Формални Езици и Езикови Процесори
ТУ, кат. КС, летен семестър 2012

Лекция 10

Тема:

Преки методи
За
Синтактичен Анализ

Съдържание:

- Преки (евристични) методи.
- Обратен Полски Запис (ОПЗ), Reverse Polish Notation (RPN).
- Алгоритми за конвертиране на АИ от инфиксна в постфиксна форма.
- Приложение на ОПЗ за генерация на обектен код
- Приложение на ОПЗ за интерпретация (пресмятане) на изрази
- Демо програма

Въведение

- Даден е аритметичен израз a + b * c
- Интуицията подсказва, че след компилация Обектният Код следва да има вида:

  1. *, b, c
  2. +, a, (1)

Какви са възможните подходи (методи) за постигане на тази цел?

– Систематични, формални (КСГ граматики)

– Евристични – свързани с въображението, интуицията и умението на изследователите

Системен, формален подход

- КСГ и трансформации на дървото на синтактичния анализ

Интуитивен подход

Как да постигнем същия резултат (генерация на Обектен Код) без формализми?

Решението:

- Да се развият евристични методи
  - Алгоритъм на Рутисхаузер
  - Концепцията ОПЗ-Reverse Polish Notation

Синтактичен Анализ

Два подхода – Интуитивен(евристичен) и формален(систематичен)

Класификация на методи за СА:

- Преки (интуитивни, евристични)
  - Метод на Рутисхаузер
  - Обратен Полски Запис ОПЗ, Reverse Polish Notation (RPN)
- Формални (основани на ТФЕ)
  - Top-down низходяща стратегия за СА
  - Bottom-up възходяща стратегия за СА

Преки методи

Алгоритъм на Рутисхаузер

Алгоритъм на Рутисхаузер

Вход: АИ, записани в пълна скобкова форма

Алгоритъм: Сканиране на входния низ и анализ на ляв контекст с дължина 4

Пример: *a + b * c* след конвертиране в скобкова форма

*(a + (b * c))* служи да се генерират следните две триади

1. *, b, c
2. +, a, (1)

Алгоритъм на Рутисхаузер

Недостатък:

Оценка:

Преки методи

Обратен Полски Запис

Reverse Polish Notation

Въведение в ОПЗ

Известни са три начина за запис и представяне на аритметични изрази:

инфиксен	**a + b**
постфиксен	**a b +**
префиксен	**+ a b**

Въведение в ОПЗ

- Естественият начин за означаване на изрази (съставени от операнди и операции) в математиката и програмирането е ***Инфиксният Запис***, при който двуместните операции се разполагат между своите операнди.
- Например, $a + b$ $c + d * e$ $a \wedge b \wedge c$
- Наред с тази общоприета нотация, възможни са още следните два начина за представяне (означаване) на изрази:
- префиксна (prefix) нотация, при която операциите се разполагат преди (предшестват) операндите. Нарича се ***Прав Полски Запис*** (ППЗ);
- постфиксна (postfix) нотация, при която операциите се разполагат след (следват) операндите. Нарича се ***Обратен Полски Запис*** (ОПЗ).

Въведение в ОПЗ

- Както бе показано, трите начина за представяне на изрази са както следва:
- Инфиксен запис a + b
- Префиксен запис + a b
- Постфиксен запис a b +

Примери за ОПЗ

a + b + c	да се преведе в	a b + c + или a b c + +
a * b * c	да се преведе в	a b * c * или a b c * *
a ^ b ^ c	да се преведе в	a b ^ c ^ или a b c ^ ^

Правила за формиране ОПЗ

Когато се срещат операции с еднакъв приоритет, от значение за превода в ОПЗ е тяхната асоциативност (лява или дясна).

Събирането е ляво асоциативна операция и правилният превод в ОПЗ на израза $a+b+c$ следва да се представи като $ab+c+$. Друго възможно означаване на същия израз като ОПЗ съгласно правилото, че операциите следват своите операнди, е $abc++$. То е некоректно, т.к. в този случай събирането се третира като дясно асоциативна операция. Това е валидно за всички ляво асоциативни операции, например +, -, *, /, %(модул).

