

Рачунање

Рачунање



- Улаz: тастатура, други уређаји, датотеке, други програми, други делови истог програма...
- Рачунање – оно шта наш програм ради да би на основу улаза створио одговарајуће излазе
- Излаз: екран, датотеке, други уређаји, други програми, други делови истог програма...

Рачунање

- Посао програмера је да изрази рачунање
 - Исправно
 - Једноставно
 - Ефикасно
- Два главна алата:
 - Подели па владај – разбити велико рачунање у више мањих
 - Апстракција – увести слојеве који скривају детаље и олакшавају размишљање
- Нагласак је на структури и организацији
 - Добар програм се не може добити само набацивањем гомиле исказа

Елементи језика

- Сваки елемент језика служи да би се изразила нека основна идеја, тј. концепт
 - На пример:
 - + : сабирање
 - * : множење
 - **if (*expression*) statement else statement ;** избор
 - **while (*expression*) statement ;** итерирање
 - **f(x);** функција/операција
 - ...
- Програмирање је комбиновање елемената језика да би се направили програми

Изрази

// срачунавање површине:

```
int length = 20;           // најједноставнији израз: литерал
```

```
int width = 40;
```

```
int area = length * width; // променљива је апстракција литерала
```

```
int average = (length + width) / 2;
```

Важе уобичајени (математички) приоритети операција:

a*b+c/d значи **(a*b)+(c/d)** а не, рецимо **a*(b+c)/d**.

Ако нисте сигурни, ставите у заграде.

Немојте писати прекомпликоване изразе, нпр.:

a*b+c/d*(e-f/g)/h+7

Изрази

- Изрази се сastoјe од операнада и операцијa
 - Операцијe одређујu шta требa да сe уради
 - Операнди одређујu сa чимe то требa да сe уради
- Логички (бул) тип: **bool** (**true** и **false**)
 - Једнакост: **= =** (једнако), **!=** (различито)
 - Логичке операцијe: **&&** (и), **||** (или), **!** (не)
 - Релацијe: **<** (мањe), **>** (већe), **<=**, **>=**
- Знаковни тип: **char** ('a', '7', '@'...)
- Интеџер (целобројни) тип: **short**, **int**, **long**
 - аритметичке операцијe: **+**, **-**, *****, **/**, **%** (остатак при дeљењu)
- Типови у покретном зарезу: e.g., **float**, **double**
 - аритметичке операцијe: **+**, **-**, *****, **/**

Сажети оператори

- За многе изразе Це++ нуди сажете операторе (приближно) истог значења
 - Пример:
 - $a += c$ значи $a = a + c$
 - $a *= scale$ значи $a = a * scale$
 - $++a$ значи $a += 1$
или $a = a + 1$
 - Сажети оператори су обично врло корисни: њима се често јасније и директније исказује идеја.

Искази

- Исказ је:
 - израз са ; на крају, или
 - декларација (са дефиницијом), или
 - „управљачка наредба“ која мења ток извршавања, или
 - блок исказа ({})
- На пример:
 - **a = b;**
 - **double d2 = 2.5;**
 - **if (x == 2) y = 4;**
 - **while (cin >> number) numbers.push_back(number);**
 - **int average = (length+width)/2;**
 - **return x;**

Избор

- Одређивање већег од два броја

```
if (a<b)
    max = b;
else
    max = a;
```

- Општа синтакса:

```
if (condition)
    statement-1 // ако је услов тачан, изврши ово
else
    statement-2 // у супротном, изврши ово
```

Итерирање (петље) вајл (енгл. while) петља

```
// срачунај и испиши квадрате свих бројева од 0 до 99:  
int main()  
{  
    int i = 0;  
    while (i < 100)  
    {  
        cout << i << '\t' << square(i) << '\n';  
        ++i ;  
    }  
}
```

Итерирање (петље) вајл (енгл. while) петља

- Уочавамо следеће делове:

- Управљачка променљива; **i**
- Иницијализација управљачке пром.; **int i = 0**
- Критеријум за прекид петље; **ако је i<100 је нетачно – заврши**
- Измена управљачке пром.; **++i**
- Нешто да се ради у свакој итерацији; **cout << ...**

```
int i = 0;
while (i < 100)
{
    cout << i << '\t' << square(i) << '\n';
    ++i ;
}
```

Итерирање (петље) фор (енгл. for) петља

- Све што је наведено на претходном слајду се сажима на једном месту, у једну наредбу, где се јасније види и боље разуме.

```
for (int i = 0; i < 100; ++i)
{
    cout << i << '\t' << square(i) << '\n';
}
```

Општи облик:

```
for (initialize; condition; increment)
    controlled statement
```

Итерирање (петље) фор (енгл. for) петља

- Али!

