Programiranje 1

Organizacija izvornog programa

Studijski program matematika

Pregled

Od izvornog do izvršnog programa

- Pisanje programa — prevođenje — izvršavanje
- Iako proces prevođenja jednostavnih programa početnicima izgleda kao jedan, nedeljiv proces, on se sastoji od više faza i podfaza (a od kojih se svaka i sama sastoji od više koraka).
- Osnovne faze prevođenja
  - Pretprocesiranje
  - Kompilacija
  - Povezivanje
- Tokom svih faza prevođenja, može biti otkrivena i prijavljena greška u programu.

Pregled

Pretprocesiranje

- Faza pretprocesiranja je pripremna faza kompilacije.
- Kompilator ne obrađuje tekst programa koji je napisao programer, već tekst programa koji je nastao pretprocesiranjem
- Rezultat rada pretprocesora, može se dobiti korišćenjem GCC prevodioca navođenjem opcije `-E` (na primer, `gcc -E program.c`).

Pretprocesiranje

- Jedan od najvažnijih zadataka pretprocesora je da omogući da se izvorni kôd pogodno organizuje u više ulaznih datoteka.
- Pretprocesor izvorni kôd iz različitih datoteka objedinjava u tzv. *jedinice prevođenja* i prosleđuje ih kompilatoru.
- Na primer, u `.c` datoteke koje sadrže izvorni kôd uključuju se *datoteke zaglavlja*

Pretprocesor

- Suštinski, pretprocesor vrši samo jednostavne operacije nad tekstualnim sadržajem programa i ne koristi nikakvo znanje o jeziku C
- Pretprocesor ne analizira značenje naredbi napisanih u jeziku C već samo *pretprocesorske direktive*

Pretprocesor

- Dve najčešće korišćene pretprocesorske direktive su `#include` (za uključivanje sadržaja neke druge datoteke) i `#define` koja zamenjuje neki tekst, *makro*, drugim tekstom.
- Pretprocesor omogućava i definisanje makroa sa argumentima, kao i uslovno prevođenje (određeni delovi izvornog programa se prevode samo ukoliko su ispunjeni zadati uslovi).

include

- U datoteku koja se prevodi, sadržaj neke druge datoteke uključuje se direktivom `#include`.

  ```
  #include "ime_datoteke"
  ili
  #include <ime_datoteke>
  ```

- U prvom slučaju, datoteka koja se uključuje traži se u okviru posebnog skupa direktorijuma *include path* (koja se većini kompilatora zadaje korišćenjem opcije `-I`) i koja obično podrazumevano sadrži direktorijum u kojem se nalazi datoteka u koju se vrši uključivanje.
- Ukoliko je ime, kao u drugom slučaju, navedeno između znakova < i >, datoteka se traži u sistemskom *include* direktorijumu u kojem se nalaze standardne datoteke zaglavlja, čija lokacija zavisi od sistema i od C prevodioca koji se koristi.

define

- Pretprocesorska direktiva `#define` omogućava zamenjivanje niza karaktera u datoteci, *makroa* (engl. macro) drugim nizom karaktera pre samog prevođenja. Njen opšti oblik je:

```
#define originalni_tekst novi_tekst
```

- U najjednostavnijem obliku, ova direktiva koristi se za zadavanje vrednosti nekom simboličkom imenu, na primer:

```
#define MAX_LEN 80
```

- O makroima je bilo reči nakon poglavlja o funkcijama

Kompilacija

- *leksička analiza* — izdvajanje *leksema*, osnovnih jezičkih elemenata;
- *sintaksička analiza* — kreiranje sintaksnog stabla;
- *semantička analiza* — provera semantike i transformacija kreiranog stabla;
- *generisanje međukôda* — generisanje kôda na jeziku interne reprezentacije;
- *optimizacija međukôda* — optimizovanje generisanog kôda;
- *generisanje kôda na mašinskom jeziku* — prevođenje optimizovanog kôda u objektne module.

Kompilacija

- Kompilacijom se od svake jedinice prevođenja gradi zasebni *objektni modul* (engl. object module)
- Objektni moduli sadrže programe (mašinski kôd funkcija) i podatke (memorijski prostor rezervisan za promenljive).
- Iako su u mašinskom obliku, objektni moduli se ne mogu izvršavati.

Povezivanje

- Povezivanje je proces kreiranja jedinstvene izvršne datoteke od jednog ili više objektnih modula
- Opcijama kompajlera, kompilaciju i povezivanje je moguće razdvojiti

Punilac

- Nakon uspešnog prevođenja može da sledi faza izvršavanja programa.
- Izvršni programi smešteni su u datotekama na disku i pre pokretanja učitavaju se u glavnu memoriju.
- Za ovo je zadužen program koji se naziva *punilac* ili *louder* (engl. loader) koji je obično integrisan u operativni sistem.
- Njegov zadatak je da proveri da li korisnik ima pravo da izvrši program, da prenese program u glavnu memoriju, da prekopira argumente komandne linije u odgovarajući deo memorije programa koji se pokreće, da inicijalizuju određene registre u procesoru i na kraju da pozovu početnu funkciju programa.

