Programowanie

Sylwester Arabas

prowadzący ćwiczenia:
Magdalena Kuich, Krzysztof Piasecki, Łukasz Dulny

Wydział Fizyki
Uniwersytetu Warszawskiego

wykład XIV
10. czerwca 2015 r.

rozwiązanie zadań z testu: zad. 1/6

Połącz typy z wyrażeniami

`std::istream`	→	`std::cin`
`int`	→	`1/2`
`char`	→	`'a'`
`double`	→	`1/2.`
`bool`	→	`true`
`float`	→	`.5f`
`std::ostream`	→	`std::cout`

rozwiązanie zadań z testu: zad. 2/6

Wpisz w pole poniżej to co poniższy program wypisze na standardowe wyjście

```
#include <iostream>
#include <stdexcept>

int main()
{
+   std::cout << "a";
    try
    {
+     std::cout << "b";
-     throw std::runtime_error("c");
-     std::cout << "d";
    }
    catch (...)
    {
+     std::cout << "e";
    }
+   std::cout << "f";
}
```

rozwiązanie zadań z testu: zad. 3/6

Zaznacz, które z poniższych linijek są poprawnym (zaznacz OK), a które niepoprawnym (zaznacz KO) kodem C++

OK / KO `/* /* */ */`

OK / KO `bool main() { return false; }`

OK / KO `int main() { return sizeof(int); }`

OK / KO `int main() { return sizeof(double); }`

OK / KO `// f-cja main()`

rozwiązanie zadań z testu: zad. 4/6

Połącz wyrażenia z ich wartościami

`(sizeof(float) > sizeof(double))`	$\longrightarrow$	0
`0+4/2`	$\longrightarrow$	2
`2+4/3`	$\longrightarrow$	3
`sizeof(char)`	$\longrightarrow$	1
`'e'-'a'`	$\longrightarrow$	4
`abs(std::complex<float>(3,4))`	$\longrightarrow$	5

rozwiązanie zadań z testu: zad. 5/6

Zaznacz poniżej które spośród poniższych funkcji mogą posłużyć do podwajania wartości liczb w tablicy zdefiniowanej poza funkcją, a przekazanej jako argument

```
void fun1(std::vector<double> &tab)
{
  tab *= 2;
}
```

KO

```
void fun2(std::valarray<double> tab)
{
  tab *= 2;
}
```

KO

```
template <typename T>
void fun3(std::valarray<T> &tab)
{
  tab *= 2;
}
```

OK

rozwiązanie zadań z testu: zad. 6/6

Wypisz w których liniach (żadnej – wpisz kreskę, jednej lub wielu – podaj numer każdej z linii) znajdują się odp. części programu

```
 1 #include <boost/units/systems/si.hpp>
 2 #include <iostream>
 3
 4 int main()
 5 {
 6   namespace bu = boost::units;
 7   namespace si = boost::units::si;
 8
 9   bu::quantity<si::pressure>
10     p(1e5 * si::pascals);
11   bu::quantity<si::temperature>
12     T(300 * si::kelvins);
13   bu::quantity<si::mass_density>
14     rho(1 * si::kilograms / si::cubic_metres);
15
16   auto R = p / rho / T;
17   std::cout << R.value() << std::endl;
18 }
```

rozwiązanie zadań z testu: zad. 6/6

Wypisz w których liniach (żadnej – wpisz kreskę, jednej lub wielu – podaj numer każdej z linii) znajdują się odp. części programu

instrukcja warunkowa: —

```
1 #include <boost/units/systems/si.hpp>
2 #include <iostream>
3
4 int main()
5 {
6   namespace bu = boost::units;
7   namespace si = boost::units::si;
8
9   bu::quantity<si::pressure>
10    p(1e5 * si::pascals);
11  bu::quantity<si::temperature>
12    T(300 * si::kelvins);
13  bu::quantity<si::mass_density>
14    rho(1 * si::kilograms / si::cubic_metres);
15
16  auto R = p / rho / T;
17  std::cout << R.value() << std::endl;
18 }
```

rozwiązanie zadań z testu: zad. 6/6

Wypisz w których liniach (żadnej – wpisz kreskę, jednej lub wielu – podaj numer każdej z linii) znajdują się odp. części programu

```
 1 #include <boost/units/systems/si.hpp>
 2 #include <iostream>
 3
 4 int main()
 5 {
 6   namespace bu = boost::units;
 7   namespace si = boost::units::si;
 8
 9   bu::quantity<si::pressure>
10     p(1e5 * si::pascals);
11   bu::quantity<si::temperature>
12     T(300 * si::kelvins);
13   bu::quantity<si::mass_density>
14     rho(1 * si::kilograms / si::cubic_metres);
15
16   auto R = p / rho / T;
17   std::cout << R.value() << std::endl;
18 }
```

