Објектно оријентисано програмирање 2

О предмету

- Основа је курс CSCE 113 на Texas A&M универзитету, чији су наставници Бјарне Строуструп (Bjarne Stroustrup) и Тереза Леик (Teresa Leyk)
- Хвала им што су нам омогућили коришћење њихових материјала
- Корисна адреса: www.stroustrup.com/Programming

Организација наставе

- Предавања:
  - Претежно прате књигу: Bjarne Stroustrup, Programming – Principles and Practices Using C++“, Addison-Wesley, 2014
  - Пратећи слајдови на Интернет страници Одсека РТ-РК
- Лабораторија:
  - X вежби
  - На Интернет страници Одсека РТ-РК налазе се одговарајући материјали

Остале информације

- Фонд часова: 3+3
- Предавачи
  - Миодраг Ђукић (miodrag.djukic@rt-rk.uns.ac.rs)
  - Милош Суботић (milos.subotic@uns.ac.rs)
  - Душан Кењић (dusan.kenjic@rt-rk.uns.ac.rs)
- Бодовање:
  - 5% посећеност вежби +
  - 20% вежбе (5% домаћи + 15% у две вежбе које се оцењују) +
  - 25% пројекти (10% групни прој.+15% самостални прој.) +
  - 50% писмени испит +
  - Бонус (плусеви на предавањима и вежбама)

Сазнаћете од асистената

Какав је ово курс?

- Ово је курс програмирања...
  - ...за (релативне) почетнике који...
    - ...желе да постану професионалци...
      - тј. желе да праве системе које ће други људи користити
    - ...и имају вољу да напорно раде,...
      - иако треба и спавати и одмарати се током студија
  - ...коришћењем Це++ (C++) програмског језика.

Какав је ово курс?

- Али ово **није**:
- курс синтаксе и језичких заврзлама Це++ програмској језика!
- курс за студенте који нису у значајној мери самостални и немају самоиницијативу
  - Сами себи везујте пертле
  - Учите и вежбате зато што желите да радите на себи, а не зато што морате
- курс који пропагира непроверене методологије развоја софтвера и користи компликоване речи и бомбастичне изразе

Циљеви

- Учити и научити
  - Неке основне концепте програмирања (не само објектно оријентисаног)
  - Кључне технике
  - Главне могућности стандардног Це++-а
- Након (успешно) завршеног курса моћи ћете да:
  - Пишете необавезне Це++ програме
  - Читате (и разумете) много веће програме
  - Даље самостално учите
- Након курса нећете (бар не још увек) бити:
  - Искусни програмери
  - Стручњаци за Це++ језик
  - Искусни корисник напредних библиотека

Сарађујте у учењу

- Програмирање није самачки посао. (И програмери нису друштвено неприлагођени самотњаци који само „хакују“.)
- Одређене ствари се боље уче у друштву (иако се тиме не може надокнадити потпуни недостатак самоиницијативе)
- Али, није свака сарадња позитивна из позиције заједничког напредовања укључених страна
  - Не преузимајте од других код директно да бисте га без разумевања потурити као свој! Очекујте објашњење од колеге, јер то је једина права помоћ. Остало је само „скидање са врата“, и медвеђа услуга.
  - Не дајте другоме свој код да би га он потурио као свој!
- Оформите групе за заједничко учење
- Размењујте искуства
- Искористите консултације (дођите са припремљеним питањима)
- Питајте често (али стварно питајте – не очекујте да асистент ради вежбу уместо вас)

