Solutions to final exam

Problem 1: C-strings

1a) Tessi

1b)

(*p_input) += pos;
(*p_input)[len] = '\0';

1c)

char *ptr = name;
substring(&ptr, 3, 2);
printf("%s\n", ptr);

Problem 2: Generics

2a)

void *end = (char *)base + (*p_nelems)*width;
memcpy(addr, min, width);
memmove(min, (char *)min + width, (char *)end - ((char *)min + width));
(*p_nelems)--;

2b)

return strlen(*(const char **)p) - strlen(*(const char **)q);

extract_min(&min, words, p_nwords, sizeof(*words), cmp_len);
Problem 3: Floats
3a) 1 10000001 00100000000000000000000
3b) 1.5
3c) inf
3d) nan
3e) true

Problem 4: Assembly
4a)
```c
int pinky(char *param1, int *param2)
{
    char *str = NULL;
    int local = strtol(param1, &str, 16);
    if (local - 7 > 12)
        local += param2[3];
    return local/4;
}
```
4b) Substituting a cheaper instruction for an expensive one. In this case the expensive instruction is div. If divisor is $2^N$, can compute division as bit shift right $N$ positions (with fixup to round negative numbers toward zero). Even though this trades four instructions for one, still comes out ahead because div is so heavyweight.
4c) Line 3: Unsigned comparison in place of signed (jle -> jbe).
Line 5: Unsigned divide does not require negative fixup, divide by is $2^N$ just logical bit shift right $N$ positions. First four instructions at .L1 are replaced with shr $0x2$

Problem 5: Runtime stack
5a) During the execution of concat, the strcpy/strcat overflow result array and overwrite saved %rbx and return address. At end of concat, retq instruction pops the garbage address from stack into %rip and attempts to resume execution at that location -> boom!
5b) When a stack frame is deallocated, the memory remains accessible and contents remain as-is until a subsequent stack frame overwrites it. The stack frame for printf is overwriting what was the stack frame for concat, but since concat had a very large frame and printf has a much smaller one, it is overwriting tail end of the stack array (unused), and never reaches down to the used part where the characters is stored.
5c) Now that the concat frame is tightly-sized, the printf frame is writing on the used contents of the stack array, garbling the string contents.
5d) On entry to function, stack protector writes a canary value on the stack next to saved registers. On exit, reads canary from stack to verify value is intact. If function overflows past end of local variable into saved registers, it will overwrite the canary in the process and stack protector will detect it.
5e) The too-small version halts with error "stack smashing detected".
Problem 6: Heap

6a)
```c
#define USED (1L << ((sizeof(long) << 3) - 1))
#define NWORDS ~(USED)
```

6b)
```c
*hdr & USED
```

6c)
```c
void *neighbor = hdr + (*hdr & NWORDS) + 1;
if (neighbor >= segment_end) return NULL;
return neighbor;
```

6d)
```c
prev = &free_list
```

6e) Saves 2-3 instructions per iteration, O(N) where N is number of entries on free list

6f)
```c
prev = *prev;
```

6g)
```c
*prev = *(void **)to_remove;
```

6h) There is a mismatch in units – nbytes is being compared to number of words. This will erroneously reject blocks that could be used because it believes them too small.

6i)
```c
*cur += *neighbor + 1; // can mask but don't need to, neighbor not in-use
remove_from_freelist(neighbor + 1);
```

6j) There is no way to directly access the block header of a left neighbor. Walking the block headers from segment_start (i.e. traverse implicit list) could find it, but in worst-case requires a complete linear traversal of the entire heap! Even if redesigned to enable direct access to left neighbor (e.g. add footer), expanding to the left for myrealloc still has to copy the payload data, which is the expensive operation you want to avoid by resize-in-place.