

# Писање програма „Калкулатор“

# Стратегија писања програма

- Шта је проблем који програм треба да реши?
  - Да ли је формулација проблема јасна?
  - Да ли је проблем решив (у задатом року, са доступним знањима, алатима и ресурсима)?
  - Шта су кључни потпроблеми?
- Поделити у мање делове
  - Да ли знамо неке алате, библиотеке и сл. који би били од помоћи?
    - За нас почетнике, и ово се рачуна: **iostream**, **vector**, итд.
- Написати малу, ограничену верзију програма, која решава кључне (пот)проблеме
  - Имплементација помаже разумевању проблема
- Ако се уз пут заглибите, вратите се корак назад (или на почетак)
  - Све док не добијемо општи приступ са којим смо задовољни
- На основу ограничене верзије напишите потпуну верзију
  - Обично коришћењем делова ограничене верзије, али не увек

# Програмирање је вештина

- Учи се кроз вежбање и примере
  - Не само кроз учење неких општих принципа
  - Не само кроз познавање језика и његових појединости
- Потребно је искуство
  - Потребно је много неуспеха – никоме не успе из прве
  - Вожња бицикла се не учи (само) читањем књиге

## Пример писања програма

- У наредних неколико часова писаћемо један програм од почетка. Уз пут ћемо правити грешке и постепено их поправљати, унапређујући наш програм.
  - Програми се ретко када праве одједном, већ постепено расту. Програми се не праве, већ узгајају.

# Калкулатор

- Корисник задаје математички израз преко тастатуре, а програм га срачунава и исписује резултат на екран:
  - На пример:
    - Израз:  $2+2$
    - Резултат: 4
    - Израз :  $2+2*3$
    - Резултат : 8
    - Израз :  $2+3-25/5$
    - Резултат : 0

# Псеудо код

- Прва идеја:

```
int main()
{
    variables
    while (get a line) {          // шта је линија?
        analyze the expression // шта ово значи?
        evaluate the expression
        print the result
    }
}
```

- Како ћемо представити **45+5/7** као податак?
- Како ћемо наћи **45 + 5 /** и **7** у улазном стрингу?
- Како ћемо постићи да **45+5/7** значи **45+(5/7)** а не **(45+5)/7**?
- Можемо ли имати променљиве? **v=7; m=9; v\*m**

# Током решавања задатка

- Наилазићемо на две врсте проблема:
  - Проблеми у поступку (алгоритамски проблеми)
    - Каквим поступком да одговарајуће решимо проблем?
    - Ту нам помажу разна знања из рачунарске науке и сродних области
  - Проблеми у изражавању поступка
    - Како да жељени поступак што лепше и једноставније изразимо?
    - Ту нам помаже познавање програмској језика и добра програмерска пракса
- Кроз суочавање са проблемом који одређени елемент језика адресира, боље се увиђа смисао и начин употребе тог елемента.

# Почнимо од једноставног потпроблема

- Сви изрази су у форми:

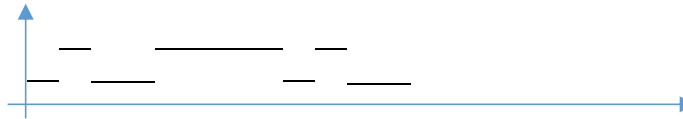
$$lval \ op \ rval$$

- Где су  $lval$  и  $rval$  реални бројеви
- А  $op$  је операција  $+$  или  $-$



# Нивои апстракције

Напон



Нуле и јединице

0100111101001111010100000011001000100000

Знакови

01001111 01001111 01010000 00110010 00100000  
'O' 'O' 'P' '2' ' '

Објекти

0100111101001111010100000011001000100000  
"OOP2"  
0011001000100000  
'2'+ ' ' или 2 (интеџер) или "2" (стринг)

## Додајемо ствари

- Сви изрази су у форми:

`lval op rval`

- Где су `lval` и `rval` реални бројеви
- А `op` је операција `+`, `-`, `*` или `/`

# Додајемо ствари

- Сви изрази су у форми:

