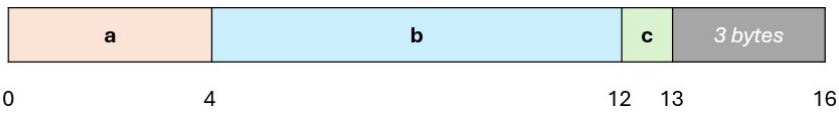
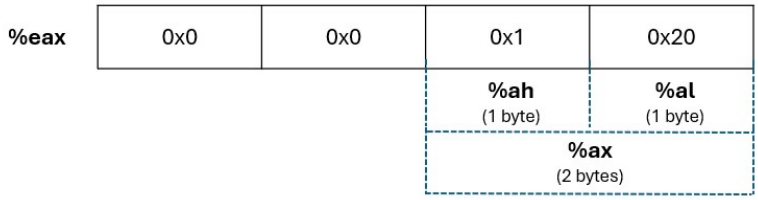
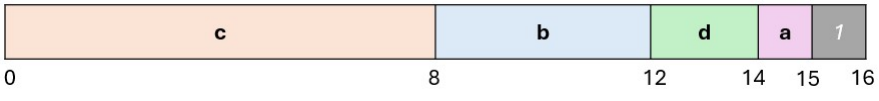
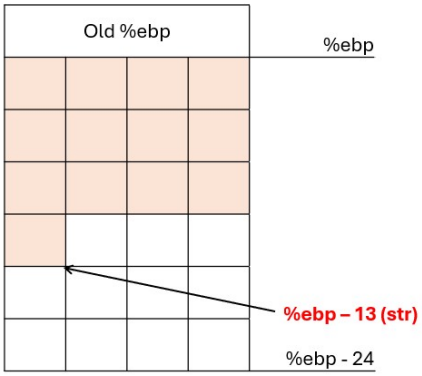
Câu	Đáp án	Giải thích
1	A	Với mảng short a[5], ta có &a[i] = &a + i*2 = %eax + %edx*2 nên ta có biểu thức địa chỉ là (%eax, %edx, 2). Lưu ý a[i] có kích thước 2 byte, cần dùng movw để lấy đúng 2 byte tại ô nhớ của a[i], không lấy dư.
2	A	Giá trị trả về của hàm trong kiến trúc 64bit được lưu trong thanh ghi, cụ thể là %rax
3	B	%ebp trỏ đến vị trí lưu old %ebp của hàm mẹ, ô nhớ lưu địa chỉ trả về là vị trí %ebp + 4, tham số thứ nhất lưu ở %ebp + 8. Trong stack không lưu giá trị trả về của hàm, giá trị này lưu trong thanh ghi %eax.
4	D	Các hàm và biến toàn cục, dù có static hay không đều là symbol. Với biến cục bộ trong hàm, nếu có static thì là symbol, biến cục bộ thường (không static) không phải symbol.
5	D	Trong gọi hàm trong kiến trúc 64 bit, 6 tham số đầu tiên được truyền qua thanh ghi, từ tham số thứ 7 được đưa vào stack. Với hệ thống 32bit, tất cả tham số được đưa vào stack.
6	D	Với lệnh pushl %ebp trong 32 bit, %esp bị giảm xuống 4 đơn vị, %ebp (thanh ghi source của lệnh push) không đổi
7	D	Stackframe hàm mẹ nằm ở địa chỉ cao hơn stackframe hàm con, được cấp phát trước và thu hồi sau stackframe hàm con. Các hàm không thể truy xuất biến cục bộ của nhau.