Правила за формиране ОПЗ

Когато се срещат операции с еднакъв приоритет, от значение за превода в ОПЗ е тяхната асоциативност (лява или дясна).

Степенуването е дясно асоциативна операция. По тази причина правилният превод на израза *a^b^c* в ОПЗ е *abc^^*. Другото възможно означение *ab^c^* третира степенуването като ляво асоциативна операция и следва да се отхвърли като неправилно.

Примери за ОПЗ

a + b * c да се преведе в a b c * +

(a + b) * c да се преведе в a b + c *

ОПЗ е безскобъчен запис.

Правила за формиране ОПЗ

Когато се срещат операции с различен приоритет, от значение за превода в ОПЗ е техният приоритет.

Операцията с по-висок приоритет се записва преди операцията с по-нисък приоритет. Високо приоритетната операция изпреварва (предшества) операцията с по-нисък приоритет, т.к. при интерпретацията на израза тя трябва да се изпълни първа, по-рано, преди останалите операции.

Правила за формиране ОПЗ

При превода в ОПЗ редът на операндите никога не се променя. Те се записват в същата последователност, в която се срещат във входния поток.

Правила за формиране ОПЗ

Природата на ОПЗ е такава, че скобите, които се използват за означаване на подизрази и се третират като операция с най-висок приоритет, стават излишни (ненужни). По тази причина те липсват при означаване на изрази в постфиксна нотация, а обратният полски запис е прието още да се нарича и безскобъчен запис.

Упражнения в/у ОПЗ

- Конвертирайте в ОПЗ:

  A

  (A)

  (((a)))

  A+B

  A+B-C

  (a+b) * (c-d)

  (a+b*c^d)

Алгоритми за преобразуване на изрази в ОПЗ

Съществува множество алгоритми за преобразуване на изрази от инфиксен в постфиксен запис. Едни се отнасят към преките (евристични, интуитивни), а други към формалните (систематични) методи за синтактичен анализ.

Четири известни алгоритми за превод в ОПЗ са:

- Възходящо обхождане на двоично дърво;
- Евристичен алгоритъм на Дийкстра;
- Метод на рекурсивното спускане (низходящ синтактичен анализ);
- Метод на операторното предшествие (възходящ синтактичен анализ).

Дв. дърво: Стратегии за обход

- Дадено: структура двоично дърво
- Три стратегии за обхождане:
  - Top-down низходяща стратегия
  - Смесена стратегия
  - Bottom-up възходяща стратегия

Дв. дърво: Стратегии за обход

- Низходящ обход (top-down, preorder)
  - Корен
  - Низходящ обход на ляво поддърво
  - Низходящ обход на дясно поддърво

