თემა 1 შესავალი დაპროგრამების ენებში

* გამოთვლითი ტექნიკის განვითარების ქრონოლოგია
* დაპროგრამების ენის აბსტრაქციის დონე
* დაპროგრამების ენების ევოლუცია
* გამოთვლადობა

სასწავლო კურსის საგანს წარმოადგენს დაპროგრამების ენა როგორც შესწავლის და კვლევის ობიექტი. კურსის მიზანია

* ისტორიულ ჭრილში განიხილოს დაპროგრამების ძირითადი კონცეფციები;
* გააცნოს მსმენელს ენის კონსტრუქციების რეალიზება და მათი მოხმარების შესაძლებლობები;
* ენების შედარებითი ანალიზის საფუძველზე დაეხმაროს მსმენელს შეგნებულად შეარჩიოს პროგრამირების შესაბამისი ტექნოლოგიები პრაქტიკულ საქმიანობაში.

სამწუხაროდ, ხშირ შემთხვევაში პროგრამისტებს არა აქვთ საკმარისი უნარ–ჩვევები, რომ გააანალიზონ და შეადარონ სხვადასხვა ენის კონსტრუქციები და ცნებები, და აქედან გამომდინარე სწორად შეაფასონ თანამედროვე ენების უპირატესობები და უარყოფითი მხარეები. კვალიფიციური დეველოპერი არ იტყვის: „ენა ლ1 უფრო მძლავრია (ან ეფექტურია), ვიდრე ენა ლ2“. ის იტყვის „ამ ამოცანათა კლასისთვის ლ1 ენის გამოყენება უფრო რეკომენდირებულია, ვიდრე ლ2 ენისა“.

*პროგრამა* წარმოადგენს სიმბოლოების თანმიმდევრობას, რომელიც განსაზღვრავს რაიმე მოქმედებებს (გამოთვლას).

*დაპროგრამების ენა* არის იმ წესების ერთობლიობა, რომელიც განსაზღვრავს ქმნის თუ არა მოცემული სიმბოლოების თანმიმდევრობა პროგრამას და რა მოქმედებებს (გამოთვლებს) აღწერს პროგრამა.

*კომპიუტერული პროგრამა* არის გარკვეულ დაპროგრამების ენაზე (მისი წესების გათვალისწინებით) ჩაწერილი ინსტრუქციების თანმიმდევრობა, რომლის შესრულება საჭიროა კონკრეტული ამოცანის გადასაწყვეტად კომპიუტერზე.

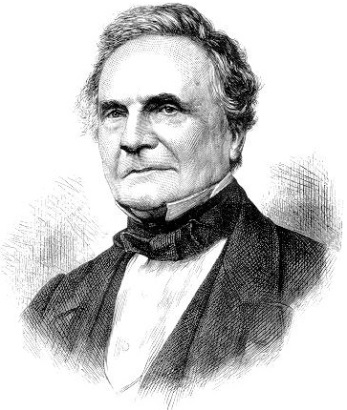
გავიხსენოთ კომპიუტერის განვითარების რამდენიმე ეტაპი:

1623 წელს, გერმანიაში, ქალაქ შტუტგარტში, **ვილჰელმ შიკარდის** მიერ შეიქმნა პირველი გამომთვლელი მანქანა სახელწოდებით ”გამომთვლელი საათი”. შიკარდის მანქანა 6–თანრიგა იყო. მას შეეძლო რიცხვების შეკრება და გამოკლება. თანრიგების გადავსებას იგი ზარის დარეკვით იტყობინებოდა. ერთი პერიოდი შიკარდის მანქანა თავის ნახაზებიანად დაიკარგა. იგი აღმოაჩინეს 1935 წელს, ხოლო მეორე მსოფლიო ომის დროს ისევ დაიკარგა. 1956 წელს იპოვეს მხოლოდ ნახაზები. 1960 წელს აღნიშნული ნახაზების მიხედვით ენთუზიასტთა ჯგუფმა ააგო მანქანა და დაამტკიცა, რომ შიკარდის მანქანა ერთ-ერთი პირველი მუშა გამომთვლელი მანქანაა. მანქანა დღესაც ინახება ქალაქ შტუტგარტის სამეფო ბიბლიოთეკაში.

1642 წელს პარიზში ფრანგმა მეცნიერმა **ბლეზ პასკალმა** წარმოადგინა გამომთვლელი მანქანა სახელწოდებით ”პასკალინა”. პასკალის მანქანა წარმოადგენდა ყუთს, რომელშიც ჩამონტაჟებული იყო ერთმანეთთან გადაბმული მრავალრიცხოვანი კბილოვანი ბორბლები. რიცხვების შეტანა ხდებოდა ბორბლების შესაბამისი მობრუნებით, მუშობის პრინციპი კი ეფუძნებოდა ამ ბრუნვების თვლას.

პირველი მანქანა ააგეს 1645 წელს, ხოლო 1652 წლამდე იყო შექმნილი ”პასკალინა”–ს დაახლოებით 50 ვარიანტი. თითქმის 300 წლის მანძილზე, პასკალის გადაბმული ბორბლების პრინციპი გამოიყენებოდა არითმომეტრების შექმნაში.