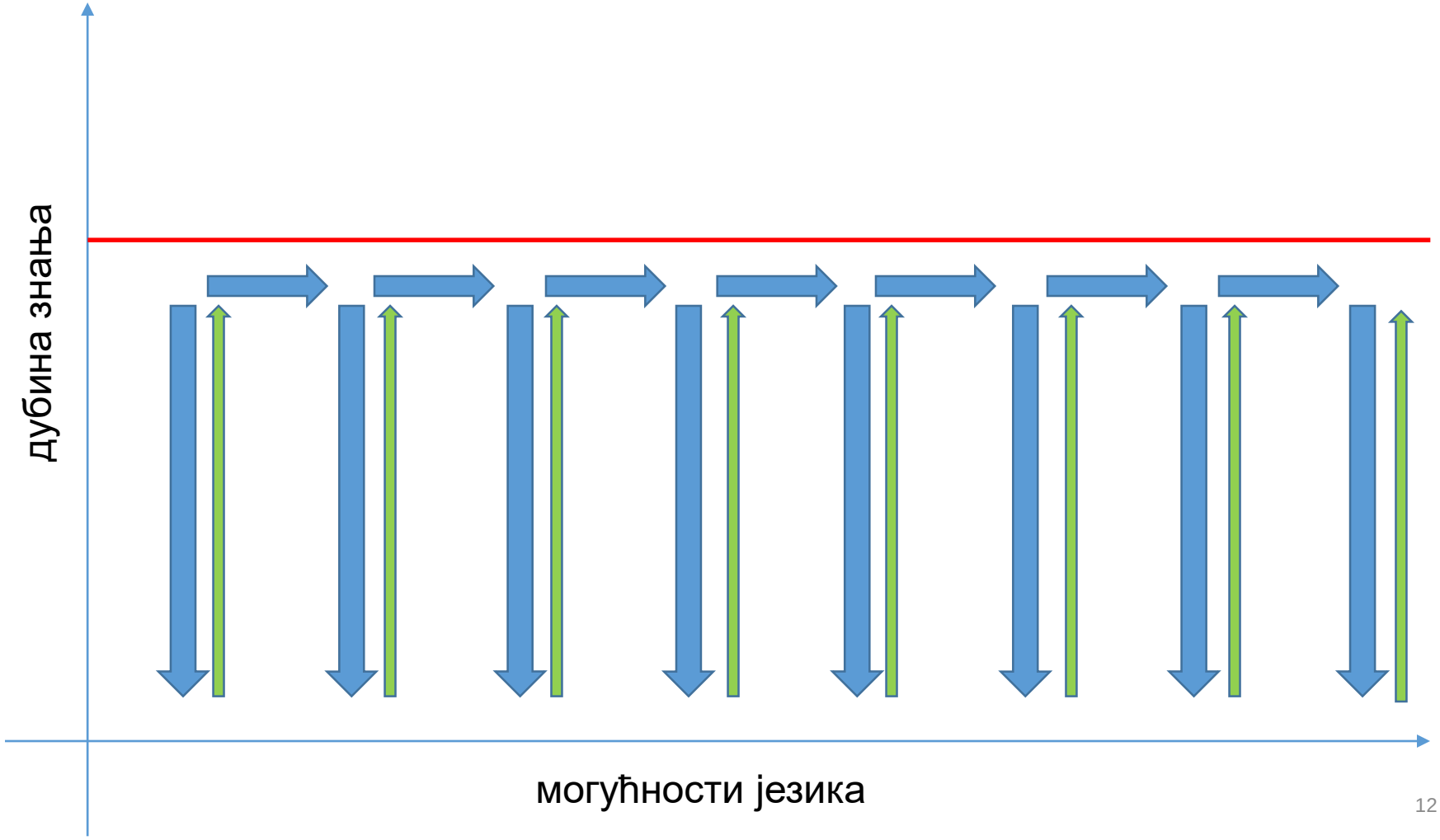
Ток курса

- Први део: Основе
  - Типови, променљиве, знаковни низови (стрингови), конзолни У/И, рачунање, грешке, класе...
- Други део: Улаз/излаз и елементи ООП
  - Улазно/излазни токови
  - Графичка корисничка спрега
- Трећи део: Структуре података и алгоритми у Це++
  - Динамичка меморија, показивачи, низови
  - Генеричко програмирање – шаблони
  - STL (стандардна библиотека шаблона)
  - Листе, мапе, вектори, уређивање, претрага...
- Четврти део: Ширење видика
  - Разне напредне теме