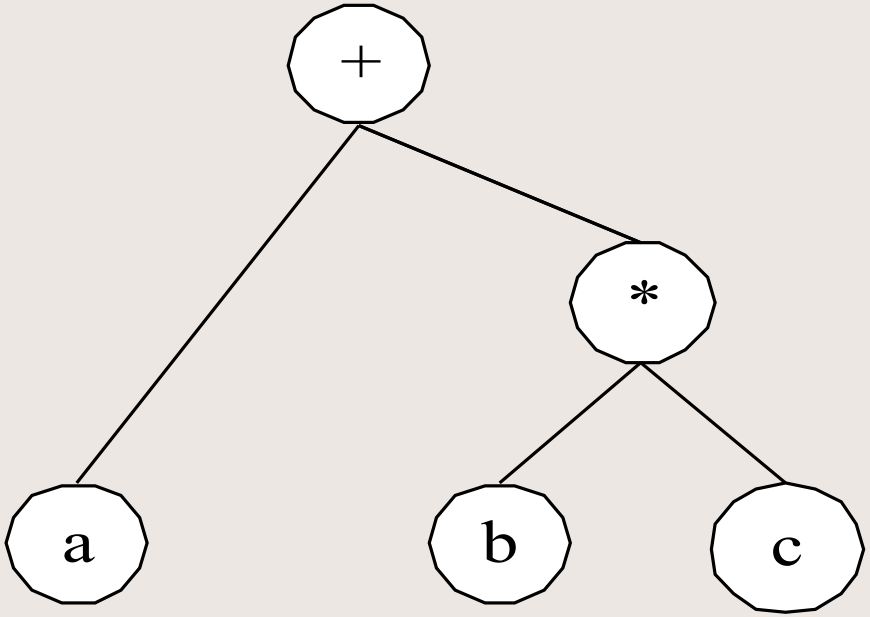
Дв. дърво: Стратегии за обход

- Смесен обход (mixed, inorder)
  - Смесен обход на ляво поддърво
  - Корен
  - Смесен обход на дясно поддърво

Дв. дърво: Стратегии за обход

- Възходящ обход (bottom-up, postorder)
  - Възходящ обход на ляво поддърво
  - Възходящ обход на дясно поддърво
  - Корен

Аритметичен израз a + b * c като двоично дърво



Аритметичен израз a + b * c като двоично дърво

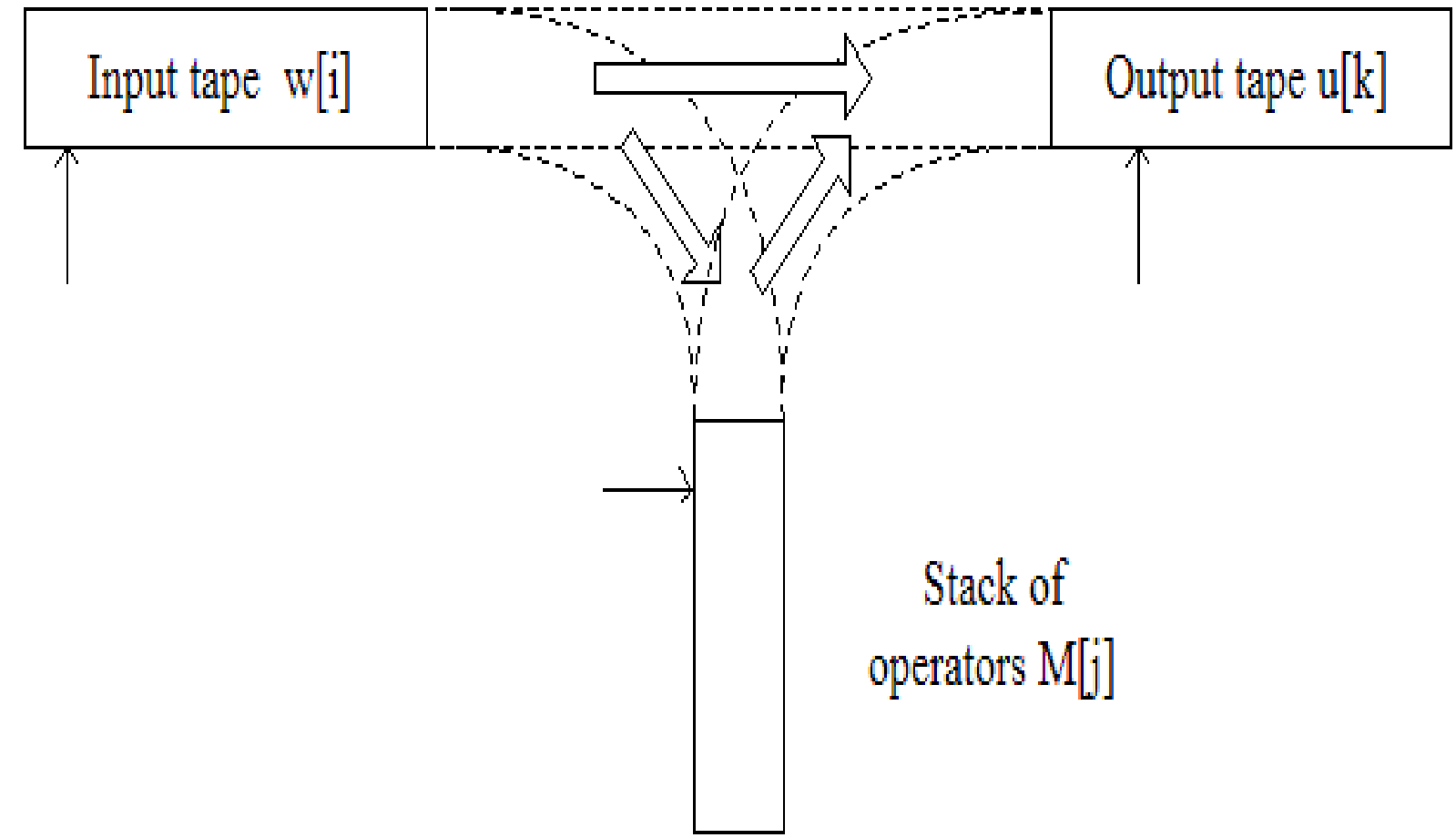
- Резултат от обхождането
  - Низходящ обход + a * b c префикс
  - Смесен обход a + b * c инфикс
  - Възходящ обход a b c * + постфикс