1674 წელს გერმანელი მეცნიერის **გოტფრიდ ვილჰელმ ლაიბნიცის** მიერ შემუშავებულ იქნა ”ბიჯური გამომთვლელი”, რომელიც ააგო ფრანგმა **ოლივერმა.** ლაიბნიცის მანქანას შეეძლო 5 და 12 ნიშნა რიცხვების გამრავლება 16 ნიშნა შედეგით.

1808 წელს ფრანგმა ფეიქარმა და გამომგონებელმა **ჟოზეფ** **მარი** **ჟაკარმა** შექმნა საქსოვი მანქანის ძაფის ავტომატური კონტროლის ხერხი, რისთვისაც გამოიყენებოდა სპეციალური ნახვრეტებიანი ბარათები. ჟაკარდის მანქანა ფართოდ გავრცელდა საფეიქრო წარმოებაში მსხვილსახიანი ქსოვილების გამოსამუშავებლად.

1812 წელს ინგლისელმა მეცნიერმა **ჩარლზ ბებიჯმა** შეიმუშავა მექანიკური კალკულატორის პროექტი, რომლის მიხედვითაც 1823 წელს ააგო მთლიანად ავტომატიზებული მანქანა Difference Engine. ბებიჯის მანქანის საშუალებით შეიძლებოდა ექვსნიშნა რიცხვების შეკრება, გამოკლება, გამრავლება და გაყოფა.

1837 წელს ჩარლზ ბებიჯმა პირველმა გამოთქვა იდეა პროგრამულად მართვადი გამომთვლელი მანქანის შესახებ. მან სცადა ანალიტიკური მანქანის Analytical Engine შექმნა. აღნიშნულ მანქანას გააჩნდა ცენტრალური პროცესორული მოწყობილობა, მეხსიერება და პერფობარათები, რომლებზეც დაიტანებოდა გარკვეული პროგრამები. ბებიჯმა ვერ შეძლო თავისი კომპიუტერის რეალიზება, რადგან ეს რთული აღმოჩნდა იმდროინდელი ტექნიკისათვის.

1842 წელს მათემატიკოსმა, ცნობილი ინგლისელი პოეტის ლორდ ბაირონის ქალიშვილმა - **ადა ბაირონ კინგმა** (ადა ლავლეისმა) გააკეთა ბებიჯის მანქანის აღწერა და შეიმუშავა მისთვის პროგრამა. მან შემოიღო ტერმინები „ციკლი“ და „მუშა უჯრა“. გამოთვლითი ტექნიკის ისტორიაში ადა ლავლეისი აღიარებულია პირველ პროგრამისტად.

1847 წელს ირლანდიელმა მათემატიკოსმა **ჯორჯ ბულმა** გამოუშვა წიგნი ”The Mathematical Analysis of Logic” (”ლოგიკის მათემატიკური ანალიზი”), სადაც ჩამოაყალიბა ახალი ალგებრული სისტემა, რომელსაც შემდგომში ”ბულის ალგებრა ”ეწოდა. ბულის ალგებრის პრინციპები ფართოდ გამოიყენება თანამედროვე კომპიუტერებში.

მხოლოდ 1935–38 წლებში ბებიჯის თეორიული გამოკვლევების საფუძველზე გერმანელმა ინჟინერმა **კონრად ცუზემ** შექმნა ავტომატურად პროგრამირებადი გამომთვლელი მანქანა Z1.

1936 წელს ინგლისელმა მეცნიერმა ალან ტიურინგმა (1912-1954) შექმნა კომპიუტერის მათემატიკური მოდელი, რომელსაც ”ტიურინგის მანქანა” ეწოდა.

1943 წელს ამერიკელმა ფიზიკოსმა **ჯონ მოჩლიმ** ელექტრონიკის ინჟინერთან **პრესპერ ეკერტთან** თანამშრომლობით წარმოადგინა გამომთვლელი მანქანის პროექტი, რომელიც განხორციელდა 1944 წელს ENIAC (**E**lectronic **N**umerical **I**ntegrator and **C**omputer) სახელწოდებით. აგებული იყო ელექტრონულ მილაკებზე, მაგნიტურ ელემენტებზე და ეკავა 300 კვ. მ. ფართობი. თუმცა პროგრამის მიწოდება ამ კომპიუტერისტვის აღმოჩნდა რთული პროცესი, და ავტორებმა მაშინვე დაიწყეს ახალი კომპიუტერის დაპროექტება, რომელსაც შეეძლო პროგრამის შენახვა მეხსიერებაში.

1945 წელს მათ შეუერთდა ცნობილი ამერიკელი მათემატიკოსი **ჯონ ფონ ნეიმანი,** რომელმაც თავის ნაშრომში წარმოადგინა შენახული პროგრამის კონცეფცია, აღწერა ახალი კომპიუტერის EDVAC-ის მოწყობილობები, ნათლად და მარტივად ჩამოაყალიბა უნივერსალური გამომთვლელი მანქანების (ე.ი. კომპიუტერების) ძირითადი პრინციპები. ფონ ნეიმანის პრინციპების საფუძველზე შეიქმნა ელექტრონული გამომთვლელი მანქანის არქიტექტურა, რომელიც თითქმის უცვლელია დღემდე.

მიუხედავად იმისა, რომ მეხსიერებაში შენახული პროგრამის კონცეფცია წამოაყენა ჯონ პრესპერ ეკერტმა 1944 წლის იანვარში, და იმავე წელს მან და ჯონ მოჩლიმ დაიწყეს მუშაობა EDVAC (**E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer) კომპიტერის შექმნაზე, პირველი კომპიუტერი შენახული პროგრამით ამუშავდა ინგლისში, კემბრიჯის უნივერსიტეტში.