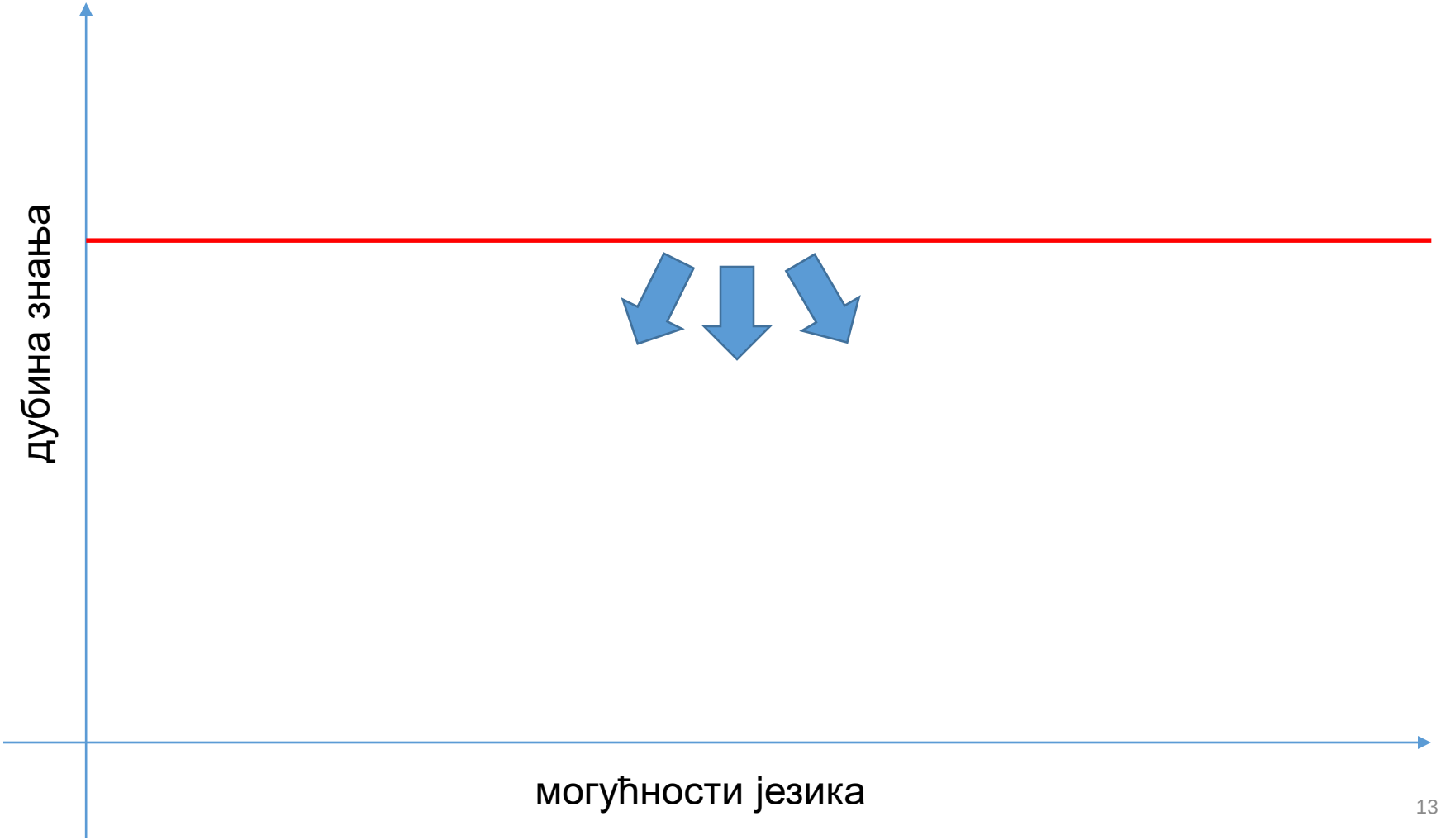

дубина знања
могућности језика

Могући приступи у учењу језика

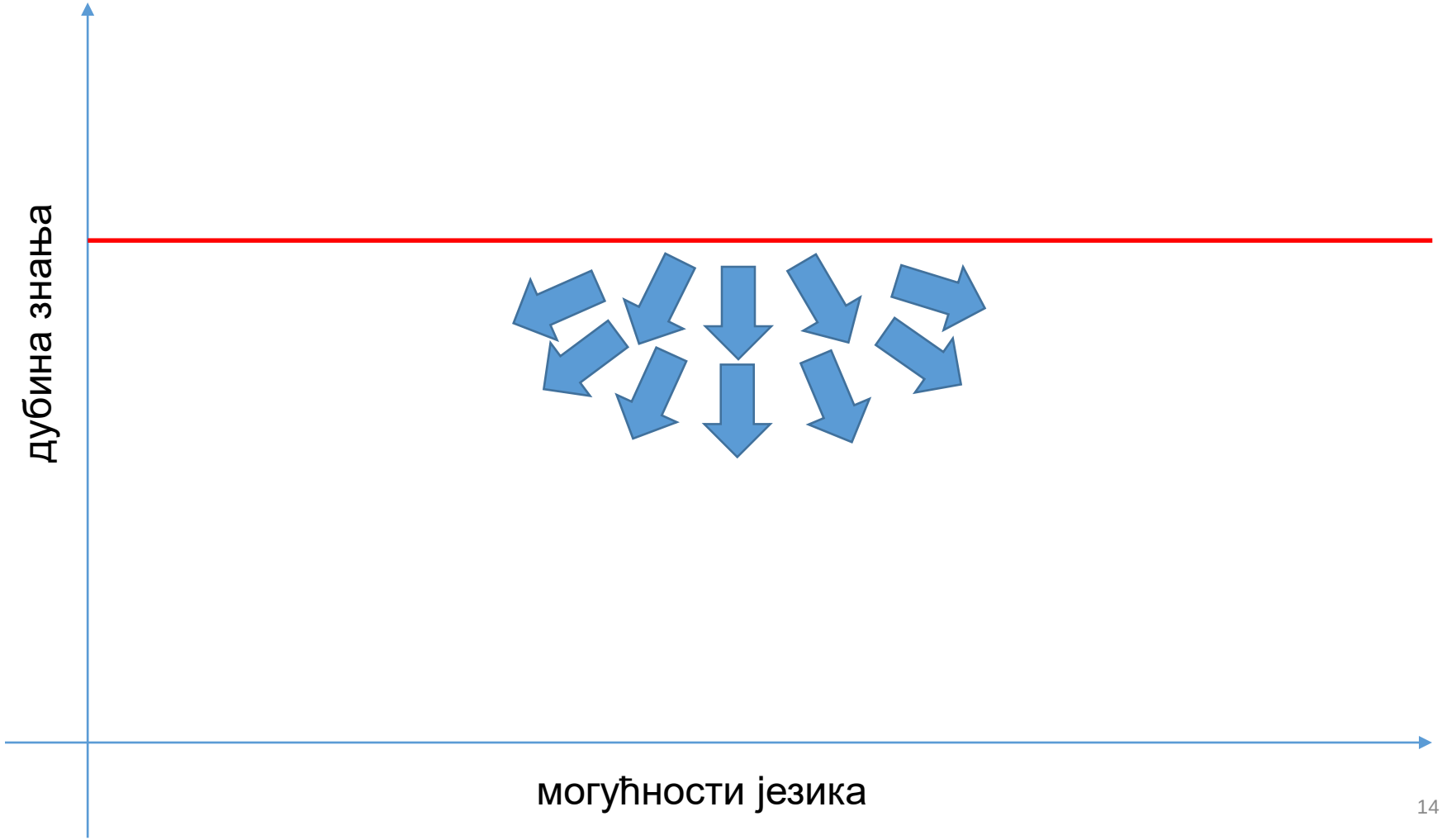
