Modularyzacja. Tablice dynamiczne

Wstęp do Informatyki i Programowania

Maciek Gębala

7 listopada 2024

Notatki

Modularyzacja

Podział kodu na osobne fragmenty mające logiczne uzasadnienie w działaniu (tworzenie bibliotek, pakietów)

Ułatwia pracę z kodem i ponowne użycie fragmentów oprogramowania.

Notatki

Modularyzacja w C

Plik nagłówkowy (końcówka `.h`) zawiera deklaracje funkcji do użycia. Odpowiadający mu plik z końcówką `.c` zawiera implementacje tych funkcji (z funkcjami pomocniczymi).

`#include "plik.h"` powoduje w czasie kompilacji wklejenie deklaracji funkcji z pliku `plik.h` i sprawdzenie czy są dobrze użyte. Połączenie skompilowanych plików następuje później.

Trzeba uważać aby kolejne wczytania pliku nagłówkowego nie spowodowały podwójnej deklaracji - `#pragma once`.

Notatki

Przykład w języku C

test.h

```
1 #pragma once
2
3 void jawne();
```

test.c

```
1 #include <stdio.h>
2
3 void tajne() { printf("Tajne\n"); }
4 void jawne() { printf("Jawne\n"); tajne(); }
```

main.c

```
1 #include "test.h"
2
3 int main() {
4   jawne();
5   tajne();
6
7   return 0;
8 }
```


Notatki

Modularyzacja z Adzie

Deklaracja pakietu funkcji/procedur zawarta jest w pliku z końcówką `.ads` i korzysta ze słowa kluczowego `package`. W deklaracji można wskazać części prywatne (niewidoczne przy korzystaniu z pakietu).

Implementacja jest w pliku z końcówką `.adb` i korzysta ze słów kluczowych `package body`.

Korzystanie z pakietu deklarujemy używają słowa kluczowego `with`. Słowo kluczowe `use` umożliwia pominięcie nazwy pakietu dla komponentu.

Notatki

Przykład w języku Ada

test.ads

```
1  package Test is
2     procedure Jawne;
3  private
4     procedure Tajne;
5  end Test;
```

test.adb

```
1  with Ada.Text_IO;
2  use Ada.Text_IO;
3
4  package body Test is
5     procedure Jawne is
6     begin
7        Put_Line ("Jawne");
8        Tajne;
9     end Jawne;
10    procedure Tajne is
11    begin
12       Put_Line ("Tajne");
13    end Tajne;
14 end Test;
```


Notatki

Przykład w języku Ada

main.adb

```
1  with Test;
2
3  procedure Main is
4  begin
5     Test.Jawne;
6  --   Test.Tajne;
7  end Main;
```

main1.adb

```
1  with Test; use Test;
2
3  procedure Main1 is
4  begin
5     Jawne;
6  --   Tajne;
7  end Main1;
```


Notatki

Modularyzacja w Pythonie

Implementacja w osobnym pliku tworzy moduł o nazwie takiej jak plik. Umieszczenie pliku w podkatalogu powoduje dodanie nazwy katalogu do nazwy modułu.

Moduł dodajemy słowem kluczowym `import` - różne odmiany dodawania w przykładach.

Notatki

Przykład w języku Python

test.py

```
1 def jawne() :
2     print("Jawne")
3     tajne()
4
5 def tajne() :
6     print("Tajne")
```

main.py

```
1 import test
2
3 def main() :
4     test.jawne()
5     test.tajne()
6
7 if __name__ == "__main__" :
8     main()
```


Notatki

Przykład w języku Python

main1.py

```
1 import test as t
2
3 def main() :
4     t.jawne()
5     t.tajne()
6
7 if __name__ == "__main__" :
8     main()
```

main2.py

```
1 from test import *
2
3 def main() :
4     jawne()
5     tajne()
6
7 if __name__ == "__main__" :
8     main()
```


Notatki

Czytanie parametrów wywołania programów

We wszystkich językach programowania istnieją metody/biblioteki umożliwiające czytanie parametrów wywołania programów z linii poleceń (jako ciąg napisów).

Notatki

Implementacja w C

params.c

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[]) {
4    printf("%d\n", argc);
5
6    for (int i = 0; i < argc; i++)
7      printf("%s\n", argv[i]);
8
9    return 0;
10 }
```

W języku C nie ma typu napisowego - zastępuje go tablica elementów typu `char`. Do operowania na napisach konieczna jest biblioteka `string.h` zawierające m.in. takie funkcje jak `strlen` czy `strcmp`.

Notatki

Implementacja w Adzie

params.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2  with Ada.Command_Line; use Ada.Command_Line;
3
4  procedure Params is
5  begin
6     Put_Line (Command_Name);
7     Put_Line (Argument_Count'Image);
8     for i in 1 .. Argument_Count loop
9        Put_Line (Argument (i)); -- funkcja, nie tablica
10    end loop;
11 end Params;
```


Notatki

Implementacja w Pythonie

params.py

```
1 import sys
2
3 def main() :
4     print(len(sys.argv))
5     for i in range(len(sys.argv)) :
6         print(sys.argv[i]);
7
8 if __name__ == "__main__":
9     main()
```


Notatki

Tablica dynamiczna w C

Do stworzenia tablicy dynamicznej w C potrzeba

- zadeklarowania wskaźnika na typ elementów tablicy (unarny operator `*`);
- użycia funkcji `malloc` do zarezerwowania odpowiedniego miejsca w pamięci;
- użycia funkcji `free` do zwolnienia pamięci, gdy już nie potrzebujemy tablicy.