კემბრიჯის უნივერსიტეტის პროფესორი **მორის უილკსი**, რომელმაც მურის სკოლაში (აშშ) მოისმინა ლექციათა კურსი EDVAC მანქანის შესახებ, დაბრუნდა ინგლისში და დაიწყო EDSAC (**E**lectronic **D**elay **S**torage **A**utomatic **C**alculator) კომპიუტერის განვითარება. ეს მოხდა 1946 წლის ნოემბერში, ხოლო დასრულდა 1949 წელს, ორი წლით ადრე, ვიდრე აშშ–ში ამუშავდა EDVAC –ი.

პირველი კომპიუტერული პროგრამები იქმნებოდა ორობით კოდში (ე.წ. მანქანურ ენაზე, რომელიც ყოველი ტიპის კომპიუტერისთვის განსხვავებულია და დამოკიდებულია მის აპარატულ ნაწილზე). ტექნიკურად ორობითი კოდი პრობლემას არ ქმნის. პრაქტიკულ გამოყენებაში კი პრობლემატურია, რადგან პროგრამის ყოველი ინსტრუქცია (შესრულებადი ბრძანება) ჩაწერილია ბიტების სახით, ეს კი ართულებს მისი შინაარსის აღქმას.

დაპროგრამების ენების განვითარების შემდეგ ეტაპზე შეიქმნა ასემბლერის ენები. მათში პროგრამისტს შეეძლო მანქანური ბრძანებების კოდების ნაცვლად ამ ბრძანებების დასახელების გამოყენება (ინგლისური ენის აბრევიატურების გამოყენება, მაგალითად ADD, SUB და სხვა), უფრო მეტიც, მონაცემების დამისამართება ხდებოდა არა მეხსიერების უჯრედის ორობითი მისამართის მითითებით, არამედ ამ მონაცემების სახელების მითითებით (შემოვიდა ცვლადის სახელის - იდენტიფიკატორის - ცნება). ასემბლერში შესაძლებელი იყო მუდმივების განსაზღვრაც.

მაგალითად, ასემბლერის ბრძანება load R3, 65 რომელიც აღნიშნავს „R3 რეგისტრში ჩატვირთე მეხსიერების 65 –ე უჯრედის მნიშვნელობა“, გაცილებით მარტივია გასაგებად, ვიდრე შესაბამისი ბიტების მიმდევრობა.

ცხადია, რომ ასემბლერზე დაწერილ პროგრამას კომპიუტერი უშუალოდ ვერ შეასრულებდა, პროგრამა ჯერ უნდა თარგმნილიყო მანქანურ ენაზე. თარგმნას ასრულებს სპეციალური პროგრამა, რომელსაც ტრანსლატორი ეწოდება. ასემბლერის ენის ტრანსლატორს ასევე ასემბლერი ეწოდება. მანქანურ ენას და ასემბლერს დაბალი დონის დაპროგრამების ენებად მოიხსენიებენ.

ახლა შეგვიძლია დავაზუსტოთ დაპროგრამების ენის განსაზღვრა:

*დაპროგრამების ენა* – აბსტრაქციის მექანიზმია. იგი საშუალებას აძლევს პროგრამისტს აღწეროს გამოთვლები (მოქმედებები) აბსტრაქტულად და ამავე დროს საშუალებას აძლევს პროგრამას (ჩვეულებრივ, სახელით ასემბლერი, კომპილერი, ან ინტერპრეტერი), გადაიყვანოს ეს აღწერა კომპიუტერზე შესრულებისათვის საჭირო დეტალურ ფორმაში.

განსხვავებული კლასის ამოცანებისთვის, სავარაუდოდ, საჭიროა აბსტრაქციის სხვადასხვა დონე, და პროგრამისტებს შეიძლება ჰქონდეთ განსხვავებული წარმოდგენა ამ აბსტრაქციის დონის შესახებ. ეს კი განაპირობებს ასობით დაპროგრამების ენების არსებობას.

აბსტრაქციის კონცეფციიდან გამომდინარეობს ზოგადი წესი: რაც მეტია აბსტრაქციის დონე, მით მეტი დეტალი იკარგება.

მაგალითად, C –ს პროგრამას, ასემბლერის პროგრამისგან განსხვავებით, არა აქვს რეგისტრების განაწილების შესაძლებლობა; თუ წერთ Prolog –ზე, კარგავთ C –ში არსებულ შესაძლებლობას განსაზღვროთ ბმული სტრუქტურები მიმთითებლის გამოყენებით.

სწრაფვა მოკლედ, მკაფიოდ და საიმედოდ გამოვხატოთ გამოთვლები აბსტრაქციის მაღალი დონით ბუნებრივად ეწინააღმდეგება სურვილს აღვწეროთ გამოთვლები მოქნილად და დეტალურად. აბსტრაქცია არასდროს იქნება ისეთივე ზუსტი ან ოპტიმალური, როგორც დაბალი დონის აღწერა.

