Razvoj programskog jezika C

Seminarski rad u okviru kursa
Dizajn programskih jezika
Matematički fakultet

Danica Babić
danicababic998@gmail.com

Milena Vidić
vidic.milena@gmail.com

20. decembar 2019

Sažetak

Programski jezik C osmišljen je ranih 1970-ih godina za potrebe sistemskog programiranja, ali je postao mnogo više od toga. Ovaj rad bavi se njegovom evolucijom, istorijom i razlozima nastanka. Opisani su njegovi prethodnici i jezici koji su najviše uticali na njegov razvoj: Fortran i Algol, CPL, BCPL i B. Izložene su njihove glavne osobine i istaknute one koje je C delimično ili u potpunosti nasledio. Pregled ovih uticaja dat je u vidu razvojnog stabla. Konačno, navedeni su najznačajniji jezici u čijem dizajnu je C bio uzor.

Sadržaj

1 Uvod

C je programski jezik opšte namene. Odlikuje ga ekonomičnost izraza, savremena kontrola toka izvršavanja i strukture podataka, kao i bogat skup operatora. C nije jezik "vrlo visokog nivoa", niti "veliki jezik", a nije ni specijalizovan za određenu oblast primene. Međutim, nepostojanje ograničenja njegove primene i njegova opštost ga čine pogodnijim i efikasnijim za mnoge zadatke od naizgled moćnijih jezika.

Neki jezici su s vremenom kategorisani uskim oblastima namene - Fortran, recimo, sa svojom namenom u inženjerstvu, dok se neki, poput COBOL-a, bore da ostanu relevantni. C ne samo da je ostao relevantan, već je istrajao kao inspiracija za razvoj brojnih i raznovrsnih programskih jezika. [13]

Slika 1: Logo programskog jezika C sa čuvenog K&R priručnika

2 Osnovno o jeziku C

Nastanak jezika C je blisko povezan sa operativnim sistemom Juniks.[1] Juniks je prvobitno bio napisan u asemblerskom jeziku, što je zahtevalo mnogo linija koda da bi se izveli jednostavni zadaci i teško održavanje i interpretaciju. Jezici Fortran i B[2] su delimično rešili taj problem, ali nisu bili potpuno dorasli zadatku. S tim ciljem je nastao jezik C, a Juniks je ponovo napisan, ovoga puta u C-u. Dizajnirao ga je i implementirao Denis Riči[3], američki naučnik u oblasti računarstva.

C je umnogome profitirao od rastuće popularnosti Juniksa. Iako nije očigledan prvi izbor za pisanje velikih komercijalnih aplikacija za obradu podataka, C ima veliku prednost u činjenici da je uvek dostupan na komercijalnim implementacijama ovog sistema. Kako je napisan u C-u, kad god je Juniks implementiran na novom tipu hardvera, prva stvar koju je trebalo obezbediti je da C kompilator radi na tom sistemu. Tako je Juniks učinio C dostupnim za stotine hiljada ljudi. Popularnost jezika rasla je i uz ekspanziju personalnih računara. U pisanju softvera C pruža čitljivost

[1] Juniks (eng. *Unix*) je prenosivi, multitasking, višekorisnički operativni sistem originalno razvijen 1969. godine u Belovim laboratorijama AT&T-a.[2]

[2] Programski jezik nastao u Belovim laboratorijama kasnih 1960-ih, o kojem će biti reči u daljem tekstu.

[3] *Dennis M. Ritchie*

i produktivnost na nivou visokih jezika, ali i moć da se maksimalno iskoristi arhitektura, bez potrebe da se pribegne asemblerskom jeziku. Ova dvoslojnost jezika, veoma poželjna na suparničkom tržištu PC softvera, čini C skoro jedinstvenim među programskim jezicima. Zahvaljujući tome se i danas mnogo koristi.[6]

Riči je, zajedno sa svojim dugogodišnjim kolegom Brajanom Kerniganom[4], svoju tvorevinu opisao u knjizi *Programski jezik C* (The C Programming Language) koja je i dan danas referentni priručnik jezika i *de facto* standard, takozvana *Bela knjiga*.[14]

ANSI (American National Standards Institute - Američki nacionalni zavod za standardizaciju) je 1983. formirao komitet sa ciljem da se standardizuje C. Rezultat je bio ANSI standard 1989.godine, takozvani *ANSI C* ili C89, koji je godinu dana kasnije prihvatila Međunarodna organizacija za standardizaciju (ISO).[13]

