آزمون نهایی درس طراحی کامپایلر

مجموع نمره‌ها: ۱۰۰ زمان آزمون: ۱۱۰ دقیقه

---

۱) گرامر روبرو با الفبای $\{x, y, \sim, (,), +, -, *\}$ را در نظر بگیرید.

```
S → +SS
S → -SS
S → *SS
S → A
A → (S)
A → ~S
A → x
A → y
```

۱.۱ (۳) چپ‌ترین اشتقاق (Left-most derivation) و درخت تجزیه (Parse tree) را برای رشته‌ی «$+ - x\ x\ (* x \sim y)$» بدست آورید.

۲.۱ (۷) توابع FIRST و FOLLOW را برای سمبل‌های S و A محاسبه نمایید.

۳.۱ (۷) جدول LL(1) را برای این گرامر تشکیل دهید. آیا این گرامر LL(1) است؟

۴.۱ (۳) با نشان دادن وضعیت پشته و ورودی، رشته‌ی سؤال ۱.۱ را تجزیه کنید.

۲) گرامر روبرو با الفبای $\{x, y, \sim, (,), +, -, *\}$ را در نظر بگیرید.

```
S → SS+
S → SS-
S → SS*
S → S~
S → (S)
S → a
S → b
```

۱.۲ (۱۰) مجموعه‌ی Itemset-های الگوریتم LALR(1) را برای این گرامر محاسبه کنید.

۲.۲ (۱۰) جدول ACTION و GOTO را برای الگوریتم LALR(1) بسازید. آیا این گرامر LALR(1) است؟

۳.۲ (۵) رشته‌ی «$x\ x - (x\ y \sim *) +$» را با استفاده از جدول LALR(1) و نمایش وضعیت پشته و ورودی تجزیه نمایید.

۳) کد میانی با کد سه-آدرسه (Three-address code) روبرو را در نظر بگیرید.

```
01   n = 1
02   i = 0
03   if i >= p goto 13
04   t1 = n * b
05   n = t1
06   if n > 200 goto 10
07   t2 = n - 200
08   n = t2
09   goto 06
10   t3 = i + 1
11   i = t3
12   goto 03
13   goto 16
14   if n >= 100 goto 16
15   n = 100
16   r = n + b
```

۱.۳ (۵) بلوک‌های پایه (Basic Blocks) را بدست آورید.

۲.۳ (۵) گراف جریان (Flow graph) را بکشید.

۳.۳ (۵) حلقه‌های (Loops) موجود در کد میانی را با بیان روش، تشخیص دهید. تشخیص حلقه‌ها چگونه می‌تواند در بهینه‌سازی‌ها و اختصاص رجیستر کمک کند؟

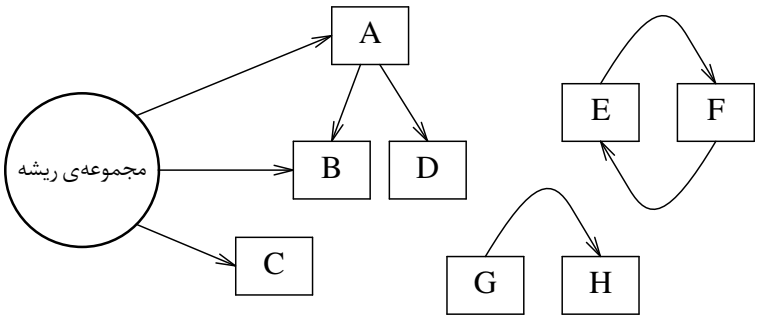
۴.۳ (۳) بلوک‌های پایه‌ای که می‌توان از کد میانی حذف کرد را با دلیل مشخص کنید.

۵.۳ (۹) با تحلیل زنده بودن سراسری (Liveness Analysis) روی بلوک‌های پایه‌ی کد میانی، متغیرهای زنده در شروع و پایان هر بلوک پایه را تشخیص دهید.

۴) (۱۲) در شکل روبرو، متغیرهایی که در هر دستور از یک کد میانی زنده هستند، نشان داده شده‌اند. با توجه به همپوشانی بازه‌های زنده بودن متغیرها، گراف تداخل رجیستر (Register Interference Graph) را بکشید و با استفاده از رنگ‌آمیزی گراف، دو رجیستر را به متغیرها اختصاص دهید. مشخص کنید کدام متغیرها در حافظه قرار می‌گیرند.

```
1: {a}
2: {a, b}
3: {b, c, d}
4: {c, d}
5: {b, d}
6: {d, e}
```

۵) در شکل زیر اشیاء (Objects) موجود در حافظه با یک مستطیل نمایش داده شده‌اند و یک یال جهت‌دار شی A را به شی B وصل می‌کند اگر A به B اشاره کند (یا یک Reference از آن را نگه دارد). اشیائی که با استفاده از یک نام در برنامه قابل دسترسی هستند (مجموعه‌ی ریشه یا «Root set») با یال‌های جهت‌داری از دایره‌ی سمت چپ شکل مشخص شده‌اند.



۱.۵ (۳) زباله‌ها را با بیان دلیل مشخص کنید.

۲.۵ (۵) تعداد اشاره‌ها (References) را برای هر یک از اشیاء محاسبه کنید و با استفاده از الگوریتم «Reference-counting»، زباله‌ها را تشخیص دهید. چه مشکلی در این روش جمع‌آوری زباله مشاهده می‌کنید؟

۳.۵ (۳) مشابه الگوریتم‌های «Trace-based» (شناسایی اشیاء قابل دسترسی با پیمایش اشاره‌گرها از مجموعه‌ی ریشه) زباله‌ها را تشخیص دهید.

۴.۵ (۵) تأثیر تشخیص خودکار زباله بر سرعت اجرای برنامه‌ها را از دو جنبه بیان کنید.