Fortran

პირველი დაპროგრამების ენა, რომელმაც მნიშვნელოვნად გადააჭარბა ასემბლერის ენის დონეს, იყო Fortran (**Form**ula **Tran**slating System). თავდაპირველად შემუშავებულ იქნა 1950-იან წლებში IBM- ის თანამშრომელთა გუნდის მიერ, რომელსაც ხელმძღვანელობდა **ჯონ ბექუსი**. ენა განკუთვნილი იყო სამეცნიერო და რიცხვითი გათვლების აბსტრაქტული აღწერისთვის და აძლევდა მეცნიერებს პროგრამების დაწერის საშუალებას მათემატიკური ნოტაციის/ენის გამოყენებით. Fortran –ს თავიდან შეხვდა დიდი წინააღმდეგობა (თუმცა, როგორც მაღალი დონის აბსტრაქციის გამოყენების ყველა შემდგომ შემოთავაზებებს), ვინაიდან პროგრამისტების უმრავლესობა თვლიდა, რომ კომპილერის მიერ დაგენერირებული კოდი ვერ იქნებოდა ასემბლერის ენაზე „ხელით“დაწერილ კოდზე უკეთესი.

თარგმნილი მანქანური კოდი აღმოჩნდა ეფექტური, და დღესაც ასეთი რჩება – Fortran- ი ახლაც ეტალონად მიიჩნევა პროგრამების შესრულების სისწრაფის თვალსაზრისით. ენამ განიცადა 10–მდე განახლება. ერთ–ერთი მოდიფიკაციის შედეგად ენაში შემოვიდა სიმბოლოებზე და სტრიქონებზე მანიპულირების საშუალებები.

Cobol და PL/1

ენა Cobol–ი (**CO**mmon **B**usiness-**O**riented **L**anguage) შეიქმნა 1950-იან წლებში კომერციული მონაცემების დამუშავების მიზნით. იგი შეიქმნა კომიტეტის მიერ, რომელშიც შედიოდნენ აშშ-ის თავდაცვის სამინისტროს, კომპიუტერების მწარმოებლების და კომერციული ორგანიზაციების (როგორიცაა სადაზღვევო კომპანიები) წარმომადგენლები.

ენის ავტორები თვლიდნენ, რომ Cobol -ი იქნებოდა მხოლოდ დროებითი გადაწყვეტა, რომელიც საჭირო იყო, სანამ შეიქმნებოდა უკეთესი პროექტი. მაგრამ ენა სწრაფად გახდა ყველაზე გავრცელებული თავის სფეროში (როგორც Fortran–ი მეცნიერებაში), რადგან იგი ითვალისწინებდა მისი სფეროს ტიპიური გამოთვლების გამოხატვის ბუნებრივ საშუალებებს.

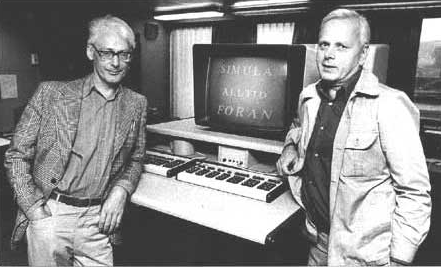
ბიზნეს–მონაცემების დამუშავება გულისხმობს შედარებით მარტივი გათვლების წარმოებას დიდი რაოდენობის რთულ მონაცემთა ჩანაწერებისათვის, ხოლო მონაცემთა სტრუქტურირების თვალსაზრისით, Cobol–ი ბევრად უკეთესია ისეთ ენებზე, როგორიცაა Fortran ან C.

მოგვიანებით IBM –მა შექმნა უნივერსალური ენა PL/1, რომელსაც ჰქონდა Fortran, Cobol და Algol ენების ყველა თვისება. PL/1- მა შეცვალა Fortran–ი და Cobol–ი IBM-ის მრავალ კომპიუტერზე, მაგრამ ეს საკმაოდ ფართო სპექტრის ენა არ იყო მხარდაჭერილი IBM-ის გარეთ. მაგალითად, მინი– და მიკროკომპიუტერებს, რომლებსაც სულ უფრო და უფრო ხშირად იყენებდნენ მონაცემთა დამუშავების ორგანიზაციები, PL/1- ის მხარდაჭერა არ გააჩნდათ.

Algol

Algol–ის (**Algo**rithmic **L**anguage) პირველი ვერსიის აღწერა გამოიცა 1958 წელს. შემოღებული ამერიკელ (Association for Computing Machinery – ACM) და ევროპელ (Society of Applied Mathematics and Mechanics – GAMM) ექსპერტთა ერთობლივი კომიტეტის მიერ, ენა განკუთვნილი იყო ზოგადი და სამეცნიერო გამოყენებებისთვის. მან ვერ მიაღწია Fortran-ის მსგავს ფართო პოპულარობას, რადგან არ ჰქონდა კომპიუტერების მწარმოებლების ისეთი მხარდაჭერა, როგორიც ჰქონდა Fortran–ს. განახლებული ვერსია, Algol 60, ფართოდ გამოიყენებოდა კომპიუტერულ სამეცნიერო კვლევებში და განხორციელდა მრავალ კომპიუტერზე, განსაკუთრებით ევროპაში. ენის მესამე ვერსია, Algol 68, გავრცელდა ენების თეორეტიკოსებს შორის, მაგრამ ფართო გამოიყენება ვერ ჰპოვა.

Algol–ის შთამომავლებს შორის უნდა აღინიშნოს ორი მნიშვნელოვანი ენა: Simula და Pascal –ი.