2.1 Karakteristike jezika C

Važno svojstvo koje C razlikuje od svojih prethodnika BCPL-a i B-a jeste postojanje tipova podataka: znakovni, celobrojni, brojevi u pokretnom zarezu različitih veličina, i cela hijerarhija izvedenih tipova formiranih pomoću pokazivača, nizova, struktura i unija. Izrazi se grade od operatora i operanada i svaki izraz može biti naredba. Pokazivači obezbeđuju mašinski nezavisnu adresnu aritmetiku. Dostupne su osnovne konstrukcije za kontrolu toka koje su neophodne za dobro strukturirane programe: grupisanje naredbi, grananje (if-else i switch), ponavljanje sa proverom uslova završetka petlje na početku (while, for) i na kraju (do) i prevremeno prekidanje petlje (break). Funkcije mogu vratiti vrednosti osnovnih tipova, strukture, unije i pokazivače; svaka funkcija se može zvati rekurzivno. Promenljive mogu imati različite dosege na nivou programa.[13]

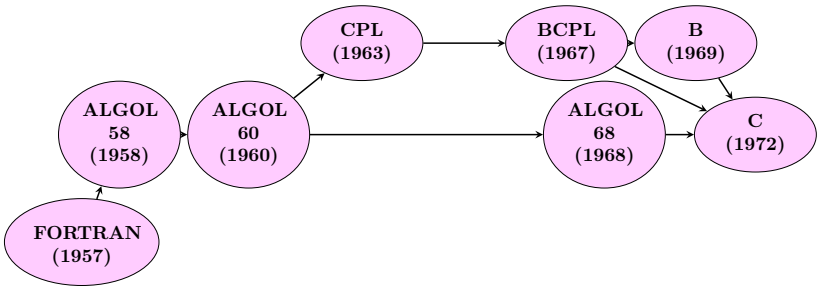
C je jezik relativno "niskog nivoa", što nije pežorativna karakterizacija, već naprosto govori da C manipuliše istom vrstom objekata sa kojima većina računara radi - brojevima, znakovima i adresama. Nizak nivo se ogleda i u nepostojanju operacija za direktno manipulisanje složenim objektima (stringovima, listama, nizovima). Ne postoji koncept hipa ili sakupljača otpadaka, kao ni ulazno-izlazni mehanizmi niti ugrađeni metodi za pristup datotekama, već svi ovi mehanizmi moraju biti naknadno obezbeđeni. C nudi samo jednonitni tok izvršavanja. Sve ovo su neodstaci sa aspekta udobnog, visoko apstrahovanog programiranja današnjice, ali očuvanje relativno skromne veličine jezika donosi koristi, čineći C malim jezikom koji se lako i brzo može naučiti, i koji se prevodi malim i jednostavnim kompilatorima.[13] Onima koji dobro poznaju kako računari rade, C omogućuje generisanje vremenski i prostorno visoko efikasnih programa.[14]

Konačno, u C-u se može dostići prenosivost programa. Naravno, kako je C kompiliran jezik, prenosivost se odnosi na to da, dok god ciljana platforma ima funkcionalan C prevodilac, dovoljno je izvorni kod prekompilirati na toj platformi bez potrebe da se modifikuje izvorni kod (*Write once, compile anywhere* - "Piši jednom, kompiliraj svuda").[13]

[4] *Brian W. Kernighan*, kanadski naučnik u oblasti računarstva

3 Razvojno stablo

Jezici **CPL** i **BCPL** su prethodnici jezika **B**, a B je bio od suštinskog značaja kao osnova jezika **C**. C, takođe, zajedno sa svojim prethodnicima, je preuzeo mnoštvo koncepata iz **Fortrana** i **Algola**, najstarijih imperativnih jezika. Razvojno stablo jezika prikazano je na slici 2.



Slika 2: Razvojno stablo jezika C

3.1 Fortran

Fortran je jezik koji se smatra prvim višim programskim jezikom. Iako je nastao 1957, decenijama je bio u neprekidnoj upotrebi u računski intenzivnim oblastima, naučnim istraživanjima i u obrazovanju, a dan danas živi u svetskim superkompjuterima. Fortran nije bio direktni prethodnik C-a. Međutim, kao prvi programski jezik i osnivač imperativne paradigme[5], on je uveo mnoge ideje od suštinskog značaja koje su mnogi jezici kasnije usvojili, a među njima i C i njegovi prethodnici. Ovo je važno jer su mnogi koncepti u imperativnom programiranju, koliko god intuitivno deluju u današnjem vremenu, bili sasvim novi pre najranijih programskih jezika. Jezici poput Fortrana i Algola su postavili uzore koje su dugo sledili i nadograđivali drugi jezici.