Могући приступи у учењу језика



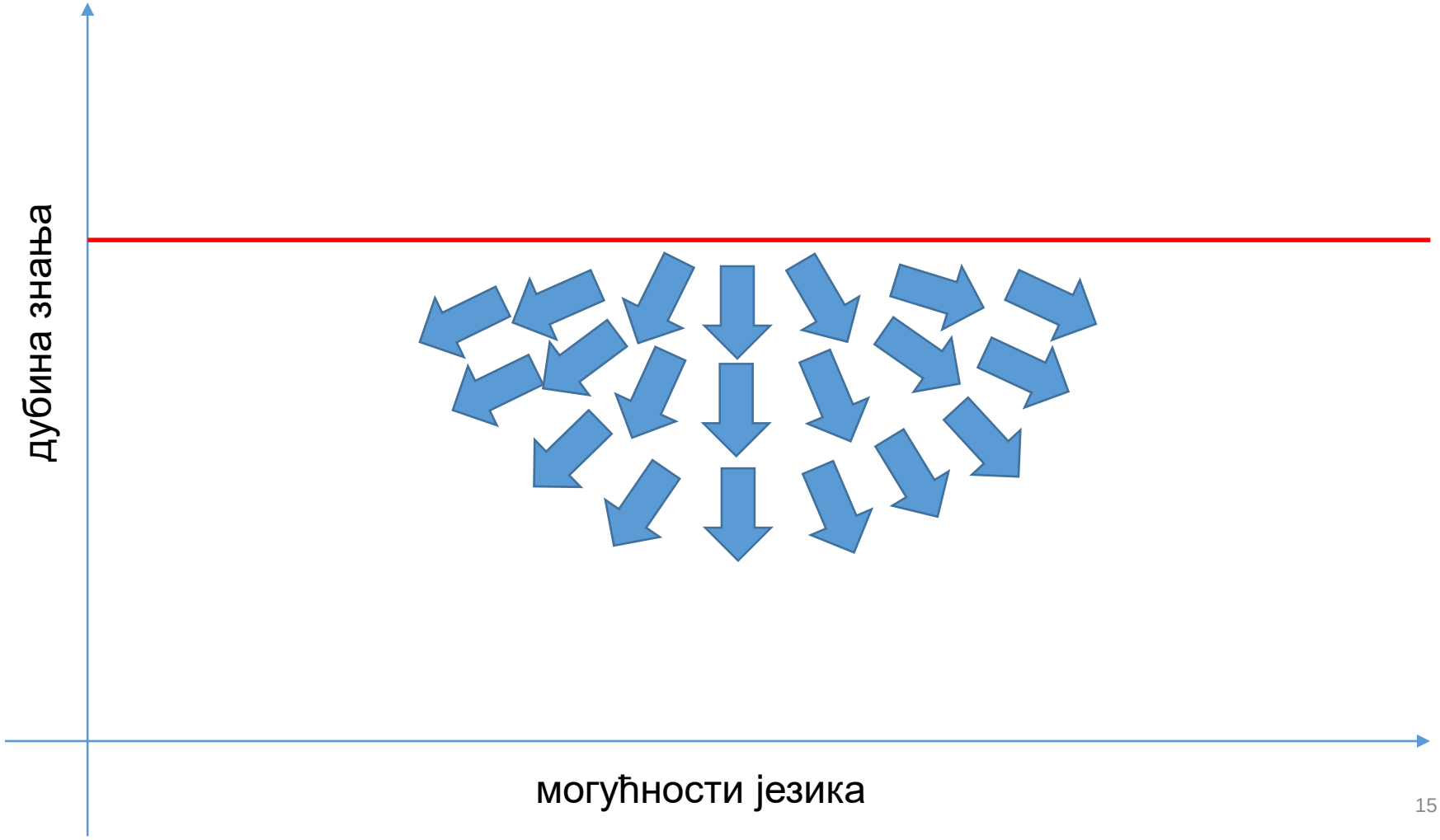
Приступ учењу језика на овом предмету