8	C	Kích thước của mảng 2 chiều: sizeof(T)*R*C = sizeof(short)*5*5 = 2*5*5 = 50 bytes
9	B	Sau lệnh 1 (pushl), %esp giảm 4 còn 0x1024. Sau lệnh 2 (subl), %esp giảm 20 (= 0x14) nên kết quả là 0x1010.
10	B	Yêu cầu căn chỉnh chung K của struct là yêu cầu căn chỉnh lớn nhất của các thành phần trong struct. float a có kích thước kiểu dữ liệu là 4 bytes nên K(a) = 4. double b có kích thước 8 bytes nhưng trong Linux 32 bit (trường hợp ngoại lệ) thì K(b) = 4. char c có kích thước 1 nên K(c) = 1. Vậy yêu cầu căn chỉnh chung K của struct là max{4,4,1} = 4
11	B	Khi cấp phát struct, các thành phần được cấp phát theo đúng thứ tự khai báo trong struct. Thành phần nào khai báo trước được cấp phát trước và nằm ở các ô nhớ có địa chỉ thấp hơn, các thành phần càng phía sau (cấp phát sau) thì địa chỉ lưu càng cao. Như vậy theo thứ tự khai báo, thành phần a có địa chỉ thấp nhất, c có địa chỉ cao nhất.
12	B	Vẽ cấp phát struct (có alignment) như hình dưới. Lưu ý khi có alignment, cần đảm bảo địa chỉ của mỗi thành phần chia hết cho yêu cầu căn chỉnh của kiểu dữ liệu tương ứng: - Địa chỉ của a chia hết cho 4. - Địa chỉ của b chia hết cho 4 (Linux 32bit chỉ yêu cầu địa chỉ của double chia hết cho 4). - Địa chỉ của c chia hết cho 1. Các thành phần được cấp nhất vùng nhớ có kích thước bằng đúng kích thước kiểu dữ liệu tương ứng (4 bytes cho kiểu float, 8 byte cho kiểu double, 1 byte cho kiểu char). Ngoài ra, khi xét yêu cầu căn chỉnh chung của struct, tổng kích thước struct cần chia hết cho K chung của nó (= 4), nếu chưa thì thêm byte trống. Do vậy, khi cấp phát xong thành phần c thì kích thước của struct là 13 bytes, chưa chia hết cho K chung là 4, nên cần 3 byte trống chèn thêm (màu xám).  Vậy kích thước của struct khi căn chỉnh trong Linux 32 bit là 16 bytes.

13	A	Khi không căn chỉnh, tổng kích thước struct bằng tổng kích thước các thành phần cộng lại: 4 (float) + 8 (double) + 1 (char) = 13 bytes. So với kích thước struct khi có căn chỉnh đã tính ở câu 12, ta có chênh lệnh 3 bytes giữa 2 trường hợp.
14	B	Kích thước union là kích thước của thành phần có kích thước lớn nhất. float a có kích thước 4 bytes, char c có kích thước 1 bytes, double b có kích thước 8 bytes. Vậy kích thước union là 8 bytes.
15	D	Các thành phần trong union đều có chung địa chỉ bắt đầu là địa chỉ của union. Như vậy biểu thức biểu diễn địa chỉ của thành phần c là (%eax). Thành phần c là kiểu char (1 byte) nên để lấy đúng 1 byte cần dùng movb. Câu lệnh đúng: movb (%eax), %bl (ĐÁP ÁN KHÁC)
16	C	Biến cục bộ (không có static) là các biến cục bộ thông thường của một hàm, sẽ nằm trong stackframe của hàm đó. Nhưng nếu thêm keyword static khi khai báo, nó sẽ nằm trong section .data nếu đã có khởi tạo giá trị hoặc .bss nếu chưa được khởi tạo giá trị.
17	D	Linker không thể liên kết 2 file source có định nghĩa 2 hàm trùng tên do đây là trường hợp phân giải định nghĩa symbol trùng tên, 2 hàm đều là 2 strong symbol nên lỗi linker. Mặt khác linker nhận đầu vào là file .o, ngay cả chỉ có 1 file .o vẫn phải dùng linker (như Lab 2 và 3). Linker mặc định làm việc với hàm và biến toàn cục.
18	B	Ô nhớ trỏ đến bởi %ebp là ô nhớ lưu old %ebp của hàm mẹ, ghi đè tùy ý có thể gây ra lỗi Segmentation fault. Ghi đè ô nhớ chứa địa chỉ trả về cũng gây lỗi. Mặt khác, chuỗi input chỉ ghi đè được các ô nhớ nằm ở địa chỉ CAO hơn nó (do chuỗi ghi dần từ địa chỉ thấp đến địa chỉ cao). Ngoài ra, trong stack frame hàm không tồn tại ô nhớ chứa giá trị trả về và biến toàn cục.