Fortran je razvio Džon Bakus[6] sa svojim timom iz kompanije IBM[7]. Prvenstvena namena mu je obavljanje velikog broja naučnih i matematičkih izračunavanja, pa je tako i dobio ime. (Fortran je skraćenica od *The IBM Mathematical **For**mula **Tran**slating System.*) Ideja je bila da se omogući unošenje matematičkih formula, koje bi se prevele u niz instrukcija koje računar može da čita i izvršava. Osim toga, Fortran je trebalo i da zameni asemblersko programiranje na IBM-ovom računaru 704.[3] Džon Bakus je rekao u jednom intervjuu: "Veliki deo mog rada je nastao zbog lenjosti. Nisam voleo da pišem programe, tako da, kada sam radio na IBM 701, pišući programe za izračunavanje putanja raketa, počeo sam da radim na sistemu programiranja koji bi mi olakšao posao."[8]

Kompilator za Fortran je bio je vrlo efikasan, što je jeziku obezbedilo uspeh jer se do tada verovalo da program napisan i kompiliran u jeziku visokog nivoa ne može raditi brzo kao program napisan ručno asemblerskim kodom. [3]

[5] Imperativno programiranje je programska paradigma koju karakteriše davanje naredbi računaru koje on treba da izvrši, za razliku od deklarativnog programiranja, gde se u programu opisuje kako rešenje problema treba da izgleda, bez opisivanja koraka.

[6] *John Backus*, američki naučnik u oblasti računarstva

[7] *IBM - International Business Machines Corporation* je američka kompanija koja se bavi računarskim tehnologijama, osnovana 1888.[9]

```
PROGRAM moj_prvi_program

! ODELJAK ZA DEKLARACIJU
INTEGER :: prviBroj
INTEGER :: drugiBroj
INTEGER :: proizvod

! POCETAK IZVRSNIH NAREDBI
WRITE(*, *) 'Unesite brojeve za mnozenje:'
READ(*, *) prviBroj, drugiBroj

proizvod = prviBroj * drugiBroj

WRITE(*, *) 'Rezulat je = ', proizvod

! ODELJAK ZA PREKID
STOP
END PROGRAM moj_prvi_program
```

Fortran program koji množi dva zadata broja

Program pisan u Fortranu se sastoji iz niza naredbi kojima se izvršava neki zadatak. Sadrži tri dela: deklaracije, izvršenje i prekid. Fortran je podržavao formatirane ulazno-izlazne operacije, imena promenljivih do 6 karaktera, korisnički definisane potprograme, IF naredbu izbora, i DO petlju. Imao je ugrađene tipove: celobrojni, realni, kompleksni, logički, karakterski, kao i konstante. Promenljive koje su počinjale slovima `I, J, K, L, M, N` su bile implicitno celobrojnog tipa, a ostale realnog; programeri su ovo mogli izbeći eksplicitnom deklaracijom. Tako je nastala čuvena šala: "In Fortran, GOD is REAL (unless declared INTEGER)."

Prvi Fortran kompilator je zaustavljao program kada se naiđe na grešku i prikazivao je odgovarajući kod, koji je programer mogao pronaći u tabeli grešaka, gde je ukratko opisan problem.[5]

Fortran je, između ostalog, uticao na sintaksu deklaracija: deklaracije u jeziku B počinju specifikatorom auto ili static, praćeno listom identifikatora, i C ne samo da prati ovaj stil, već ga dopunjava ključnim rečima tipova na početku deklaracija.[14]

3.1.1 Paradigma jezika C i Fortran kao njen osnivač

C je jezik imperativne paradigme, i praktično među poslednjim jezicima koji su bili suštinski jednoparadigmatski, za razliku od savremenih jezika. Ova paradigma je nastala sa Fortranom. On je uveo imenovane promenljive, kompleksne izraze, njihovu evaluaciju i dodelu rezultujuće vrednosti memorijskoj lokaciji, potprograme, uslovno i bezuslovno grananje, i mnoge druge karakteristike koje su sada uobičajene u imperativnim jezicima i koje C takođe poseduje.