Лоше формиране фор петље

```
int i = 0;  
for (; i < 100;)  
{  
    cout << i << '\t' << square(i) << '\n';  
    ++i;  
}
```

```
int r = 0;  
for (int i = 0; r < 100; ++i)  
{  
    r = square(i);  
    cout << i << '\t' << r << '\n';  
    ++i;  
}
```

Итерирање (петље) фор (енгл. for) петља


```
int r = 0;
for (int i = 0; r < foo(r) ;)
{
    r = square(i);
    cout << i << '\t' << r << '\n';
    ++i;
}
```

Функције

- Шта је **square(i)**?

- Позив функције **square()**

```
int square(int x)
{
    return x*x;
}
```

- Функције уводимо када желимо да одвојимо део рачунања.
 - Зашто бисмо то желели да радимо? Зато што:
 - Тиме издавамо логичку целину
 - Чинимо програмски код читљивијим (именовањем дела посла)
 - Тај део онда можемо искористити на више места у програму
 - То олакшава испитивање, поделу послана и одржавање

Функције

- Функција **square()**

```
int square(int x)
{
    return x*x;
}
```

је конкретизација општег облика

Return_type function_name (Parameter list) // са декларацијом
{
 // употреба параметара у коду
 return some_value; // типа *Return_type*
}

Још један пример функције

- Функција која враћа већи од два броја.

```
int max(int a, int b) // прима два параметра
{
    if (a < b)
        return b;
    else
        return a;
}

int x = max(7, 9);
int y = max(19, -27);
int z = max(20, 20);
```

Потпис функције

- Променљиву јединствено одређује њено име.
- Али код функција то није случај, тј. може постојати више различитих функција које се исто зову!

```
void foo(int a);
```

```
long foo(float a);
```

```
void foo(int a, int  
b);
```

- Функцију једнозначно идентификује: 1. њено име, 2. број параметара, 3. типови параметара.
- То називамо потписом функције.
- Повратна вредност није део потписа.

Потпис функције

- Како онда компајлер (преводилац) зна коју функцију треба да позове на неком месту?
- Важна чињеница је то што је Це++ статички типизиран језик.
- Тип свих параметара се зна током превођења.

```
int x, y;
float z;

foo(x); // ?
foo(y, 7); // ?
foo(z); // ?
foo(); // ?
foo(y, x, z); // ?
```

```
void foo(int a);
long foo(float a);
void foo(int a, int b);
```

Потпис функције

- Када имамо више различитих функција истог назива кажемо да се оне преклапају.
- Дефинисање нове функције `foo` представља ново преклапање.
- Скуп свих функција са називом `foo` је „скуп преклапајућих функција“, или само „преклапајући скуп“.

```
void foo(int a);
```

```
long foo(float a);
```

```
void foo(int a, int  
b);
```

```
...
```

Потпис функције

- Зашто је ово важно?
- Врло ситан пример, за сада:

```
float sqrt(float x);  
  
int sqrt(int x);  
  
double sqrt(double x);  
  
...  
  
sqrt(a);
```

- Иако су функције различите, све носе исти назив и подједнако природно се позивају.

Скупови података - Вектор

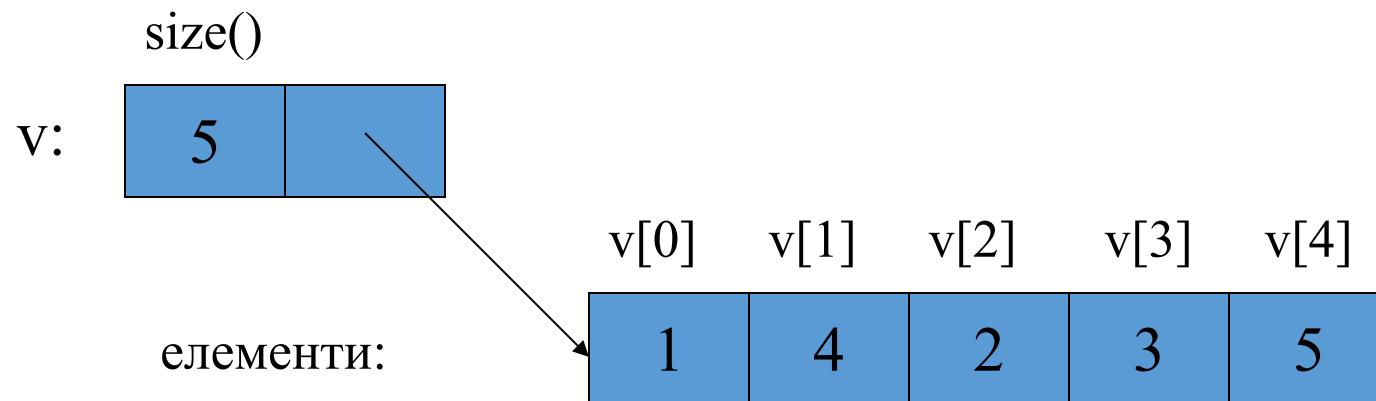
- За сваки озбиљнији програмски посао треба нам скуп података над којим се нешто ради. За ту сврху врло је згодан тип вектора: **vector**. На пример:

```
// учитај мерења температуре у вектор:  
int main()  
{  
    vector<double> temps; // декларација вектора (уређеног скупа)  
                           // елемената типа double  
    double temp;          // једна променљива типа double  
    while (cin >> temp)  
        temps.push_back(temp);  
    // ...  
}  
// cin >> temp враћа true докле год не дођемо до краја  
датотеке или не нађемо на нешто што није запис реалног  
броја; нпр. реч "end"
```

Вектор

- Вектор је најупотребљаванији тип из стандардне библиотеке
 - `vector<T>` садржи низ објеката типа `T`

Вектор имена `v` садржи 5 елемената: {1, 4, 2, 3, 5}.