Приступ учењу језика на овом предмету



Приступ учењу језика на овом предмету

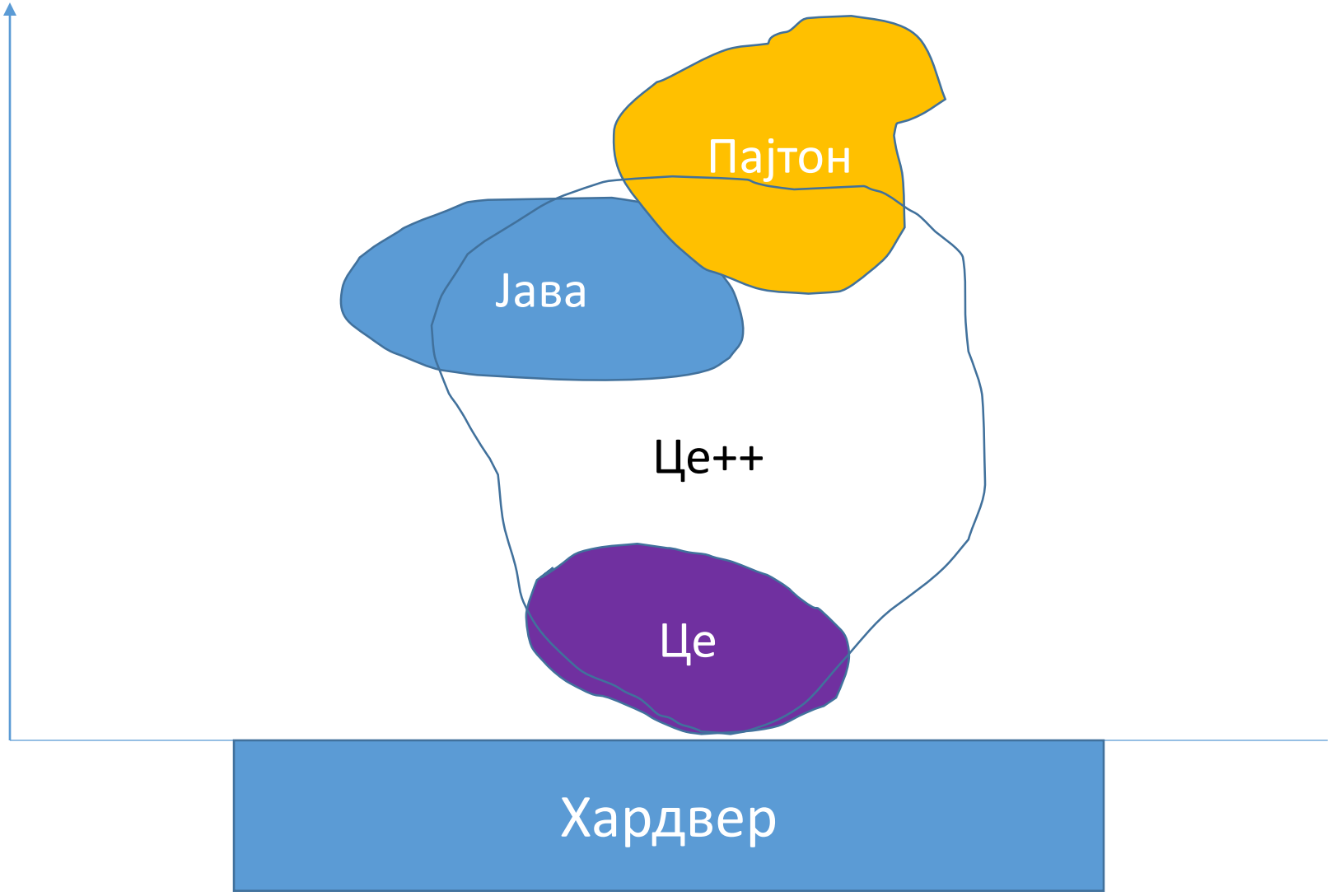


Зашто Це++?