19	D	%eax = 0x120. Ta có hình dưới minh họa vị trí và giá trị của các thanh ghi con của %eax.  Như hình ta thấy %ah là thanh ghi 1 byte nằm trong thanh ghi %eax. Với %eax = 0x120, ta có %ah = 0x1. Câu lệnh 2 gán 0x0 cho %ah, khi đó giá trị của %eax sẽ là 0x20. Lệnh 3 tăng %eax lên 1 đơn vị nên %eax = 0x21 (hexan) hoặc 33 (hệ 10).
20	B	Cùng một khai báo nhưng mảng có kích thước khác nhau trong 32 bit và 64 bit là do kích thước kiểu dữ liệu khác nhau giữa 2 hệ thống. long hoặc kiểu pointer là kiểu dữ liệu như vậy.
21	C	Trường hợp cấp phát như khai báo: tổng kích thước 20 bytes (K chung = 4).  Trường hợp tối ưu: sắp xếp lại các thành phần trong struct theo thứ tự giảm dần của kích thước dữ liệu, ta có struct ex {double c; char* b; short d; char a;} có tổng kích thước 16 bytes (K chung = 4).  Như vậy sau khi tối ưu ta giảm được 4 bytes.

22	D	sym1 có trong symbol table, tức là nó là 1 symbol. Cột Bind cho biết kiểu symbol là GLOBAL hay LOCAL. Mặt khác Ndx cho biết chỉ số của section có chứa định nghĩa symbol trong file .o, nếu là UND (Undefined) nghĩa là không có định nghĩa của symbol trong file .o hiện tại, mà nằm ở file khác (External symbol).
23	C	Để xác định đúng độ dài gây lỗi và không gây lỗi, cần sử dụng code assembly để vẽ được stackframe của hàm func như hình bên: - Lệnh push đưa old %ebp (của hàm mẹ) vào stack. - Gán giá trị %esp cho %ebp, khi đó %ebp sẽ trỏ đến ô nhớ đang lưu old %ebp. - Trừ %esp để tạo stackframe (-0x18 = -24 bytes) - Xác định vị trí chuỗi str (tham số duy nhất của hàm gets). Trước khi call gets, lệnh leal lấy địa chỉ vị trí -0xd(%ebp), tức %ebp – 13 và push vào stack làm tham số cho lệnh call gets. Chỗ này là vị trí của chuỗi str.  Khi nhập chuỗi, các ký tự của chuỗi sẽ được lưu từ vị trí %ebp – 13 đến địa chỉ cao hơn, nếu đè Old %ebp sẽ lỗi. Cần lưu ý khi nhập từ bàn phím, sẽ có thêm ký tự null chèn thêm sau chuỗi. Do đó nếu nhập 13 ký tự 'A', thực chất sẽ có 14 byte được thêm vào stack, gồm 13 bytes nằm trong 13 ô màu cam, 1 byte null (0x00) nằm trên ô nhớ có địa chỉ thấp nhất của Old %ebp, ghi đè giá trị cũ → Lỗi Segmentation fault. Càng tăng kích thước chuỗi thì vẫn lỗi. Như vậy nhập 13 ký tự 'A' là có thể gây lỗi Segmentation fault, chuỗi ngắn hơn thì các ký tự vẫn nằm trong các ô màu cam, chưa ghi đè giá trị quan trọng nên không gây lỗi.
24	B	Ta có tổng kích thước của mảng B là 4*N*5 = 20*N, với N là số nguyên. Mặt khác mảng có tồn tại B[2][3] nên N > 2 nên kích thước của mảng B > 40 bytes và chia hết cho 20. Từ đó ta có đáp án là 60 bytes là phù hợp.
25	C	Công thức tính địa chỉ của phần tử trong mảng 2 chiều: &B[i][j] = B + sizeof(T)*i*C + sizeof(T)*j Áp dụng vào mảng int B[N][5] với địa chỉ bắt đầu là 0x1010, ta có: &B[2][3] = B + 4*2*5 + 4*3 = 0x1010 + 40 + 12 = 0x1044