`val1 op1 val2 op2 val3 ...`

- Где су `valN` реални бројеви
- А `opN` је операција `+`, `-`, `*` или `/` - операције су истог приоритета
- Са `;` се означава крај израза

## Додајемо ствари

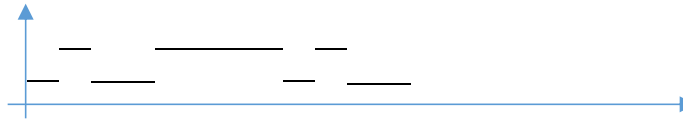
- Сви изрази су у форми:

`val1 op1 val2 op2 val3 ...`

- Где су `valN` реални бројеви
- А `opN` је операција `+`, `-`, `*` или `/` - операције су истог приоритета
- Са `;` се означава крај израза
- У изразу могу постојати заграде – али их само треба игнорисати

# Нивои апстракције

Напон



Нуле и јединице

0100111101001111010100000011001000100000

Знакови

01001111 01001111 01010000 00110010 00100000  
'O' 'O' 'P' '2' ' '

Објекти

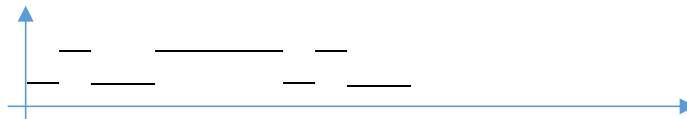
0100111101001111010100000011001000100000  
"OOP2"

0011001000100000

'2' + ' ' или 2 (интеџер) или "2" (стринг)

# Симболи (токени)

Напон



Нуле и јединице

0100111101001111010100000011001000100000

Знакови

01001111 01001111 01010000 00110010 00100000  
 'O' 'O' 'P' '2' ' ' ' '

Објекти

0100111101001111010100000011001000100000  
 "OOP2"  
 0011001000100000  
 '2'+ ' ' или 2 или "2"

Симболи (токени)

0100111101001111010100000011001000100000

врста	var.	врста	oper.	врста	num.
вредност	"O"	вредност	+	вредност	2

# Нови тип: симбол (токен)

## Кориснички тип (Апстрактни тип)

```
struct Token {  
    char kind;  
    double value;  
};
```

или

```
class Token {  
public:  
    char kind;  
    double value;  
};
```

(1.5+4)\*11

\ '('	\ '8'	\ '+'	\ '8'	\ ')'	\ '*'	\ '8'
	1.5		4			11

# Кориснички (апстрактни) тип

- Концепт апстрактног (корисничког) типа, тј. апстракције података, је кључан у развоју парадигме објектно оријентисаног програмирања.
- Али, то је засебан концепт.
- Обично је ОО надкуп апстракције података, у смислу да постоје језици који подржавају апстракцију података, али не и ООП.
- Дакле, није ООП све што има class.



# Објекат, тип, променљива

- Неки основни појмови - подсећање:
  - Објекат је парче меморије у којем је смештена вредност неког типа
  - Променљива је објекат који има име (и којем се, због тога, можемо обраћати директно)
  - Тип одређује скуп вредности које објекат може да има и скуп операција које се над тим вредностима могу извршавати

# Тип

- Скуп вредности
- Главни поступак је композиција других типова.
- На почетку имамо уграђене, тј. основне типове.
- Кључне речи struct или class

```
class Token {  
public:  
    char kind;  
    double value;  
};
```

- Али, не морају увек све комбинације свих вредности елементарних типова бити ваљане вредности корисничког типа.