- Це++ је прецизно и разумљиво дефинисан ISO стандардом
- Це++ је доступан на скоро свим рачунарима
- Це++ најраспрострањенији језик у инжењерским областима

  http://www.stroustrup.com/applications.html
- Це++ подржава **неколико** врло корисних програмерских парадигми: објектно оријентисано програмирање, генеричко програмирање, функционално програмирање...
- Це++ омогућава рад на различитим нивоима апстракције, од врло ниских, до врло високих.

Релативни ниво апстракције Це++-а



Апстракција – појам и употреба

- У суштини, то је процес занемаривања небитних особина, тј. уочавање и издвајање само битних особина.
- Сваки програм, па и најједноставнији, јесте врста апстракције.

Модел рачунара (који ћемо по некада користити за одређене илустрације)

- $r је неки регистар, $val је нека непосредна вредност

```
$r <- $val

mem[$val] <- $r
mem[$r] <- $r
mem[$r + $val] <- $r
$r <- mem[$addr]
$r <- mem[$r]
$r <- mem[$r + $val]

$r <- $r + $r
$r <- $r + $val
$r <- $r - $r
$r <- $r * $r

call $val
call $r
```

Апстракција – појам и употреба

```
5 + 7; // Врло конкретно
```

```
$r1 <- 5
$r2 <- 7
$r3 <- $r1 + $r2
```

Апстракција – појам и употреба

```
5 + 7; // Врло конкретно
```

```
$r1 <- 5
$r2 <- $r1 + 7
```

Апстракција – појам и употреба

```
5 + 7; // Врло конкретно          $r1 <- 12
```

Апстракција – појам и употреба

```
a + b; // ???                          ???
```

Апстракција – појам и употреба

```
a + b; // Превише апстракције   ???
```

Зашто је тип важан, тј. шта нам он говори?

Апстракција – појам и употреба

```
a + b; // Превише апстракције   ???
```

Зашто је тип важан, тј. шта нам он говори?

- Да ли је a + b уопште смислено (шта је смисао тог кода)
- Који код треба да се генерише

Апстракција – појам и употреба

0x86-64

```
int a, b;                          leal    (%rdi,%rsi), %eax
a + b;
```

Апстракција – појам и употреба

0x86-64

```
float a, b;
a + b;
```

```
addss   %xmm1, %xmm0
```

Апстракција – појам и употреба

```
5 + 7; // Врло конкретно


a + b; // Превише апстракције


int a; // a је цео број; Који цео број? Је л` број 7?
int b;
a + b;
```

Сада су вредности апстраховане.
a и b могу да имају било које вредности.
...
али операнди увек морају да буду баш те две променљиве...

Апстракција – појам и употреба

```
5 + 7; // Врло конкретно


a + b; // Превише апстракције


int a; // a је цео број; Који цео број? Је л` број 7?
int b; // ма онако „цео број“, то ти је апстракција
a + b;


int saberi(int x, int y)
{
  return x + y;
}

saberi(5, 7);
saberi(a, b);
saberi(c, d);
saberi(c, d*e);
```

Сада су и операнди апстраховани.

Апстракција – појам и употреба

```
int saberi(int x, int y); // сабира два цела броја
```

```
saberi(5, 7);
saberi(a, b);
saberi(c, d);
saberi(c, d*e);
```

Не можемо баш увек занемарити имплементацију, али врло често можемо и то нам олакшава размишљање.

```
int saberi(int x, int y)
{
  int i;
  if (y >= 0)
    for (i = 0; i <= y; ++i)
      x += 1;
  else
    for (i = 0; i <= -y; ++i)
      x -= 1;
  return x;
}
```

```
int saberi(int x, int y)
{
  return x + y;
}
```

Апстракција – појам и употреба

```
void foo(int** mat, int n, int m)
{
  int k;
  for (k = 0; k < m; k += 2)
  {
    int i;
    for (i = 0; i < (n - 1); i++)
    {
      int j;
      for (j = i + 1; j < n; j++)
      {
        if (mat[k][i] < mat[k][j])
        {
          int tmp;
          tmp = mat[k][i];
          mat[k][i] = mat[k][j];
          mat[k][j] = tmp;
        }
      }
    }
  }
  for (k = 1; k < m; k += 2)
  {
    int i;
    for (i = 0; i < n; i++)
    {
      mat[k][i] = 0;
    }
  }
}
```