instrukcja warunkowa: —

deklaracja funkcji: 4

rozwiązanie zadań z testu: zad. 6/6

Wypisz w których liniach (żadnej – wpisz kreskę, jednej lub wielu – podaj numer każdej z linii) znajdują się odp. części programu

```
 1 #include <boost/units/systems/si.hpp>
 2 #include <iostream>
 3
 4 int main()
 5 {
 6   namespace bu = boost::units;
 7   namespace si = boost::units::si;
 8
 9   bu::quantity<si::pressure>
10     p(1e5 * si::pascals);
11   bu::quantity<si::temperature>
12     T(300 * si::kelvins);
13   bu::quantity<si::mass_density>
14     rho(1 * si::kilograms / si::cubic_metres);
15
16   auto R = p / rho / T;
17   std::cout << R.value() << std::endl;
18 }
```

instrukcja warunkowa: —

deklaracja funkcji: 4

instrukcje preprocesora: 1,2

rozwiązanie zadań z testu: zad. 6/6

Wypisz w których liniach (żadnej – wpisz kreskę, jednej lub wielu – podaj numer każdej z linii) znajdują się odp. części programu

```
 1 #include <boost/units/systems/si.hpp>
 2 #include <iostream>
 3
 4 int main()
 5 {
 6   namespace bu = boost::units;
 7   namespace si = boost::units::si;
 8
 9   bu::quantity<si::pressure>
10     p(1e5 * si::pascals);
11   bu::quantity<si::temperature>
12     T(300 * si::kelvins);
13   bu::quantity<si::mass_density>
14     rho(1 * si::kilograms / si::cubic_metres);
15
16   auto R = p / rho / T;
17   std::cout << R.value() << std::endl;
18 }
```

instrukcja warunkowa: —

deklaracja funkcji: 4

instrukcje preprocesora: 1,2

jawne wywołanie konstruktora: 10,12,14

rozwiązanie zadań z testu: zad. 6/6

Wypisz w których liniach (żadnej – wpisz kreskę, jednej lub wielu – podaj numer każdej z linii) znajdują się odp. części programu

```
 1 #include <boost/units/systems/si.hpp>
 2 #include <iostream>
 3
 4 int main()
 5 {
 6   namespace bu = boost::units;
 7   namespace si = boost::units::si;
 8
 9   bu::quantity<si::pressure>
10     p(1e5 * si::pascals);
11   bu::quantity<si::temperature>
12     T(300 * si::kelvins);
13   bu::quantity<si::mass_density>
14     rho(1 * si::kilograms / si::cubic_metres);
15
16   auto R = p / rho / T;
17   std::cout << R.value() << std::endl;
18 }
```

instrukcja warunkowa: —

deklaracja funkcji: 4

instrukcje preprocesora: 1,2

jawne wywołanie konstruktora: 10,12,14

odwołanie do metody (nie konstruktora): 17

rozwiązanie zadań z testu: zad. 6/6

Wypisz w których liniach (żadnej – wpisz kreskę, jednej lub wielu – podaj numer każdej z linii) znajdują się odp. części programu

```
 1 #include <boost/units/systems/si.hpp>
 2 #include <iostream>
 3
 4 int main()
 5 {
 6   namespace bu = boost::units;
 7   namespace si = boost::units::si;
 8
 9   bu::quantity<si::pressure>
10     p(1e5 * si::pascals);
11   bu::quantity<si::temperature>
12     T(300 * si::kelvins);
13   bu::quantity<si::mass_density>
14     rho(1 * si::kilograms / si::cubic_metres);
15
16   auto R = p / rho / T;
17   std::cout << R.value() << std::endl;
18 }
```

instrukcja warunkowa: —

deklaracja funkcji: 4

instrukcje preprocesora: 1,2

jawne wywołanie konstruktora: 10,12,14

odwołanie do metody (nie konstruktora): 17

deklaracja i inicjalizacja zmiennej: 9/10, 11/12, 13/14, 16

rozwiązanie zadań z testu: zad. 6/6

Wypisz w których liniach (żadnej – wpisz kreskę, jednej lub wielu – podaj numer każdej z linii) znajdują się odp. części programu

```
 1 #include <boost/units/systems/si.hpp>
 2 #include <iostream>
 3
 4 int main()
 5 {
 6   namespace bu = boost::units;
 7   namespace si = boost::units::si;
 8
 9   bu::quantity<si::pressure>
10     p(1e5 * si::pascals);
11   bu::quantity<si::temperature>
12     T(300 * si::kelvins);
13   bu::quantity<si::mass_density>
14     rho(1 * si::kilograms / si::cubic_metres);
15
16   auto R = p / rho / T;
17   std::cout << R.value() << std::endl;
18 }
```