Nije slučajno to što je upravo imperativna jedna od paradigmi najranijih jezika i zašto je tako važna u nameni C-a. Hardverska implementacija skoro svih računara je imperativna po prirodi: hardver je dizajniran da izvršava mašinski kod koji je računaru razumljiv i pisan imperativnim stilom. Iz ove perspektive, stanje programa definisano je sadržajem memorije, i naredbe su instrukcije računaru na mašinskom jeziku. Prvi imperativni jezici, a za njima i C, apstrahuju asemblersko programiranje, ali i dalje prate imperativnu paradigmu.

3.2 Algol

Algol (***Algo****rithmic* ***L****anguage*), prvobitno IAL (***I****nternational* ***A****lgebraic* ***L****anguage)* je imperativni, strukturirani[8] jezik. Nastao je kao rezultat međunarodne saradnje između Evrope i Amerike u Cirihu, 1958, sa ciljem da se stvori jezik što bliži matematičkoj notaciji, da programi budu čitljivi i da se mogu prevesti u mašinski jezik.[4] Napravljen je i kako bi se izbegli pojedini nedostaci Fortrana. Kroz godine, uticao je na mnoge nove jezike: PL/I, Simula, BCPL, B, C, Pascal, i stoga ima veliki značaj u istoriji programiranja. Brojne verzije, od kojih su najvažnije ALGOL 58, ALGOL 60, ALGOL 68, imenovane su po godini nastanka.

Na mnogo načina, prvobitni Algol je naslednik Fortrana. Generalizovao je mnogo njegovih karakteristika - na primer, imao je cilj da ne bude vezan za određenu mašinu, i da postane fleksibilan i jak jezik. Formalizovao je koncept tipova podataka. Identifikatori su mogli biti bilo koje dužine, za razliku od Fortranovog ograničenja na šest karaktera, a bila je dozvoljena i bilo koja dužina niza. Iz svega toga je nastala je retka kombinacija jednostavnosti i elegancije.[16]

Važne ideje koje uvodi ALGOL 60 su bile: koncept blokova koda, odvojenih ključnim rečima `begin` i `end` - što je većina kasnijih jezika preuzela (programeru omogućuje da lokalizuje delove programa); dozvoljeno je dva načina prosleđivanja parametara potprogramu: po vrednosti, i po imenu (*call-by-name*); omogućene su ugnježdene funkcije; procedure su sada mogle biti rekurzivne;[9] uveden je dinamički niz.[10] [16] Ovo su neki od vrlo korisnih koncepata u programiranju koje revnosno nasleđuju jezici koji nastaju posle Algola, i C ih uspešno realizuje.

Algol je dugo bio jedini prihvatljiv formalni način zapisivanja algoritama u računarskoj literaturi, i ostao je preko 30 godina jezik za objavljivanje algoritama Udruženja za računarske mašine.[11][1] Sa druge strane, nikada nije postao dominantan jezik u široj upotrebi. Na primer, prosleđivanje parametara potprogramima preko imena je mogućnost koju nisu preuzeli kasniji jezici. Još jedan razlog je nedostatak ulazno/izlaznih alata, te je on zavisio od implementacije i otežavao prenosivost. Na kraju, uticalo je i to što je korišćenje Fortrana već bilo duboko usađeno, kao i nedostatak podrške IBM-a.

Mnogi smatraju da je ALGOL 60 trebalo da zameni Fortran jer je između ostalog elegantniji i ima bolju kontrolu toka. To se nije desilo delimično jer programeri tog vremena nisu potpuno jasno shvatali konceptualni dizajn jezika. Bilo im je teško da ga razumeju, i nisu cenili prednosti strukture blokova, rekurzije, i dobro strukturirane kontrole toka, mada je značaj ovih koncepata i te kako shvaćen kasnije kroz Algolove naslednike poput jezika C i Pascal.[16]

Važan primer koncepta koji C najviše duguje Algolu 68 je shema tipova: ona se zasniva na atomičnim tipovima (uključujući strukture), a od koje se grade nizovi, pokazivači i funkcije. Algolov koncept unija i kastovanja je takođe izvršio direktan uticaj na C.[14]

[8] Strukturirano programiranje je programska paradigma koja za cilj ima poboljšanje kvaliteta, jasnoće i vremena razvoja programa. To se postiže korišćenjem naredbi kontrola toka (if, then, else), ponavljanja (while, for), blokova koda i potprograma.