Pregled

Greške u programu

- I u fazi prevođenja i u fazi izvršavanja mogu se javiti greške i one moraju biti ispravljene da bi program funkcionisao ispravno.
- Greška se može javiti u bilo kojoj fazi prevođenja

Primer — pretprocesiranje

```
#Include<stdio.h>
int main() {
  int a = 9;
  if (a == 9)
     printf("9");
  return 0;
}
```

Primer — pretprocesiranje

```
#Include<stdio.h>
int main() {
  int a = 9;
  if (a == 9)
     printf("9");
  return 0;
}
```

```
primer.c:1:2: error: invalid preprocessing
                     directive #Include
```

Primer — kompilacija — leksička analiza

```
#include<stdio.h>
int main() {
  int a = 09;
  if (a == 9)
     printf("9");
  return 0;
}
```

Primer — kompilacija — leksička analiza

```
#include<stdio.h>
int main() {
  int a = 09;
  if (a == 9)
     printf("9");
  return 0;
}
```

```
primer.c:4:9: error: invalid digit "9" in octal constant
```

Primer — kompilacija — sintaksna analiza

```
#include<stdio.h>
int main() {
  int a = 9;
  if a == 9
     printf("9");
  return 0;
}
```

Primer — kompilacija — sintaksna analiza

```
#include<stdio.h>
int main() {
  int a = 9;
  if a == 9
     printf("9");
  return 0;
}
```

```
primer.c:5:6: error: expected ‘(’ before ‘a’
```

Primer — kompilacija — semantička analiza

```
#include<stdio.h>
int main() {
  int a = 9;
  if ("a" * 9)
     printf("9");
  return 0;
}
```

Primer — kompilacija — semantička analiza

```
#include<stdio.h>
int main() {
  int a = 9;
  if ("a" * 9)
     printf("9");
  return 0;
}
```

```
primer.c:5:8: error: invalid operands to binary *
                     (have ‘char *’ and ‘int’)
```

Primer — povezivanje

```
#include<stdio.h>
int main() {
  int a = 9;
  if (a == 9)
     print("9");
  return 0;
}
```

Primer — povezivanje

```
#include<stdio.h>
int main() {
  int a = 9;
  if (a == 9)
     print("9");
  return 0;
}
```

```
primer.c:(.text+0x20): undefined reference to ‘print’
collect2: ld returned 1 exit status
```

Primer — upozorenja

```
#include<stdio.h>
int main() {
  int a = 9;
  if (a = 9)
     printf("9");
  return 0;
}
```

Primer — upozorenja

```
#include<stdio.h>
int main() {
  int a = 9;
  if (a = 9)
     printf("9");
  return 0;
}
```

```
primer.c:4: warning: suggest parentheses around assignment
            used as truth value
```

Primer — upozorenja

```
#include<stdio.h>
int main() {
  int a = 9;
  if (a / 0)
     printf("9");
  return 0;
}
```

Primer — upozorenja

```
#include<stdio.h>
int main() {
  int a = 9;
  if (a / 0)
     printf("9");
  return 0;
}
```

```
primer.c:4: warning: division by zero
```

Primer — upozorenja

```
#include<stdio.h>
int main() {
  int a = 9;
  if (9 == 9)
     printf("9");
  return 0;
}
```

Primer — upozorenja

```
#include<stdio.h>
int main() {
  int a = 9;
  if (9 == 9)
     printf("9");
  return 0;
}
```

```
primer.c:3: warning: unused variable ‘a’
```

Greške u programu

- I u fazi prevođenja i u fazi izvršavanja mogu se javiti greške i one moraju biti ispravljene da bi program funkcionisao ispravno.
- Greške tokom izvršavanja mogu biti takve da program ne može dalje da nastavi sa izvršavanjem, ali mogu da budu takve da se program nesmetano izvršava, ali da su rezultati koje prikazuje pogrešni.
- Dakle, to što ne postoje greške u fazi prevođenja i to što se program nesmetano izvršava, još uvek ne znači da je program ispravan, tj. da zadovoljava svoju specifikaciju i daje tačne rezultate za sve vrednosti ulaznih parametara.

Greške u programu

- Ispravnost programa u tom, dubljem smislu zahteva formalnu analizu i ispitivanje te vrste ispravnosti najčešće je van moći automatskih alata.
- Ipak, sredstva za prijavljivanje grešaka tokom prevođenja i izvršavanja programa znatno olakšavaju proces programiranja i često ukazuju i na suštinske propuste koji narušavaju ispravnost programa.

Pronalaženje grešaka

- Pre konačne distribucije korisnicima, program se obično intenzivno testira tj. izvršava za različite vrednosti ulaznih parametara.
- Ukoliko se otkrije da postoji greška tokom izvršavanja (tzv. *bag* od engl. bug), vrši se pronalaženje uzroka greške u izvornom programu (tzv. *debagovanje*) i njeno ispravljanje.
- U pronalaženju greške mogu pomoći programi koji se nazivaju *debageri* i koji omogućavaju izvršavanje programa korak po korak (ili pauziranje njegovog izvršavanja u nekim karakterističnim tačkama), uz prikaz međuvrednosti promenljivih.

Literatura

Slajdovi su pripremljeni na osnovu materijala iz devetog poglavlja knjige:
Filip Marić, Predrag Janičić: Programiranje 1
Nastali su dopunom slajdova prof. dr Milene Vujošević Janičić.
Za pripremu ispita nisu dovoljni slajdovi, potrebno je koristiti knjigu!