Апстракција – појам и употреба

```
void foo(int** mat, int n, int m)
{
  int k;
  // иди кроз непарне редове (парни индекси)
  for (k = 0; k < m; k += 2)
  {
    // сортирај редове bubble sort алгоритмом
    int i;
    for (i = 0; i < (n - 1); i++)
    {
      int j;
      for (j = i + 1; j < n; j++)
      {
        if (mat[k][i] < mat[k][j])
        {
          int tmp;
          tmp = mat[k][i];
          mat[k][i] = mat[k][j];
          mat[k][j] = tmp;
        }
      }
    }
  }
  // иди кроз парне редове (непарни индекси)
  for (k = 1; k < m; k += 2)
  {
    // постави цео ред на 0
    int i;
    for (i = 0; i < n; i++)
    {
      mat[k][i] = 0;
    }
  }
}
```

Апстракција – појам и употреба

```
void foo(int** mat, int n, int m)
{
  int k;
  for (k = 0; k < m; k += 2)
  {
    sort(mat[k], n);
  }
  for (k = 1; k < m; k += 2)
  {
    zero(mat[k], n);
  }
}
```

```
void foo(Matrix& mat)
{
  int k;
  for (k = 0; k < mat.rowNo(); k += 2)
  {
    mat[k].sort(); //sort(mat[k]);
  }
  for (k = 1; k < mat.rowNo(); k += 2)
  {
    mat[k].reset();
    // reset(mat[k]);
    // mat[k] = 0;
  }
}
```

Апстракција – појам и употреба

```
def sortAndNull(matrix):
  for i in range(0,len(matrix)):
    if i%2 == 0:
      matrix[i].sort();
    else:
      for j in range(0,len(matrix[i])):
        matrix[i][j]=0;
return matrix
```

```
void foo(Matrix& mat)
{
  for (int k = 0; k < mat.rowNo(); ++k)
  {
    if (k % 2 == 0)
      mat[k].sort();
    else
      mat[k].reset();
  }
}
```

Важност стила кодирања

```
void foo(Matrix& mat)
{
  for (int k = 0; k < mat.rowNo(); ++k)
  {
    if (k % 2 == 0)
      mat[k].sort();
    else
      mat[k].reset();
  }
}
```

```
void foo(Matrix& mat) { for (int k
   = 0; k < mat.rowNo(); ++k) {
                       if
                (k %
      2 == 0) mat[k].sort(); else mat[k]
        .reset(); } }
```

Лингвистички нивои (нивои апстракције)

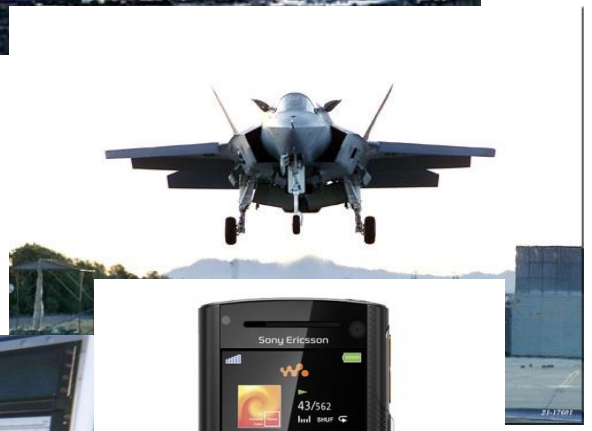
- Проширење: нове процедуре користе примитиве основног нивоа
- Превођење (компајлирање): са новог језика на језик основног нивоа
- Интерпретација: исто што и претходно, али су фаза превођења и извршавања временски зависне

За шта све програмирање служи?

- Програми су данас саставни део скоро сваке људске делатности и укључени су прављење скоро свих производа
- Сходно томе, потреба за програмирањем је у данашње време огромна
- Већина програма се извршава на рачунарима који уопште не личе на персонални рачунар
  - Екран, тастатура, кућиште... – уопште не морају постојати за те рачунаре

Бродови

Авиони



Телефони

Аутомобили

Наш први програм у Це++-у

```
// први програм:

#include "std_lib_facilities.h"   // укључити потребне библиотеке

int main()   // main() је почетна тачка извршавања програма
{
  cout << "Hello, world!\n"; // испис 13 знакова Hello, world!
                             // које прати прелазак у нови ред
  return 0; // повратна вредност која потврђује успешност
}
```