[9] Iako je ovo novost za imperativne jezike, rekurzivne funkcije su već postojale u funkcionalnom LISP-u od 1959.

[10] Dinamički niz je niz kome se dodeljuju dužina i memorija tokom izvršenja programa, ali su od tog trenutka fiksne.

[11] *ACM - Association for Computing Machinery* je međunarodno udruženje za računarstvo, osnovano 1947. godine. Predstavlja najveće naučno i edukativno računarsko društvo.

3.3 CPL

CPL[12] je programski jezik osmišljen na Kembridžu ranih 1960-ih godina kao glavni jezik za moćan Feranti Atlas računar.[12]

CPL je imao nameru da istovremeno omogući programiranje niskog nivoa i visoku apstrakciju. To je bio jezik u algolovskoj tradiciji, ali sa mnogo novina i značajnim dodacima u cilju da se proširi domen njegove namene. Između ostalog, to je uključivalo bogatije kontrolne konstrukte poput `IF, UNLESS, WHILE, UNTIL, REPEATWHILE, SWITCHON` naredbe. Mogao je da radi sa različitim strukturama podataka i bio je jedan od prvih strogo tipiziranih jezika koji je omogućio strukturirani mehanizam za udobno rukovanje listama, stablima i usmerenim grafovima.

Međutim, rad na CPL-u, koji je trajao otprilike od 1961. do 1967.godine, eventualno je zapostavljen. Bio je preveliki i komplikovan za tadašnje dostupne mašine, i pragmatične stvari poput efikasnosti i implementabilnosti su zanemarene. Jezik se sporo razvijao i evenutalno praktično iščezao oko 1970-te, ali je imao ključnu ulogu kao osnova sledećeg jezika u razvojnom stablu jezika C, jezika BCPL. Kako je implementacija jezika prvobitno bila namenjena za EDSAC II, rani britanski računar, a na Atlas preneta na polovini projekta, javila se potreba za prenosivošću CPL kompilatora. Ovo je postignuto tako što je kompilator napisan u malom podskupu CPL-a, koji je potom mogao da se prevede i proširi do asebmlerskog koda za bilo koju od dve mašine. Jezik BCPL je bio inicijalno sličan ovom podskupu CPL-a.[12]

3.4 BCPL

BCPL, imperativni programski jezik koji je dizajnirao Martin Ričards[13] je u potpunosti implementiran 1967.godine na osnovi jezika CPL. Jezik je primarno imao sistemsku namenu, pre svega za pisanje kompilatora.[12]

BCPL omogućuje jednostavnu mašinski nezavisnu pokazivačku aritmetiku. I ovaj jezik se čvrsto uklapa u proceduralnu familiju vođenu Fortranom i Algolom. Mali je, sažeto opisan, sa brzim kompilatorom koji je lako prenosiv na nove mašine. BCPL je "blizak mašini", jer su apstrakcije koje uvodi dobro utemeljene na arhitekturi mašine i oslanjaju se na bibliotečke procedure za ulazno-izlazne operacije i druge interakcije sa operativnim sistemom.[14] Funkcije su rekurzivne i varijadičke, ali, sa ograničenjem koje je i C preuzeo: ne dozvoljavaju dinamičke slobodne promenljive.[14] Baš kao i u C-u, ne postoji ugrađeni sakupljač otpadaka, i sve ulazno-izlazne operacije se vrše bibliotečkim pozivima.[12] Pojedini sintaksički i leksički mehanizmi BCPL-a su čak elegantniji i precizniji nego kod naslednika B i C, recimo, skup petlji je kompletniji. Uprkos tome, većina naredbi i operatora jezika BCPL se preslikava direktno u odgovarajuće u jezicima B i C.

Ono što je važno je da je, za razliku od CPL-a, BCPL jezik bez tipova. Preciznije, on poseduje jedan jedini tip podatka, takozvanu *reč* ili *ćeliju*, skup bitova fiksirane dužine koji se uglavnom poklapa sa mašinskom reči.

[12] Originalno *Cambridge Programming Language*; kasnije je doneta odluka da razvoj CPL-a bude zajednički projekat između dva univerziteta, u Kembridžu i Londonu, te je ime promenjeno u *Combined Programming Language*.