Алгоритъмът за възходящ обход на двоично дърво служи като метод за превод на изрази от инфиксен запис в постфиксен запис ОПЗ.

Евристичен алгоритъм на Дийкстра

.

Обща схема



Обща схема включва:

- Входна лента *w* със сканиращ показалец *i*. Съдържа инфиксния запис на низа (израза), подлежащ на преобразуване в ОПЗ;
- Изходна лента *u* със сканиращ показалец *k*. Служи за запис на резултата от работата на алгоритъма, т.е. ОПЗ на низа от входната лента;
- Стек *M* със стеков показалец *j*. Служи за временно съхраняване на символи операции от входния поток. Нарича се стек на операциите.

Алгоритъм на Дийкстра

Алгоритъмът се основава на следната идея:

Входният низ, съдържащ операнди и операции, се сканира последователно Л>>Д. Операндите се записват направо в изходната лента. Операциите временно се съхраняват в стека, след което също се записват в изходната лента.

Следва последователността от стъпки:

Алгоритъм на Дийкстра

1. Входната лента се сканира Л -> Д.

2. Ако входният поток е изчерпан и празен стек на операции, *Край*.

3. Ако текущият сканиран символ е операнд, той се записва в изходната лента. u[k++] = w[i++]; Преход към 1.

4. Ако текущият сканиран символ е операция, нейният приоритет се сравнява с приоритета на операцията от върха на стека:

А/ Ако входната операция има по-висок приоритет, т.е. prec(w[i])>prec(M[j]), тя се записва на върха на стека.

M[++j] = w[i++]; Преход към 1.

Б/ Обратно, ако входната операция е с по-нисък или равен приоритет, т.е. prec(w[i])<=prec(M[j]), тогава операцията от върха на стека се записва в изходната лента. **Сканиращият показалец на входния низ не се променя**. Сравнението е между входния низ и стека, а действието е между стека и изходната лента.

u[k++] = M[j--]; Преход към 1.

Подробности по алгоритъма

- Символът, който ограничава входния низ (служи за край на входния запис) # или EOF, се приема за операция с най-нисък приоритет.
- Как ще работи алгоритъмът в началото, когато стекът е празен и няма записана операция в него? Приема се, че маркерът за дъно на стека подобно на ограничителя на входния низ, е операция с най-нисък приоритет.

Подробности по алгоритъма

Една примерна схема на приоритети за двуместните операции (адитивни – събиране, изваждане и мултипликативни – умножение, деление, модул (остатък от деление)) на аритметични изрази с отчитане на горните забележки може да се зададе по следния начин:

- Символ край на входния поток # 0
- Маркер дъно на стек ў 0
- Събиране, изваждане +, – 1
- Умножение, деление, модул *, /, % 2

Направените уточнения позволяват прилагането на алгоритъма върху инфиксни изрази без скоби от вида *a + b * c #*.