instrukcja warunkowa: —

deklaracja funkcji: 4

instrukcje preprocesora: 1,2

jawne wywołanie konstruktora: 10,12,14

odwołanie do metody (nie konstruktora): 17

deklaracja i inicjalizacja zmiennej: 9/10, 11/12, 13/14, 16

deklaracja bez inicjalizacji zmiennej: —

standardy/style/konwencje kodowania w C++

Wybrane źródła

- Google C++ Style Guide (2014)
  http://google-styleguide.googlecode.com/svn/trunk/cppguide.html
- JSF Air Vehicle - C++ Coding Standards (2005)
  http://www.jsf.mil/downloads/documents/JSF_AV_C++_Coding_Standards_Rev_C.doc
- CERN C++ Coding Standard Specification (2000)
  http://pst.web.cern.ch/PST/HandBookWorkBook/Handbook/Programming/CodingStandard/c++standard.pdf

miejsce deklaracji / zasięg widoczności zmiennych

Google Place a function's variables in the narrowest scope possible, and initialize variables in the declaration

JSF Declarations should be at the smallest feasible scope

CERN Declare each variable with the smallest possible scope and initialise it at the same time

długość funkcji

Google If a function exceeds about 40 lines, think about whether it can be broken up without harming the structure of the program

JSF Any one function (or method) will contain no more than 200 logical source lines of code

CERN As a rule of thumb, remember the 7 ± 2 rule: typically methods should not be longer than 7 ± 2 statements

długość linii

Google Each line of text in your code should be at most 80 characters long

JSF Source lines will be kept to a length of 120 characters or less

wcięcia – tabulator czy spacje (i ile spacji)

Google Use only spaces, and indent 2 spaces at a time

JSF All indentations will be at least two spaces and be consistent within the same source file

CERN The code must be properly indented for readability reasons.

nawiasy klamrowe w blokach kodu z jedną instrukcją

Google In general, curly braces are not required for single-line statements...

JSF The statements forming the body of an if, else if , else, while, do...while or for statement shall always be enclosed in braces, even if the braces form an empty block

CERN Follow all flow control primitives (if, else, while, for, do, switch, and case) by a block, even if it is empty

```
 1 #include <random>
 2 #include <iostream>
 3
 4 int main()
 5 {
 6   std::random_device rdev;
 7   std::default_random_engine rng(rdev());
 8   std::uniform_int_distribution<int> dist(1, 6);
 9
10   int n = dist(rng);
11   if (n == 6)
12   {
13     std::cout << "Brawo!\n";
14   }
15   return n;
16 }
```

```
 1 #include <random>
 2 #include <iostream>
 3
 4 int main()
 5 {
 6   std::random_device rdev;
 7   std::default_random_engine rng(rdev());
 8   std::uniform_int_distribution<int> dist(1, 6);
 9
10   int n = dist(rng);
11   if (n == 6) std::cout << "Brawo!\n";
12   return n;
13 }
```

zmienne globalne

Google global variables of class type are forbidden ... global variables [...] should be rare in any case

JSF Unencapsulated global data will be avoided

CERN Do not declare global variables

```
 1 #include <sstream>
 2 #include <iostream>
 3
 4 int main()
 5 {
 6   std::ostringstream tmp;
 7
 8   tmp << "Hello";
 9   tmp << ", world!";
10   tmp << "\n";
11   std::cout << tmp.str();
12 }
```

```
 1 #include <sstream>
 2 #include <iostream>
 3
 4 std::ostringstream tmp;
 5
 6 int main()
 7 {
 8   tmp << "Hello";
 9   tmp << ", world!";
10   tmp << "\n";
11   std::cout << tmp.str();
12 }
```

formatowanie tekstu

Google Do not use streams, except where required by a logging interface. Use printf-like routines instead

CERN Use the iostream functions rather than those defined in stdio

```cpp
#include <iostream>

int main()
{
  std::cout << "Hello, world!\n";
}
```

```cpp
#include <cstdio>

int main()
{
  printf("Hello, world!\n");
}
```

wyjątki

Google We do not use C++ exceptions... Things would probably be different if we had to do it all over again from scratch.

JSF C++ exceptions shall not be used (i.e. throw, catch and try shall not be used.)

CERN Use exception handling instead of status values and error codes.

Dziękuję za uwagę!