[13] *Martin Richards*, britanski naučnik u oblasti računarstva

[14] U programiranju, termin slobodna promenljiva se odnosi na promenljive koje se koriste u funkciji, a koje nisu lokalne promenljive niti su parametri te funkcije.

Memorija se, stoga, sastoji iz linearnog niza ovakvih ćelija, a interpretacija sadržaja ćelije zavisi od operacije koja se primenjuje.

BCPL je prvi jezik koji je koristio vitičaste zagrade za odvajanje blokova koda. [14] Iako je sintaksno još uvek daleko od C-a, C od njega ipak nasleđuje mnogo identičnih koncepata preko jezika B.

Za programera se uvek očekuje da zna šta radi i on nije sputan sitničavim ograničenjima jezika. [17]

Ova rečenica o jeziku BCPL čiji je autor sam Ričards, govori dosta o filozofiji ovog jezika, koja je ostala važan aspekt C-a i njegove popularnosti kod programera.

3.5 B

B je programski jezik koji nastaje u Belovim laboratorijama, oko 1969.godine. Dizajnirali su ga Ken Tompson[15] i Denis Riči. Osmišljen je za nenumeričke, mašinski zavisne aplikacije, kao što je sistemski softver, i B, kao direktan naslednik BCPL-a, predstavlja poslednji jezik u lancu prethodnika jezika C kao njegov najuticajniji i najbliži predak. [10]

U jeziku B konačno se jasno prepoznaje sintaksa koju je C skoro u potpunosti preuzeo. Zanimljivo je što je B, kao i BCPL, jezik bez tipova, ali se za C ipak odlučuje da bude strogo tipiziran, baš kao i CPL. [14]

Struktura tipova i pravila evaluacije izraza iz BCPL-a ostaju neta knute kao nasleđene u jeziku B. B stoga takođe ima reč, odnosno ćeliju, kao jedini tip i meorija je analogno linearan niz ćelija, a vrednosti su interpretirane na osnovu operatora. Pokazivači u BCPL-u i B-u su jednostavno celobrojni indeksi memorijskog niza: ako je `p` adresa ćelije, `p+1` je adresa susedne ćelije. Ova konvencija je osnova semantike niza u oba jezika. Recimo, sa

```
v[10]
```

alocira se ćelija sa imenom v i 10 susednih ćelija, a memorijski indeks prve ćelije se smešta u v. Takođe, sa

```
*(v+i)
```

što je kao i u C-u ekvivalentno sa `v[i]`, a u BCPL-u sa `v!i`, pristupa se i-toj lokaciji od lokacije `v`. Ovakav pristup nizovima bio je neuobičajen čak i za ono vreme; C ga je kasnije asimilirao (uz indeksnu sintaksu pristupa nizu jezika B), štaviše u još manje konvencionalnom maniru.

Neki ne preterano *prijatni* aspekti BCPL-a su izbegnuti u dizajnu jezika B. Recimo, BCPL je koristio globalni vektor za komunikaciju između različitih odvojeno kompiliranih delova programa, tako što je programer ručno asocirao ime svake spolja vidljive procedure i podataka sa numeričkim indeksom u vektoru. B je oslobodio programera ovog tereta tako što koristi konvencionalni linker da razreši spoljna imena iz odvojeno kompiliranih datoteka. Ovakav koncept i rad linkera C je u potpunosti preuzeo.[14]

Svaki B program se sastoji iz jedne ili više funkcija, koje su slične Fortranovim funkcijama i potprogramima. Svaki program mora da sadrži *main* funkciju, na čijoj prvoj naredbi počinje izvršavanje programa, i obično se završava na poslednjoj. Main će pozivati obično druge funkcije iz istog programa ili iz biblioteka. Funkcije mogu biti rekurzivne.

[15] *Ken Thompson*, američki naučnik i jedan od pionira računarstva

Naredbe su sačinjene od izraza, a izrazi od operatora, identifikatora, konstanti, uz opcione zagrade. Generalno, naredbe, operatori, aritmetika, kondicionali, i sve ono što karakteriše jezik su skoro identični kao u jeziku C. B je, na primer, koristio ternarni operator `:?` koji preuzima C i većina jezika C-ovske sintakse: [10]

```
x = a < b?  a :  b;
```

Dakle, C je u pravom smislu te reči nasledio B: zadržao je mnogo toga što je B nudio, i zamenio ga kao nadograđena i poboljšana verzija jezika sa tipovima i strukturama podataka.