Подробности по алгоритъма

Ново уточнение позволява алгоритъмът без изменение да третира и входни записи с подизрази, заградени в скоби ( ). Скобите се третират като операции с динамичен приоритет. Лявата скоба във вх. низ е операция с най-висок приоритет. Алгоритъмът определя еднозначно символ (винаги да се записва в стека. След като се запише в стека, лявата скоба променя своя приоритет в приоритет, по-нисък от приоритета на аритметичните операции. Дясната скоба във входния низ се приема за операция с приоритет по-нисък от този на аритметичните операции. Тази схема позволява всеки подизраз да се обработи автономно, независимо от контекста, в който се среща. След като се генерира ОПЗ на подизраза, се получава конфигурация, при която показалецът на входния низ сочи дясна скоба, а на върха на стека има лява скоба. Те взаимно се неутрализират, като входният показалец се инкрементира, а лявата скоба от върха на стека се извежда от стека, без да се записва в изходната лента (ОПЗ е безскобъчен запис).

Подробности по алгоритъма

- Следната таблица представя схема на динамичните приоритети за аритметични изрази, които допускат всички бинарни аритметични операции (събиране, изваждане, умножение, деление, модул и степенуване), както и подизрази, заградени в скоби:

Операция	Динамичен приоритет	
	вх. Лента	Стек
Символ край на входния поток #	0	n/a
Маркер дъно на стек ў	n/a	0
Лява скоба (	max	1
Дясна скоба)	1	n/a
Събиране, изваждане +, –	2	2
Умножение, деление, модул *, /, %	3	3
Степенуване ^	4	4

Подробности по алгоритъма

- Следната таблица представя схема на динамичните приоритети за аритметични изрази, които допускат всички бинарни аритметични операции (събиране, изваждане, умножение, деление, модул и степенуване), както и подизрази, заградени в скоби:

Операция	Динамичен приоритет	
	вх. Лента	Стек
Символ край на входния поток #	0	n/a
Маркер дъно на стек ў	n/a	0
Лява скоба (	max	1
Дясна скоба)	1	n/a
Събиране, изваждане +, –	2	2
Умножение, деление, модул *, /, %	3	3
Степенуване ^	5	4

Подробности по алгоритъма

- Следната таблица представя схема на динамичните приоритети за аритметични изрази, които допускат всички бинарни аритметични операции (събиране, изваждане, умножение, деление, модул и степенуване), както и подизрази, заградени в скоби:

Операция	Динамичен приоритет	
	вх. Лента	Стек
Символ край на входния поток #	0	n/a
Маркер дъно на стек ў	n/a	0
Лява скоба (	>=6	1
Дясна скоба)	1	n/a
Събиране, изваждане +, –	2	2
Умножение, деление, модул *, /, %	3	3
Степенуване ^	5	4

Алгоритъм на Дийкстра - пример

- a + b * c

Алгоритъм на Дийкстра - пример

- (a + b) * c

Алгоритъм на Дийкстра - пример

- Възможно е входните данни да съдържат както синтактично верни, така и неправилно конструирани инфиксни изрази. Описаният алгоритъм не проверява входния низ за правилен синтаксис и има смисъл да се активира само при коректни входни записи. По тази причина се въвежда проверка за правилността на входния низ, която предшества ОПЗ алгоритъма. Подобно на алгоритъма, и проверката се основава на евристични съображения като:
- Баланс на скоби в изразите. Броят на скобите се контролира с брояч на събиране за всяка лява скоба и на изваждане за всяка дясна скоба.
- Допустимо съседство между различни входни символи операнди, операции, леви и десни скоби

Алгоритъм на Дийкстра - пример

- Програма *opz8e.cpp* е реализирана по описания евристичен алгоритъм. Входният поток се въвежда от клавиатурата или пренасочва от предварително създаден текстов файл. Входната последователност съдържа изрази, съставени от операнди и операции. Операндите се задават като едносимволни идентификатори, едносимволни десетични константи и подизрази, заградени в скоби ( ). За означаване на двуместните операции служат символите + за събиране, – за изваждане, * за умножение, / за деление. % за модул (остатък от деление) и ^ за степенуване. Интервалите (празен символ или табулация) се третират като незначещи разделители и се допускат навсякъде във входния поток.