Simula შეიქმნა ოსლოში, ნორვეგიის კომპიუტერული ტექნიკის ცენტრში 1960-იან წლებში, **ოლე-იოჰან დალის** და **კრისტენ ნიგაარდის** მიერ. როგორც მისი სახელიდან ჩანს, ეს იყო მოდელირების ერთ–ერთი პირველი ენა. Simula ორი დაპროგრამების ენის სახელი – Simula I –ის და Simula 67–ის.

Simula ითვლება პირველ ობიექტზე ორიენტირებულ პროგრამირების ენად. აქვს ფართო სპექტრის გამოყენებები როგორიცაა სიმულაცია, პროცესის მოდელირება, პროტოკოლები, ალგორითმები და სხვა პროგრამები, როგორიცაა კომპიუტერული გრაფიკა. ამ ენის გავლენა შეინიშნება C ++, Java და C #- ენებზე. C ++ –ის ავტორი ბიარნ სტრაუსტრუპი აღიარებს, რომ Simula 67-მა დიდი გავლენა მოახდინა C ++- ის განვითარებაზე.

Pascal

Pascal–ი შექმნილია 1968/9 წლებში და გამოქვეყნდა 1970 წელს შვეიცარიელი მეცნიერის, პროგრამირების თეორეტიკოსის **ნიკლაუს ვირტის** მიერ როგორც პატარა და ეფექტური ენა, რომლის მიზანი იყო მონაცემთა ტიპებზე განაცხადების გაკეთებისა და ტიპების შესაბამისობის კონტროლის იდეების დემონსტრირება. ეს ყველაზე მნიშვნელოვანი კონცეფციებია ოდესმე შემოთავაზებული ენების დაპროექტებაში. პასკალი დიდწილად ცნობილია როგორც პროგრამირების სწავლების ენა, რომელიც უვითარებს დამწყებებს დაპროგრამების კარგი პრაქტიკის უნარ–ჩვევებს.

პრაქტიკული დაპროგრამების თვალსაზრისით, პასკალს აქვს ერთი დიდი უპირატესობა და ერთი დიდი ნაკლი. პასკალის კომპილერი თავდაპირველად დაიწერა პასკალზევე და, ამდენად, ადვილად გადატანადი იყო ნებისმიერ კომპიუტერზე. ენა სწრაფად გავრცელდა, განსაკუთრებით იმ დროს შექმნილ მინი– და მიკროკომპიუტერებზე. სამწუხაროდ, როგორც პრაქტიკული დაპროგრამების ენა, პასკალი ძალიან მცირეა და მასზე ვერ შექმნი დიდ პროგრამულ უზრუნველყოფას. შემდგომში ვირტმა განავითარა ობიექტზე ორიენტირებული ვერსია Object Pascal–ი.

C/C++

დაპროგრამების ენა C შეიქმნა 70–იანი წლების დასაწყისში ამერიკული სატელეფონო კორპორაციის AT&T Bell Laboratory -ის თანამშრომლის **დენის რიჩის** მიერ. C იღებს თავის საწყისებს ორი ენიდან: B და BCPL. BCPL-ის ავტორმა **მარტინ რიჩარდსმა** 1967 წელს შექმნა ენა სისტემური პროგრამების და კომპილერების დაპროგრამებისთვის. 1970 წელს **კენ ტომპსონმა** გამოიყენა დაპროგრამების ენა B ოპერაციული სისტემა UNIX ერთ-ერთი პირველი ვერსიის შესაქმნელად. C ითვლება სისტემური პროგრამირების ენად, აღიარება მოიპოვა, როგორც UNIX ოპერაციული სისტემის ენამ. C -ში განხორციელებულია მაღალი დონის ალგორითმული ენების ყველა საუკეთესო თვისება და კომპიუტერის აპარატურასთან უშუალო წვდომის შესაძლებლობა, მაგალითად, თითოეულ ბიტზე წვდომა (რაც, ჩვეულებრივ, შესაძლებელია ასემბლერზე დაპროგრამებისას). ენის პირველი აღწერა – დენის რიჩის და ბრაიან კერნიგანის წიგნი “The ‘C’ Programming Language” – გამოვიდა 1978 წელს, და დიდი ხნის მანძილზე ეს იყო ენის ერთადერთი სტანდარტი. C -ს უპირატესობათაგან უნდა აღინიშნოს, რომ მისი შესრულება შესაძლებელია სხვადასხვა არქიტექტურის მანქანებზე, სხვადასხვა ოპერაციულ სისტემებში. რადგანაც C -ს აქვს მაღალი დონის ალგორითმული ენის ყველა დამახასიათებელი კომპონენტი, იგი მალევე გახდა პოპულარული აგრეთვე როგორც გამოყენებითი პროგრამირების ენა. C -ს გამოყენებამ სხვადასხვა ტიპის კომპიუტერებზე (სხვადასხვა პლატფორმაზე) გამოიწვია ენის ვერსიების გაჩენა, რაც ხშირად უშლიდა პროგრამების თავსებადობას. ამიტომაც, 1983 წელს ამერიკის სტანდარტების ნაციონალურმა კომიტეტმა ინფორმაციის დამუშავებისა და კომპიუტერების დარგში დააწესა ტექნიკური კომისია, რომელსაც დაევალა ცალსახა და მანქანურ-დამოუკიდებელი C -ენის განსაზღვრა, ანუ სტანდარტის შემუშავება. სტანდარტი საბოლოოდ დამტკიცდა 1989 წელს, ხოლო 1988 წელს გამოვიდა კერნიგანის და რიჩის წიგნის მეორე გამოცემა, რომელშიც აღწერილია C -ს ბოლო ვერსია. ეს ვერსია ცნობილია როგორც ANSI C და ფართოდ გამოიყენება მსოფლიოში.