Zanimljivo je da je prvi put u istoriji programiranja zabeležen čuveni *"Hello world"* kod upravo u jeziku B, u Kerniganovom memorandumu *Uvod u programski jezik B*: [10]

```
main() {
    extrn a, b, c;
    putchar(a); putchar(b); putchar(c); putchar('!*n');
     }
a 'hell';
b 'o, w';
c 'orld';
```

Program ilustruje korišćenje karakterskih konstanti i promenljivih. Ono što se u B-u naziva karakterom jeste jedan do najviše četiri ASCII[16] karaktera, unutar jednostrukih navodnika. Karakteri su smešteni u jednu mašinsku reč. Eksterne, za sve funkcije vidljive promenljive `a, b, c` čuvaju takve karaktere.[10]

Ni B, dakle, nije u pravom smislu podržavao karakterski tip u jeziku. Mehanizmi za manipulisanje individualnim karakterima su bili loši i nepraktični; stringovi su se otpakivali u pojedinačne ćelije, te ponovo zapakivali nakon obrade karaktera. Takođe, pokazivačka aritmetika jezika B i BCPL je s vremenom postala nepogodna. Shema koja će rešiti probleme karaktera i pogodnog adresiranja bajtova, i pripremiti se za nadolazeći hardver koji je podržavao aritmetiku u pokretnom zarezu, bila je neophodna. Konačno, osim nedostataka samog jezika B, kompilator ovog jezika je generisao kod nedovoljno efikasno da bi zamenio asembler u prepisivanju nekih delova operativnog sistema u jeziku B. [14]

Suštinski progres u evolutivnom lancu jeste konačno uvođenje *tipova*, zajedno sa novim objektom koji ih povezuje elemente određenog tipa u niz, a zatim i pokazivačima na dati tip. Ovo je donelo i pravilo koje i danas živi u C-u: vrednosti nizovskog tipa se u izrazima konvertuju u pokazivač na prvi element niza. Sledećom naredbom se u takozvanom *novom B* (NB, *new B*, prelazni jezik između B i C) i C-u deklarišu redom `int`, pokazivač na `int`, i pokazivač na pokazivač na `int`:

```
int i, *pi, **ppi;
```

Kako su mašine koje procesuiraju ASCII postale uobičajene, konkretno DEC PDP-11 koji je pristigao u Belove laboratorije, postala je važna podrška za karakterski tip podatka. Priroda jezika B koji nije imao tipove je viđena kao mana, koja je Tompsona i Ričija navela da razviju proširenu verziju jezika koji podržava nove interne i korisnički definisane tipove, kao i strukture podataka. Do rane 1973. godine, bila je kreirana srž jezika C.[14]

[16] *American Standard Code for Information Interchange*, standard kodiranja karaktera u elektronskoj komunikaciji.

4 C i njegovi naslednici

Jezik C je neposredno ili posredno uticao na mnoštvo drugih programskih jezika, među kojima su C++, C#, D, Go, Java, JavaScript, Perl, PHP, Python, Rust, Juniksov skript jezik *shell*. Neki su nasledili C u pravom smislu te reči, preuzevši celokupan koncept jezika i dodavši klase, strukture podataka i/ili naprednije funkcije uz preuzimanje C-ove sintakse, a neki su preuzeli samo pojedine aspekte značajno se udaljivši od C-a u nameni. Mnoge konvencije, posebno sintaksne, poput operatora inkrementiranja (++), vitičastih zagrada ({}), i drugih idioma, postali su standard za celu familiju jezika. Već ubrzo posle nastanka C-a, novonastali jezici postaju mahom višeparadigmatski, te njegovi naslednici pružaju mogućnosti karakteristične za nekoliko paradigmi.