Алгоритъм на Дийкстра - пример

- Демо програма opz8e.cpp
- Изпълним код opz8e.exe

Алгоритъм на Дийкстра - пример

- Програмата е композирана от три функции:

Функция	Предназанчение
void compress(void);	Елиминира незначещи разделители във входа.
int syntax(void);	Проверява входния низ за правилен синтаксис.
void opz(void);	Реализира евристичен алгоритъм и генерира ОПЗ.

Алгоритъм на Дийкстра - пример

```
 void main( )
{
  //   въвеждане на входния низ
  compress( );
  switch  (  syntax( )  )        {
case 0: printf("\nПравилен синтаксис. ");                opz();     break;
case 1: printf("\nНеправилен символ за начало на израз");    break;
case 2: printf("\nНевалиден символ във входния поток");     break;
case 3: printf("\nНевалидна двойка съседни символи");       break;
case 4: printf("\nНебалансирани скоби  (  ) ");                    break;
case 5: printf("\nДясна скоба открита преди лява скоба");  break;
                                  };
}
```

Приложения на ОПЗ

Генерация на обектен код

- Едно типично приложение на ОПЗ е да се използва като междинен език при генериране на обектен код за процесор с регистрова архитектура. Генерацията на обектен код се получава много лесно, ако входът е представен в постфиксна вместо в инфиксна форма. Алгоритъмът се основава на следната идея: Входният низ съдържа запис в постфиксна форма ОПЗ. Той се сканира последователно отляво надясно. Всеки срещнат операнд се пропуска без обработка. При всяка срещната операция се обработва ляв контекст на сканирания входен поток с дължина 2, който заедно с текущия сканиран символ представлява низ от вида

  <операнд1> <операнд2> <операция>

- Такъв низ е много лесен за локализиране. Той е ресурс, от който се генерира една инструкция от обектния код.

Генерация на обектен код

1. Входната лента се сканира Л -> Д.
2. Ако входният поток е изчерпан, тогава *Край*.
3. Ако текущият сканиран символ е операнд, не се предприема действие. Преход към 1.
4. Ако текущият сканиран символ е операция, се генерира елемент на обектния код след обработка (локализиране) на ляв контекст с дължина 2. Преход към 1.

- Следват два примера като илюстрация.

Примери

- Представени са два варианта за генериране на обектен код в термините на асемблерски инструкции за хипотетичен процесор с регистрова архитектура:
- Двуадресни инструкции с операнди символи от входния низ или междинни резултати от изпълнението на предишни инструкции, указани в скоби.
- Триадресни инструкции, чиито операнди са символи от входния низ или регистри с общо предназначение, които се задават мнемонично като *regx*, където *x* варира от *0* до *max*.

Пример 1

Infix Notation RPN	2-address object code	3-address object code
a + b * c		
a b c * +	1. mul, b, c 2. add, a, (1)	1. mul, b, c, reg0 2. add, a, reg0, reg0

Пример 2

Infix Notation RPN	2-address object code	3-address object code
(a +b) / (c – d)		
a b + c d – /	1. add, a, b	1. add, a, b, reg0
	2. sub, c, d	2. sub, c, d, reg1
	3. div, (1), (2)	3. div, reg0, reg1, reg0

Интерпретация на изрази

Друго характерно приложение на ОПЗ е да се използва като междинен език при интерпретиране (пресмятане, оценяване) на АИ в процесор със стекова архитектура. Това се постига лесно, ако входът е представен в постфиксна вместо в инфиксна форма. Алгоритъмът се основава на следната идея: Входният низ съдържа запис във форма ОПЗ. Сканира се Л->Д. Всеки срещнат операнд се записва в стека. Всяка срещната операция се изпълнява с операнди, които се четат от стека. Резултатът от изпълнението на операцията се записва в стека. Предполага се, че входната лента съдържа коректно формиран (свободен от грешки) обратен полски запис.