# Тип

- Скуп операција
- Главни поступак је увођење функција.
- Али, које операције нам требају?
  - Зависи шта желимо да радимо са објектима тог типа.

```
class Token {  
public:  
    char kind;  
    double value;  
};
```

# Тип

- Скуп операција
- Али, које операције нам требају?
  - Зависи шта желимо да радимо.
- Нпр.:

```
void foo() {  
    MyType x;  
    MyType y;  
    add(x, y) ;  
}
```

# Тип

- Скуп операција
- Али, које операције нам требају?
  - Зависи шта желимо да радимо.
- Нпр.:

```
void foo() {  
    int x;  
}
```

```
void foo() {  
    Token x;  
}
```

# Тип

- Скуп операција
- Али, које операције нам требају?
  - Зависи шта желимо да радимо.
- Нпр.:

```
void foo() {  
    int x;  
}
```

```
void foo() {  
    int x = 5;  
}
```

```
void foo() {  
    Token x;  
}
```

# Тип

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)

```
void foo() {      class Token {  
    Token x;      public:  
}                Token() {}  
                  char kind;  
                  double value;  
                };
```

Конструктор је функција (чланица). Специјална функција, али и даље функција.

# Тип

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)

```
void foo() {  
    Token x;  
}  
  
class Token {  
public:  
    Token() : kind('x'), value(0.0) {}  
    char kind;  
    double value;  
};  
  
class Token {  
public:  
    Token() { kind = 'x'; value = 0.0; }  
    char kind,  
    double value;  
};
```



# Тип

- Скуп операција
  - Које операције нам требају?
  - Операција стварања променљиве нам је увек потребна. (конструктор)
  - ?
- 
- Наставићемо мало касније...
  - Сад се враћамо на задатак 😊

# Додајемо ствари

- Сви изрази су у форми:

`val1 op1 val2 op2 val3 ...`

- Где су `valN` реални бројеви
- А `opN` је операција `+`, `-`, `*` или `/` - операције су истог приоритета
- Са `;` се означава крај изрази
- У изразу могу постојати заграде – и треба прво да се срачунају изрази у њима - ??? како ово ???

# Корак назад

## Формалне граматике

- Како би описали наш израз (без заграда)?
- Шта су могући операнди? Назовимо то **Number**.

$N \rightarrow \text{real number (double prec. floating point num.)}$

- Да ли су ово изрази које ми прихватамо?

$N + N, N - N, N + N * N, N - N - N, N * - N N$

- Ево правила која описују ваљане изразе (**Expression**):

1.  $E \rightarrow N$

2.  $E \rightarrow E + N$

3.  $E \rightarrow E - N$

4.  $E \rightarrow E * N$

5.  $E \rightarrow E / N$

$E$   
 $E + N$  (2)  
 $N + N$  (1)

$E$   
 $E * N$  (4)  
 $E + N * N$  (2)  
 $N + N * N$  (1)

# Формалне граматике

- Па да пробамо ово да имплементирамо:

$E \rightarrow N$

$E \rightarrow E + N$

$E \rightarrow E - N$

$E \rightarrow E * N$

$E \rightarrow E / N$

Синтаксна анализа са рекурзивним спуштањем.

# Формалне граматике

- Па да пробамо ово да имплементирамо:

~~$E \rightarrow N$~~

~~$E \rightarrow E + N$~~

~~$E \rightarrow E - N$~~

~~$E \rightarrow E * N$~~

~~$E \rightarrow E / N$~~

$E \rightarrow N$

$E \rightarrow N + E$

$E \rightarrow N - E$

$E \rightarrow N * E$

$E \rightarrow N / E$

Синтаксна анализа са рекурзивним спуштањем.  
Проблем леве рекурзије.

# Формалне граматике

- Ипак ћемо овако писати, али имплементација је мало другачија:

$E \rightarrow N$

$E \rightarrow E + N$

$E \rightarrow E - N$

$E \rightarrow E * N$

$E \rightarrow E / N$

# Формалне граматике

- Уводимо заграде.
- Увешћемо нови нетерминални симбол: P (**P**rimary)

$E \rightarrow P$

$E \rightarrow E + P$

$E \rightarrow E - P$

$E \rightarrow E * P$

$E \rightarrow E / P$

$P \rightarrow N$

$P \rightarrow ( E )$

$N \rightarrow \text{Real number}$

# Додајемо ствари

- Сви изрази су у форми:

`val1 op1 val2 op2 val3 ...`

- Где су `valN` реални бројеви
- А `opN` је операција `+`, `-`, `*` или `/` - операције су истог приоритета
- Са `;` се означава крај израза
- У изразу могу постојати заграде – и треба прво да се срачунају изрази у њима





# Додајемо ствари

- Сви изрази су у форми:

`val1 op1 val2 op2 val3 ...`

- Где су `valN` реални бројеви
- А `opN` је операција `+`, `-`, `*` или `/` - **`*` и `/` су приоритетније**
- Са `;` се означава крај израза
- У изразу могу постојати заграде – и треба прво да се срачунају изрази у њима

# Формалне граматике

- Заграде раде!!!
- Што наводи како да уведемо приоритете операција
- Увешћемо још један нетерминални симбол: T (Term)

$E \rightarrow T$	$T \rightarrow P$	$P \rightarrow N$	$N \rightarrow \text{Real number}$
$E \rightarrow E + T$	$T \rightarrow T * P$	$P \rightarrow ( E )$	
$E \rightarrow E - T$	$T \rightarrow T / P$		

# Додајемо ствари

- Сви изрази су у форми:

`val1 op1 val2 op2 val3 ...`

- Где су `valN` реални бројеви
- А `opN` је операција `+`, `-`, `*` или `/` - `*` и `/` су приоритетније
- Са `;` се означава крај изрази
- У изразу могу постојати заграде – и треба прво да се срачунају изрази у њима



## Додајемо ствари

- Сви изрази су у форми:

`val1 op1 val2 op2 val3 ...`

- Где су `valN` реални бројеви – могу почињати са `.`, `+` или `-`
- А `opN` је операција `+`, `-`, `*` или `/` - `*` и `/` су приоритетније
- Са `;` се означава крај израза
- У изразу могу постојати заграде – и треба прво да се срачунају изрази у њима

# Додајемо ствари

- Сви изрази су у форми:

`val1 op1 val2 op2 val3 ...`

- Где су `valN` реални бројеви – могу почињати са `.`, `+` или `-`
- А `opN` је операција `+`, `-`, `*`, `/` или `%` - `*`, `/` и `%` су приоритетније
- Са `;` се означава крај изрази
- У изразу могу постојати заграде – и треба прво да се срачунају изрази у њима
- Омогућити унос више изрази – излазак на `q`

# Додајемо ствари

- Сви изрази су у форми:

`val1 op1 val2 op2 val3 ...`

- Где су `valN` реални бројеви – могу почињати са `.`, `+` или `-`
- А `opN` је операција `+`, `-`, `*`, `/` или `%` - `*`, `/` и `%` су приоритетније
- Са `;` се означава крај изрази
- У изразу могу постојати заграде – и треба прво да се срачунају изрази у њима
- Омогућити унос више изрази – излазак на `q`
- У случају лошег изрази опоравити се и наставити иза првог следећег `;`

# Додајемо ствари

- Сви изрази су у форми:

`val1 op1 val2 op2 val3 ...`

- Где су `valN` реални бројеви – могу почињати са `.`, `+` или `-`
- А `opN` је операција `+`, `-`, `*`, `/` или `%` - `*`, `/` и `%` су приоритетније
- Са `;` се означава крај изрази
- У изразу могу постојати заграде – и треба прво да се срачунају изрази у њима
- Омогућити унос више изрази – излазак на `q`
- У случају лошег изрази опоравити се и наставити иза првог следећег ;
- Променљиве се могу уводити следећим изразом:  
`let varName = someExpression`
- Уведена променљива се може спомињати у каснијим изразима

# Додајемо ствари

- Граматика која укључује и променљиве
- Додајемо S (**S**tatement) и V (**V**ariable):

$S \rightarrow \text{let } V = E$	$E \rightarrow T$	$T \rightarrow P$	$P \rightarrow N$
$S \rightarrow E$	$E \rightarrow E + T$	$T \rightarrow T * P$	$P \rightarrow V$
	$E \rightarrow E - T$	$T \rightarrow T / P$	$P \rightarrow ( E )$

$N \rightarrow \text{Real number}$

$V \rightarrow \text{Valid name}$