C –ს შემუშავების მიზანი იყო ასემბლერთან მიახლოებული ენის შექმნა, და ეს უზრუნველყოფს მის მაღალ მოქნილობას; მაგრამ პრობლემა ის არის, რომ ამ მოქნილობის გამო C –ს პროგრამებში შეიძლება ადვილად გაიპაროს დაფარული შეცდომები (გაჩნდეს გვერდითი ეფექტები). მიზეზი: C –ს კომპილერი არ ამოწმებს არასანდო კონსტრუქციებს, არ აკეთებს ტიპების შესაბამისობის კონტროლს და ა.შ.

80-ანი წლების დასაწყისში იგივე Bell Laboratory -ში **ბიარნ სტრაუსტრუპის** მიერ იყო შემოთავაზებული C -ენის გაფართოება, რომელსაც თავიდან ეწოდებოდა “C კლასებით” და რომელიც სინამდვილეში იყო ახალი ობიექტზე ორიენტირებული ენა. მას მოგვიანებით დაერქვა C++. C ++ –ში გასწორებულია C ენის ბევრი შეცდომა და მასზე დაპროგრამება სასურველია მაშინაც, როდესაც ობიექტზე ორიენტირებული თვისებები არ არის საჭირო. C++-ში შესაძლებელია პროგრამირება C-ს სტილშიც, ობიექტზე ორიენტირებულ სტილშიც და ორივე სტილში ერთდროულად. აქვს აგრეტვე განზოგადებული დაპროგრამების მხარდაჭერა. მე-20 საუკუნის 90-იანი წლების დასაწყისიდან C++ გახდა მსოფლიოში ერთ-ერთი ყველაზე აღიარებული ენა. C++ არის განვითარებადი ენა, მისი ბოლო სტანდარტებია C++11, C++14, C++17 და მზადდება სტანდარტი C++20.

Ada

1977 წელს, ამერიკის შეერთებული შტატების თავდაცვის სამინისტრომ გამოაცხადა კონკურსი ახალი დაპროგრამების ენის შექმნაზე, რომელზეც დაიწერებოდა პროგრამული უზრუნველყოფა სამხედრო სისტემებისათვის (მაგ. სამხედრო ობიექტების ბორტზე მყოფი კონტროლის სისტემები) და რომელიც დაფუძნებული იქნებოდა უკვე არსებულ კარგ ენაზე. კონკურსმა გამოავლინა ენა, რომელსაც დაერქვა Ada. ენა დაფუძნებული იყო პასკალზე და სიმულაზე. მისი დამფუძნებელია ფრანგი მეცნიერი, დაპროგრამების ენების დეველოპერი **ჟან იშბია**. Ada –ს სტანდარტი მიიღეს 1983 წელს. ენა უნიკალურია რამდენიმე ასპექტში:

* ენების უმრავლესობის (Fortran, С, Pascal) სტანდარტიზაცია მოხდა მათი ფართო გავრცელების შემდეგ, ამდენად, ენაში დაშვებული შემთხვევითი შეცდომები შედიოდა სტანდარტში. Ada–მ კი გაიარა ინტენსიური ტესტირება და კრიტიკული ანალიზი სტანდარტიზაციის დაწყებამდე.
* ბევრი ენა თავიდან რეალიზებული იყო ერთ კომპიუტერზე და მათზე დიდი გავლენა იქონია ამ კომპიუტერის თავისებურებებმა. ენა Ada თავიდანვე განვითარდა გადატანადი პროგრამების შესაქმნელად.
* Ada–მ გააფართოვა პროგრამირების ენების შესაძლებლობები – მასში განხორციელებულია გამონაკლისი სიტუაციების დამუშავების და პარალელური დაპროგრამების შესაძლებლობები, რომლებიც ტრადიციულად ოპერაციული სისტემის ფუნქციას წარმოადგემდა.

1995 წელს დამტკიცდა ენის ახალი სტანდარტი Ada 95, რომელშიც გასწორდა წინა ვერსიის გარკვეული შეცდომები და, რაც მთავარია, განვითარდა ობიექტზე ორიენტირებული კონცეფციები. ენამ ვერ მოიპოვა პოპულარობა სამხედრო და სხვა ფართომასშტაბიანი პროექტების გარეთ (როგორიცაა კომერციული ავიაცია და სარკინიგზო გადაზიდვები). იგი ითვლება რთულ ენად იმის გამო, რომ მხარს უჭერს პროგრამირების მრავალ ასპექტს, რომლებსაც სხვა ენები უტოვებენ ოპერაციულ სისტემას.

Lisp

1958 წელს **ჯონ მაკკარტიმ** შექმნა ენა Lisp (**LIS**t **P**rocessing), რომელშიც პროგრამა წარმოდგენილია სიმბოლოების წრფივი სიების სისტემის სახით. Lisp-ის ძირითად მონაცემთა სტრუქტურას ასევე ბმული სია წარმოადგენს. თავდაპირველად შეიქმნა კვლევებისთვის გამოთვლით თეორიაში. "სიის" ცნება ძალიან მრავალმხრივი აღმოჩნდა. სიების სახით შეგვიძლია გამოვსახოთ ალგებრული გამოსახულებები, გრაფები, სიმრავლეები, დასკვნის გამოყვანის წესები და სხვა რთული ობიექტები.