Notatki

Implementacja w C

primenumbers.h

```
1 #pragma once
2 #include <stdbool.h>
3
4 void compute_sieve(bool s[], unsigned n);
5
6 unsigned count_primes(bool s[], unsigned n);
```


Notatki

Implementacja w C

primenumbers.c

```c
#include "primenumbers.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;
}
```


Notatki

Implementacja w C

primenumberstest.c

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

#include "primenumbers.h"

int main(int argc, char *argv[]) {
  unsigned long n, c;
  bool *s;

  if (argc != 2) {
    printf("Zła liczba argumentów\n");
    return -1;
  }
  sscanf(argv[1], "%lu", &n);
  s = malloc((n+1)*sizeof(bool));
  compute_sieve(s, n);
  c = count_primes(s, n);
  free(s);
  printf("%lu\n", c);
  return 0;
}
```


Notatki

Tablica dynamiczna w Adzie

Do stworzenia tablicy dynamicznej w Adzie potrzeba

- zadeklarowania typu wskaźnika na typ tablicy (słowo kluczowe access);
- zadeklarowania wskaźnika na typ tablicy;
- użycia operatora `new` do zarezerwowania odpowiedniego miejsca w pamięci;
- deklaracji i użycia funkcji `free` do zwolnienia pamięci, gdy już nie potrzebujemy tablicy.

Notatki

Implementacja w Adzie

primenumbers.ads

```
package PrimeNumbers is
   type Sieve is array (Positive range <>) of Boolean;
   type Sieve_Ptr is access Sieve;

   procedure ComputeSieve (s : Sieve_Ptr);

   function CountPrimes (s : Sieve_Ptr) return Natural;
end PrimeNumbers;
```


Notatki

Implementacja w Adzie

primenumbers.adb

```
1  package body PrimeNumbers is
2     procedure ComputeSieve (s : Sieve_Ptr) is
3        j : Natural;
4     begin
5        s.all := (others => True);
6        for i in s'Range loop
7           if s (i) then
8              j := i + i;
9              while j <= s'Last loop
10                s (j) := False; j := j + i;
11             end loop;
12          end if;
13       end loop;
14    end ComputeSieve;
15    function CountPrimes (s : Sieve_Ptr) return Natural is
16       c : Natural := 0;
17    begin
18       for i in s'Range loop
19          if s (i) then
20             c := c + 1;
21          end if;
22       end loop;
23       return c;
24    end CountPrimes;
25 end PrimeNumbers;
```


Notatki

Implementacja w Adzie

primenumberstest.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2  with Ada.Command_Line; use Ada.Command_Line;
3  with Ada.Unchecked_Deallocation;
4  with PrimeNumbers; use PrimeNumbers;
5
6  procedure PrimeNumbersTest is
7     procedure Free is
8        new Standard.Ada.Unchecked_Deallocation (Sieve, Sieve_Ptr);
9     n : Natural;
10    c : Natural := 0;
11    s : Sieve_Ptr;
12 begin
13    if Argument_Count /= 1 then
14       Put_Line ("Zła␣liczba␣argumentów");
15       return;
16    end if;
17    n := Natural'Value (Argument (1));
18    s := new Sieve (2 .. n);
19    ComputeSieve (s);
20    c := CountPrimes (s);
21    Free (s);
22    Put_Line (c'Image);
23 end PrimeNumbersTest;
```


Notatki

Implementacja w Pythonie

W Pythonie nie ma tablic a lista jest typem dynamicznym.

packages/primenumbers.py

```
1  def create_sieve(s, n) :
2      for _ in range(n+1) :
3          s.append(True)
4
5  def compute_sieve(s) :
6      for i in range(2, len(s)) :
7          if s[i] :
8              j = i + i
9              while j<len(s) :
10                 s[j] = False
11                 j = j + i
12
13 def count_primes(s) :
14     c = 0
15     for i in range(2, len(s)) :
16         if s[i] :
17             c = c + 1
18     return c
```


Notatki

Implementacja w Pythonie

main.py

```
1  import sys
2
3  import packages.primenumbers
4
5  def main() :
6      if len(sys.argv) != 2 :
7          print("Zła␣liczba␣argumentów")
8          return
9
10     n = int(sys.argv[1])
11     s = []
12
13     packages.primenumbers.create_sieve(s, n)
14     packages.primenumbers.compute_sieve(s)
15     c = packages.primenumbers.count_primes(s)
16     print(c)
17
18 if __name__ == "__main__":
19     main()
```


Notatki

Implementacja w Pythonie

main1.py

```
1  import sys
2
3  import packages.primenumbers as pp
4
5  def main() :
6      if len(sys.argv) != 2 :
7          print("Zła liczba argumentów")
8          return
9
10     n = int(sys.argv[1])
11     s = []
12
13     pp.create_sieve(s, n)
14     pp.compute_sieve(s)
15     c = pp.count_primes(s)
16     print(c)
17
18 if __name__ == "__main__":
19     main()
```


Notatki

Implementacja w Pythonie

main2.py

```
1  from sys import argv
2
3  from packages.primenumbers import *
4
5  def main() :
6      if len(argv) != 2 :
7          print("Zła liczba argumentów")
8          return
9
10     n = int(argv[1])
11     s = []
12
13     create_sieve(s, n)
14     compute_sieve(s)
15     c = count_primes(s)
16     print(c)
17
18 if __name__ == "__main__":
19     main()
```


Notatki

Notatki

Notatki