Вектор

`vector<int> v;` // настаје празан

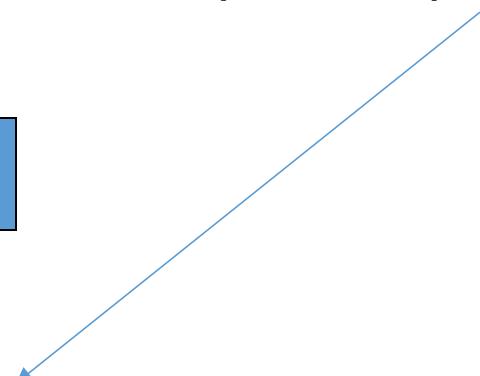


Елементи иду један иза другог, без рупа. Величина елемента (величина типа T, int у овом случају) је важна.

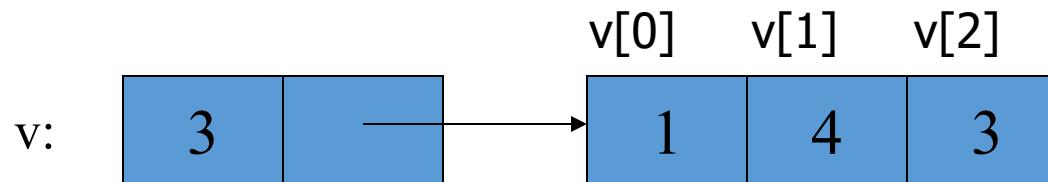
`v.push_back(1);`



`v.push_back(4);`



`v.push_back(3);`



Вектор

- Када су подаци смештени у вектор, можемо једноставно радити са њима:

```
// срачунавање средње и медиан температуре:  
int main()  
{  
    vector<double> temps;  
    double temp;  
    while (cin >> temp) temps.push_back(temp);  
  
    double sum = 0;  
    for (int i = 0; i < temps.size(); ++i)  
        sum += temps[i];  
  
    cout << "Prosečna temperatura: " << sum/temps.size() << endl;  
  
    sort(temps.begin(), temps.end());  
    cout << "Median temperatura: " << temps[temps.size()/2] << endl;  
}
```

Комбинање језичких елемената

- Многи програми се могу написати комбинањем споменутих елемената језика, урађених и библиотечких типова.
 - До сада смо споменули
 - Променљиве и литерале типа **bool, char, int, double**
 - **string**
 - **vector, push_back(), []** (индексирање)
 - **!=, ==, =, +, -, +=, <, &&, ||, !**
 - **max(), sort(), cin>>, cout<<**
 - **if, for, while**

Пример – Листа речи

```
vector<string> words;
string s;
while (cin >> s && s != "quit")
    words.push_back(s);

sort(words.begin(), words.end()); // уреди речи које су прочитане

for (int i = 0; i < words.size(); ++i)
    cout << words[i] << "\n";
```

Листа речи – Уклони дупликате

```
vector<string> words;
string s;
while (cin >> s && s != "quit")
    words.push_back(s);

sort(words.begin(), words.end());
"get rid of duplicates in words"      // (псеудо код)
```

Како ћемо уклонити дупликате користећи само оно што смо до сада урадили? Који алгоритам ћемо употребити?

Листа речи – Уклони дупликате

```
// једно могуће решење:  
  
vector<string> words;  
  
string s;  
  
while (cin >> s && s != "quit")  
    words.push_back(s);  
  
sort(words.begin(), words.end());  
  
vector<string> w2;  
  
if (0 < words.size())  
{  
    w2.push_back(words[0]);  
    for (int i = 1; i < words.size(); ++i)  
        if (words[i - 1] != words[i])  
            w2.push_back(words[i]);  
}  
  
cout << "found " << words.size() - w2.size() << " duplicates\n";  
for (int i = 0; i < w2.size(); ++i)  
    cout << w2[i] << "\n";
```

Још неке форме фор петље

```
for (int i = 0; i < w2.size(); ++i) cout << w2[i] << "\n";
```

- Омогућава јасније истицање шта се заправо овде дешава (а шта се то дешава?):

```
for (string it : w2) cout << it << "\n";
```

```
for (auto it : w2) cout << it << "\n";
```

- Када:
 - идемо од почетка до краја
 - идемо један по један