Možda najkonkretniji naslednik, *C++*, postao je prvi široko korišćen objektno-orijentisan jezik, nastao 1980-ih godina. U objektno orijentisanom jeziku *Java*, koja nastaje skoro deceniju kasnije, takođe se vidi jasan uticaj C-a i posebno C++-a. Međutim, ono što Javu razlikuje jeste njena interpretiranost, dok C++ zadržava efikasnost i mogućnost prilaza hardveru i upravljanje memorijom koju C pruža. Stoga i namena Jave pretežno naginje ka veb aplikacijama, dok se većina sistemskog programiranja i dalje obavlja u C-u i C++-u. *Objective C* je još jedan objektno-orijentisani jezik, koji kombinuje uticaje jezika Smalltalk i C. Značajan multiparadigmatski jezik koji je takođe inspirisan C-om je *C#*, koji podržava imperativnu, deklarativnu, generičku, objektno-orijentisanu, komponentnu i funkcionalnu paradigmu.[11]

Oko 2007. godine, tim inženjera iz Gugla, među kojima je bio i Ken Tompson, razvio je programski jezik *Go*.[7] To je C-ovski jezik, sa poboljšanom čitljivošću, produktivnošću i performansama - pogotovo što se tiče konkurentnosti i paralelizacije. Slično kao Java i mnogi drugi dinamički jezici, Go ima sakupljač otpadaka koji može dovesti do nepredvidljivosti u vremenu odziva i mogućih problema kod aplikacija koje se izvršavaju u realnom vremenu. Uprkos tome, Go je široko usvojen u Guglu, a i šire. Programi se brzo kompiliraju i izvršavaju.

Rust [15] je razvijan otprilike u isto vreme. Slično jeziku Go, Rust je sintaksno C-ovski jezik, ali dok Go insistira na brzini, Rust je više okrenut ka memorijskoj bezbednosti i predvidljivom ponašanju. Rust nema sakupljač smeća i akcenat stavlja na ručno upravljanje memorijom. Zbog bezbednosnih mera, Rust programeru nameće velika ograničenja; dok C i C++ svoj uspeh duguju tome što programer ima veliku slobodu u pisanju programa. Go i Rust su, uprkos ovim ograničenjima, zadržali C-ovsku prednost u odnosu na interpretirane dinamičke jezike, tamo gde je od suštinskog značaja brzina i efikasnost.[16]

5 Zaključak

U ovom radu je opisan razvoj jednog od najuticajnijih jezika današnjice: kako se rađala i realizovala ideja C-a, i koji jezici su u tome bili ključni. Ukratko su pomenuti i jezici u čijem je nastanku C bio uzor.

> *C je čudnovat, nesavršen, i ogroman uspeh.*
>
> (Denis Riči)

Literatura

[1] ACM. Collected Algorithms of the ACM. http://calgo.acm.org/.

[2] AT&T. Zvanična stranica kompanije AT&T. https://www.att.com/.

[3] John Backus. The History of Fortran I, II, and III. *IEEE Annals of the History of Computing*, 20(4), 1998.

[4] HT de Beer. The History of the ALGOL Effort. Master's thesis, Eindhoven University of Technology, 2006.

[5] Applied Science Division and IBM Programming Research Department, October 1956. http://archive.computerhistory.org/resources/text/Fortran/102649787.05.01.acc.pdf.

[6] Mike Banahan, Declan Brady, Mark Doran. *The C Book*. Addison Wesley, 1991.

[7] Go. Zvanična stranica programskog jezika Go. https://golang.org/.

[8] IBM. IBM Archives - John Backus. https://www.ibm.com/ibm/history/exhibits/builders/builders_backus3.html.

[9] IBM. Zvanična stranica kompanije IBM. https://www.ibm.com.

[10] S. C. Johnson, B. W. Kernighan. A tutorial introduction to the language B, 1973. https://www.bell-labs.com/usr/dmr/www/bintro.html.

[11] Allen B. Tucker, Robert E. Noonan. *Programming Languages: Principles and Paradigms*. McGraw Hill, 2nd edition, 2007. http://www.learngroup.org/uploads/2015-02-15/Programming_Languages_Principles_and_Paradigms_2nd_Edition_By_Tucker_Noonan.pdf.

[12] Martin Richards. The BCPL Cintsys and Cintpos User Guide by, 2015. https://www.cl.cam.ac.uk/~mr10/bcplman.pdf.

[13] Brian Kernighan, Dennis Ritchie. *The C Programming Language*. Prentice Hall, 1978.

[14] Dennis M. Ritchie. The Development of the C language. *SIGPLAN Not.*, 1993.

[15] Rust. Zvanična stranica programskog jezika Rust. https://www.rust-lang.org/.

[16] Robert W. Sebesta. *Concepts of Programming Languages*. Addison-Wesley, 7 edition, 2005.

[17] Martin Richards, Colin Whitby-Stevens. *BCPL: The Language and its Compiler*. Cambridge University Press, 1981.