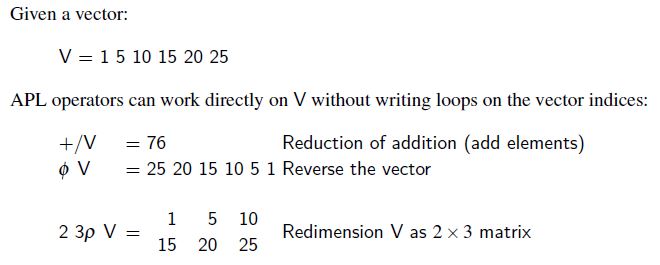
ენა წარმატებით გამოიყენებოდა ბევრ სფეროში, კერძოდ, ხელოვნურ ინტელექტში კვლევების დიდი წილი ჩატარდა Lisp –ზე. ენა იმდენად მნიშვნელოვანი აღმოჩნდა, რომ Lisp –ის პროგრამების ოპტიმალურად შესრულებისთვის სპეციალურად დაპროექტებული და აგებული იყო კომპიუტერები.

რადგან ენა განხორციელდა განსხვავებულ კომპიუტერებზე, წარმოიშვა პრობლემა – Lisp-ის სხვადასხვა დიალექტის გავრცელება. მოგვიანებით შეიქმნა ენა Common Lisp, რომელიც პროგრამების ერთი კომპიუტერიდან მეორეში გადატანის საშუალებას იძლეოდა. Lisp-ის პოპულარულ დიალექტს წარმოადგენს CLOS, რომელშიც მხარდაჭერილია ობიექტზე ორიენტირებული დაპროგრამების სტილი.

APL

1962 წელს ჰარვარდის უნივერსიტეტის პროფესორმა და IBM- ის თანამშრომელმა **კენეტ ევერსონმა** წიგნში „A Programming Language“ აღწერა ახალი ენა. მოგვიანებით, ენა იყო სტანდარტიზებული. სტანდარტი ISO 8485: 1989 აღწერს Core APL –ს; ISO / IEC 13751: 2001 - გაფართოებულ APL –ს. გარდა ამისა, გაკეთდა მცდელობები მისი შევსებისა თანამედროვე შესაძლებლობებით - ობიექტზე ორიენტირებული სტილი, SQL მონაცემთა ბაზებთან წვდომა და სხვა.

APL არის მათემატიკური ფორმალიზმის განვითარება, რომელიც გამოიყენება გამოთვლების აღსაწერად. ძირითადი მონაცემთა სტრუქტურები არის ვექტორები და მატრიცები, ხოლო ოპერაციები ხორციელდება ასეთ სტრუქტურებზე პირდაპირ, ციკლების გარეშე. APL–ის პროგრამები ძალიან მოკლეა ტრადიციულ ენებზე დაწერილ ანალოგიურ პროგრამებთან შედარებით. APL - ის გამოყენებას ართულებდა ის ფაქტი, რომ ენაში გადმოვიდა მათემატიკის სიმბოლოების დიდი ნაკრები. ეს საჭიროებდა სპეციალურ ტერმინალებს (კლავიატურებს) და აფერხებდა ექსპერიმენტებს APL - თან ძვირი აპარატურის გარეშე. თანამედროვე მომხმარებლის გრაფიკული ინტერფეისები პროგრამული შრიფტების გამოყენებით გახდა ამ პრობლემის გადაჭრა. რამდენიმე მაგალითი:



**გამოთვლადობა**

ჯერ კიდევ 1930-იან წლებში, ციფრული კომპიუტერების გამოგონებამდე, მეცნიერები ინტენსიურად მუშაობდნენ გამოთვლადობის თეორიაში, იკვლევდენ გამოთვლების აბსტრაქტულ კონცეფციებს. 1936 წელს ინგლისელმა მათემატიკოსმა, ლოგიკოსმა და კრიფტოგრაფმა **ალან ტიურინგმა** შექმნა აბსრტაქტული კომპიუტერის მათემატიკური მოდელი, რომელსაც ”ტიურინგის მანქანა” ეწოდა.

ტიურინგმა დაამტკიცა, რომ ასეთ მანქანას შეუძლია ნებისმიერი მათემატიკური გამოთვლების შესრულება, რომლებიც წარმოიდგინება ალგორითმის სახით. მან ასევე დასვა საკითხი მანქანის გაჩერების თაობაზე და აჩვენა, რომ მისი მანქანის გაჩერების პრობლემა გადაუჭრელია: ზოგადად შეუძლებელია ალგორითმულად განვსაზღვროთ გაჩერდება თუ არა ოდესმე მოცემული ტიურინგის მანქანა.

ამერიკელმა მათემატიკოსმა და ლოგიკოსმა **ალონზო ჩერჩმა** განავითარა გამოთვლადობის მოდელი (ლამბდა–კალკულუსი), რომელიც დაფუძნებული იყო ფუნქციის მათემატიკურ ცნებაზე. 1936 წელს თავის ცნობილ სტატიაში მან აჩვენა ე.წ. გადაუჭრელი ამოცანების არსებობა. ეს სტატია წინ უძღვოდა ალან ტიურინგის ცნობილ კვლევას „გაჩერების პრობლემის“ თაობაზე, რომელშიც ასევე იყო ნაჩვენები მექანიკური მეთოდებით გადაუჭრელი ამოცანების არსებობა.

