text_IO;
3  with Ada.Numerics.Elementary_Functions;
4  use Ada.Numerics.Elementary_Functions;
5
6  procedure example is
7     type Point is record
8        x : Float := 0.0;
9        y : Float := 0.0;
10    end record;
11
12    type Segment is array (1 .. 2) of Point;
13
14    function length (s : Segment) return Float is
15    begin
16       return Sqrt ((s (1).x - s (2).x)**2 +
17                    (s (1).y - s (2).y)**2);
18    end length;
19
20    function length (a : Point; b : Point) return Float is
21    begin
22       return Sqrt ((a.x - b.x)**2 + (a.y - b.y)**2);
23    end length;
```


Notatki

Przykład w języku Ada

example.adb

```
25    a, b : Point;
26    c : Segment;
27 begin
28    a.x := -1.0;
29    a.y := -2.0;
30    b.x := 3.0;
31    b.y := 1.0;
32    c (1) := a;
33    c (2) := b;
34    Put_Line (Float'Image (length (a, b)));
35    Put_Line (Float'Image (length (c)));
36 end example;
```


Notatki

Język Python

Język Python nie ma tablic ani struktur. Zastępują je heterogeniczne listy i krotki (ale bez nazw pól).

Listy inicjujemy przez podstawienie `[]` (listy pustej) lub ciągu elementów w tych nawiasach. Elementy dodajemy metodą `append` - lista ma dynamiczną długość. Elementy są indeksowane od zera i możemy się do nich odwołać operatorem dostępu `[]`. Funkcja `len` na liście zwraca jej długość.

Notatki

Przykład użycia tablic: Sito Eratostenesa

Prosta metoda znajdowania liczb pierwszych na przedziale od 2 do ustalonego n.

Pseudokod algorytmu

```
1:  for i ← 2...n do
2:      sito[i] ← true
3:  end for
4:  for i ← 2...n do
5:      if sito[i] then
6:          j ← i + i
7:          while j ⩽ n do
8:              sito[j] ← false
9:              j ← j + i
10:         end while
11:     end if
12: end for
```

Po zakończeniu algorytmu p jest pierwsze jeśli $sito[p] = true$.

Notatki

Problem do implementacji

Przy pomocy sita Eratostenesa napisać funkcję która dla podanego n zwróci liczbę liczb pierwszych na przedziale od 2 do n.

Notatki

Implementacja w C

```c
#include <stdio.h>
#include <stdbool.h>

void compute_sieve(bool s[], unsigned n) {
  unsigned i, j;
  for ( i=2; i <= n; i++ ) s[i] = true;
  for ( i=2; i <= n; i++ )
    if ( s[i] )
      for ( j=i+i; j <= n; j+=i )
        s[j] = false;
}
unsigned count_primes(bool s[], unsigned n) {
  int i, c = 0;
  for ( i=2; i <= n; i++ )
    if ( s[i] ) c++;
  return c;
}
unsigned primenumbers(unsigned n) {
  bool sieve[n+1];
  compute_sieve(sieve, n);
  return count_primes(sieve, n);
}
```


Notatki

Implementacja w C

```c
int main() {
  unsigned n;

  scanf("%u", &n);
  printf("%u\n", primenumbers(n));

  return 0;
}
```


Notatki

Implementacja w Adzie

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
procedure PrimeNumbers is
   function PrimeNumbers (n : Natural) return Natural is
      type Sieve is array (2 .. n) of Boolean;

      procedure ComputeSieve (s : in out Sieve) is
         j : Natural;
      begin
         for i in s'Range loop
            s (i) := True;
         end loop;
         for i in s'First .. s'Last loop
            if s (i) then
               j := i + i;
               while j <= n loop
                  s (j) := False;
                  j := j + i;
               end loop;
            end if;
         end loop;
      end ComputeSieve;
```


Notatki

Implementacja w Adzie

```
      function CountPrimes (s : Sieve) return Natural is
         c : Natural := 0;
      begin
         for i in s'First .. s'Last loop
            if s (i) then
               c := c + 1;
            end if;
         end loop;
         return c;
      end CountPrimes;

      s : Sieve;
   begin
      ComputeSieve (s);
      return CountPrimes (s);
   end PrimeNumbers;

   n : Natural;
begin
   Get (n);
   Put_Line (Integer'Image (PrimeNumbers (n)));
end PrimeNumbers;
```


Notatki

Implementacja w Pythonie

```
def primenumbers(n) :
    def create_sieve(s, n) :
        for i in range(n+1) :
            s.append(True)
        s[0] = False
        s[1] = False
    def compute_sieve(s) :
        for i in range(2, len(s)) :
            if s[i] :
                j = i + i
                while j<len(s) :
                    s[j] = False
                    j = j + i
    def count_primes(s) :
        c = 0
        for e in s :
            if e :
                c = c + 1
        return c

    s = []
    create_sieve(s, n)
    compute_sieve(s)
    return count_primes(s)
```


Notatki

Implementacja w Pythonie

```
def main() :
    n = int(input(""))
    print( primenumbers(n) )

if __name__ == "__main__":
    main()
```


Notatki