Интерпретация на изрази

1. Входната лента се сканира Л -> Д.
2. Ако входният поток е изчерпан, тогава *Край*.
3. Ако текущият сканиран символ е операнд, той се записва в стека. Преход към 1.
4. Ако текущият сканиран символ е операция, тя се изпълнява с операнди от върха на стека. Резултатът се записва обратно в стека. Преход към 1.

- Следват два примера като илюстрация.

Пример 1 описва израз с операнди константи. Изразът се пресмята и стойността му остава на върха на стека.

Infix notation: 5 + 6 * 8 RPN: 5 6 8 * +

Input tape:	5 6 8 * + #	Stack: ў
Scanned string:	5 6 8 * + #	Stack: ў 5
	^	
Scanned string:	5 6 8 * + #	Stack: ў 5 6
	^	
Scanned string:	5 6 8 * + #	Stack: ў 5 6 8
	^	
Scanned string:	5 6 8 * + #	Stack: ў 5 48
	^	
Scanned string:	5 6 8 * + #	Stack: ў 53
	^	
Scanned string:	5 6 8 * + #	Stack: ў 53
	^	

Край. Вх. Низ е изчерпан. Стойността на израза е на върха на стека.

Пример 2 описва израз с АО и операция за присвояване. Операнд a се третра като l-value. Затова a’ означава адрес на a.

Infix notation: a = (5+6) / (b–c) RPN: a’ 5 6 + b c – / =

Input tape:	a’ 5 6 + b c – / = #	Stack: ў
Scanned string:	a’ 5 6 + b c – / = #	Stack: ў a’
	^	
Scanned string:	a’ 5 6 + b c – / = #	Stack: ў a’ 5
	^	
Scanned string:	a’ 5 6 + b c – / = #	Stack: ў a’ 5 6
	^	
Scanned string:	a’ 5 6 + b c – / = #	Stack: ў a’ 11
	^	
Scanned string:	a’ 5 6 + b c – / = #	Stack: ў a’ 11 b
	^	
Scanned string:	a’ 5 6 + b c – / = #	Stack: ў a’ 11 b c
	^	
Scanned string:	a’ 5 6 + b c – / = #	Stack: ў a’ 11 b–c
	^	
Scanned string:	a’ 5 6 + b c – / = #	Stack: ў a’ 11/(b–c)
	^	
Scanned string:	a’ 5 6 + b c – / = #	Stack: ў
	^	
Scanned string:	a’ 5 6 + b c – / = #	Stack: ў
	^	

Край. Вх.низ изчерпан. Стек празен. Стойността на израза е запазена в a’, т.е. На адреса на променливата a.

ОПЗ и АИ за стек базиран процесор

ОПЗ на изрази, които се интерпретират, може да се представи и като последователност от асемблерски инструкции за виртуален стеков процесор. Ще въведем следните асемблерски команди.

Първите три имат код на операция и операнд, а останалите имат само код на операция.

Конвенцията по подразбиране е, че операндите са разположени на върха на стека.

ОПЗ и АИ за стек базиран процесор

PushAdr <operand>	Запис в стека на адреса на променливата операнд.
PushVal <operand>	Запис в стека на стойността на променливата операнд.
PushImd <operand>	Запис в стека на стойността непосредствен операнд.
Add	Събиране. Сумата замества операндите в стека.
Sub	Изваждане. Разликата замества операндите в стека.
Mul	Умножение. Произведението замества операндите в стека.
Div	Деление. Частното замества операндите в стека.
Assgn	Присвояване. Стойността от върха на стека се изпраща на адрес, разположен под върха на стека.

Пример 1

RPN: 5 6 8 * +

ОПЗ като последователност от АЕ мнемонични инструкции за процесор със стек ориентирана архитектура:

PushImd 5
PushImd 6
PushImd 8
Mul
Add

Пример 2

RPN: a’ 5 6 + b c – / =

ОПЗ като последователност от АЕ мнемонични инструкции за процесор със стек ориентирана архитектура:

PushAdr a
PushImd 5
PushImd 6
Add
PushVal b
PushVal c
Sub
Div
Assgn

Благодаря
За
Вниманието