მოგვიანებით, ჩერჩმა და ტიურინგმა აჩვენეს, რომ ლამბდა–აღრიცხვას და ტიურიგის მანქანას ერთნაირი თვისებები ჰქონდა, რაც ამტკიცებდა, რომ სხვადასხვა "მექანიკურ გამოთვლით პროცესებს" შეიძლება ჰქონდეს ერთნაირი შესაძლებლობები. მათი ერთობლივი ნაშრომის საფუძველზე ჩამოყალიბდა მტკიცებულება, ცნობილი, როგორც ჩერჩი - ტიურინგის თეზისი:

ნებისმიერი ეფექტური გამოთვლა შეიძლება შესრულდეს როგორც ტიურინგის მანქანაზე, ისე ლამბდა–აღრიცხვის გამოყენებით.

ტიურინგის მანქანები ძალზედ მარტივია. მათი აღწერისთვის გამოვიყენოთ C–ს სინტაქსი:

გვაქვს ორი განაცხადი მონაცემებზე:

char tape[. . .];

int current = 0;

პირველი განაცხადი გვიჩვენებს, რომ გვაქვს უსასრულო ლენტი. ეს არ ნიშნავს, რომ უნდა იყოს გამოყოფილი მეხსიერების სიტყვების უსასრულო რიცხვი, ნიშნავს მხოლოდ იმას, რომ ჩვენ შეგვიძლია საჭიროების მიხედვით უფრო მეტი მეხსიერების მოთხოვნა. პროგრამა შედგება ნებისმიერი რაოდენობის შემდეგი სახის შეტყობინებისაგან:

L17: if (tape[current] == 'g') {

tape[current++] = 'j';

goto L43;

}

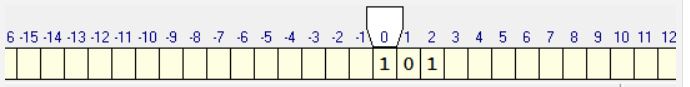
ტიურინგის მანქანის ინსტრუქციების შესრულება ხდება ოთხ ეტაპად:

* წაიკითხე და შეამოწმე მიმდინარე უჯრედის მიმდინარე სიმბოლო
* შეცვალე სიმბოლო სხვა სიმბოლოთი (არასავალდებულო)
* გადადი ერთი უჯრედით მარცხვნივ ან მარჯვნივ
* გადადი ნებისმიერ სხვა შეტყობინებაზე

ჩერჩი - ტიურინგის თეზისის თანახმად, ნებისმიერი გამოთვლა, რომელიც შეიძლება ეფექტურად აღიწეროს, შეგვიძლია დავაპროგრამოთ ამ პრიმიტიული მანქანაზე. თეზისის მტკიცებულება ეყრდნობა ორ ფაქტს:

* მკვლევარებმა შეადგინეს ათობით გამოთვლითი მოდელი და ყველა მათგანი აღმოჩნდა ტიურინგის მანქანების ექვივალენტური
* არასოდეს არავის აღუწერია გამოთვლები, რომელთა გამოთვლა შეუძლებელია ტიურინგის მანქანაზე

ვინაიდან ტიურინგის მანქანის მოდელირება ადვილად შესაძლებელია ნებისმიერ დაპროგრამების ენაზე, არსებული დაპროგრამების ენები „აკეთებენ“ ერთი და იგივეს.

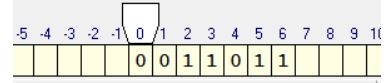


A={a1, a2, a3, …, an} — გარე ანბანი: საწყისი მონაცემების სიმრავლე. ჩვენ შემთხვევაში A={0, 1}

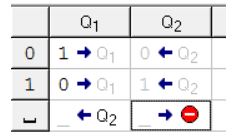
Q={ q0, q1, q2, q3,…, qm} — შიგა ანბანი: აღწერს წაკითხვა–ჩაწერის მოწყობილობის (თავაკის) მდგომარეობებს

q0 — გაჩერების მდგომარეობა: მასში მოხვედრისას მანქანა ამთავრებს მუშაობას – ჩერდება.

სავარჯიშო 1: ვთქვათ, A={ 0, 1, \_ }.

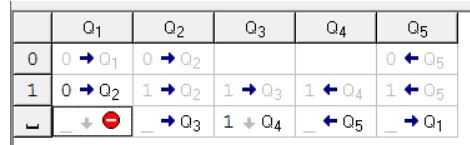


შეასრულეთ პროგრამა, რომელიც საწყის მიმდევრობას გაუკეთებს ინვერტირებას, და დააბრუნებს თავაკს საწყის პოზიციაში. თავიდან თავაკი დგას პირველ სიმბოლოზე.



სავარჯიშო 2: მიმდევრობა 001101011101 გარდავქმნათ შემდეგი სახით:

000000000000 1111111



სავარჯიშო 3: მოცემულია ნატურალური n რიცხვის ათობითი ჩანაწერი, n > 1. გავზარდოთ რიცხვი ერთით. თავაკი დგას რიცხვის ბოლო ციფრზე.