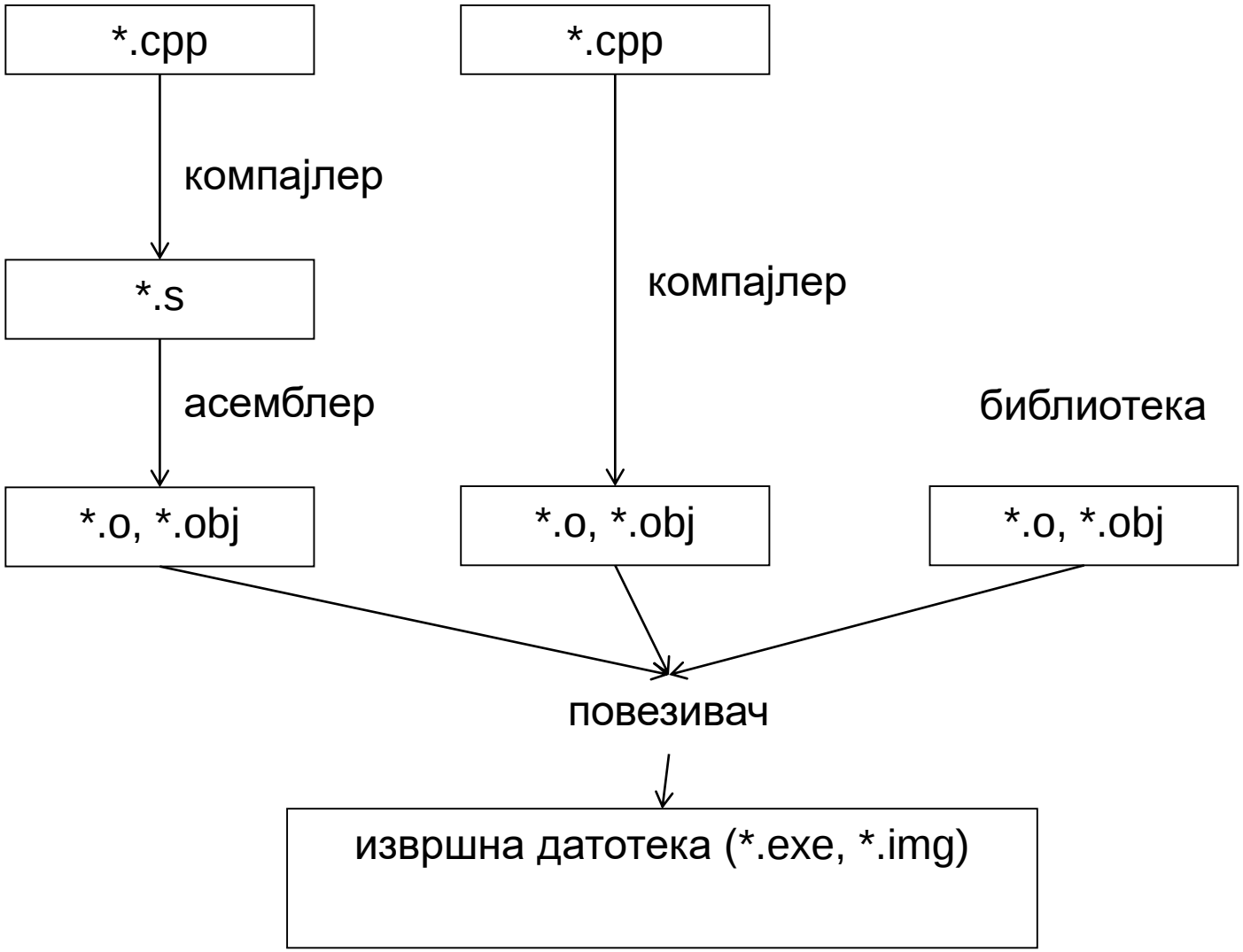
Здраво, свете!

- „Здраво, свете“ је важан први програм
  - Уз помоћ њега можете се упознати са радним окружењем и алатима
    - Компајлер
    - Интегрисано развојно окружење
  - Након што сте унели програм направите неколико намерних грешака да бисте видели како се окружење понаша; на пример:
    - Немојте укључити заглавље
    - Немојте затворити наводнике код знаковног низа (стринга)
    - Погрешите у куцању неке кључне речи (рецимо ***return***)
    - Изоставите тачка-зарез у неком реду.
    - Изоставите витичасте заграде
    - **…**

Здраво, свете!

- Већина кода је само подршка.
  - Само наредба **cout << "Hello, world!\n"** ради нешто суштински корисно
- То је потпуно нормално.
  - Већина кода који пишемо, и већима система које користимо, постоји само да би неки други део кода, или система, био ефикасан и елегантан.
- Није довољно да „одрадимо посао“. Није довољно да „наш програм ради“. Важно је да програм буде написан елегантно, разумљиво, исправно и да је користан.

Превођење (компајлирање) и повезивање



Превођење (компајлирање) и повезивање

- Програмер пише Це++ код
  - Изворни код би требао бити читљив људима (до разумне мере)
- Компајлер преводи тај код у објектни код (илити машински код)
  - Који је разумљив рачунарима
- Повезивач повезује све датотеке са објектним кодом које су потребне за прављење крајњег програма

- На овом курсу као развојно окружење користи се MS Visual Studio.