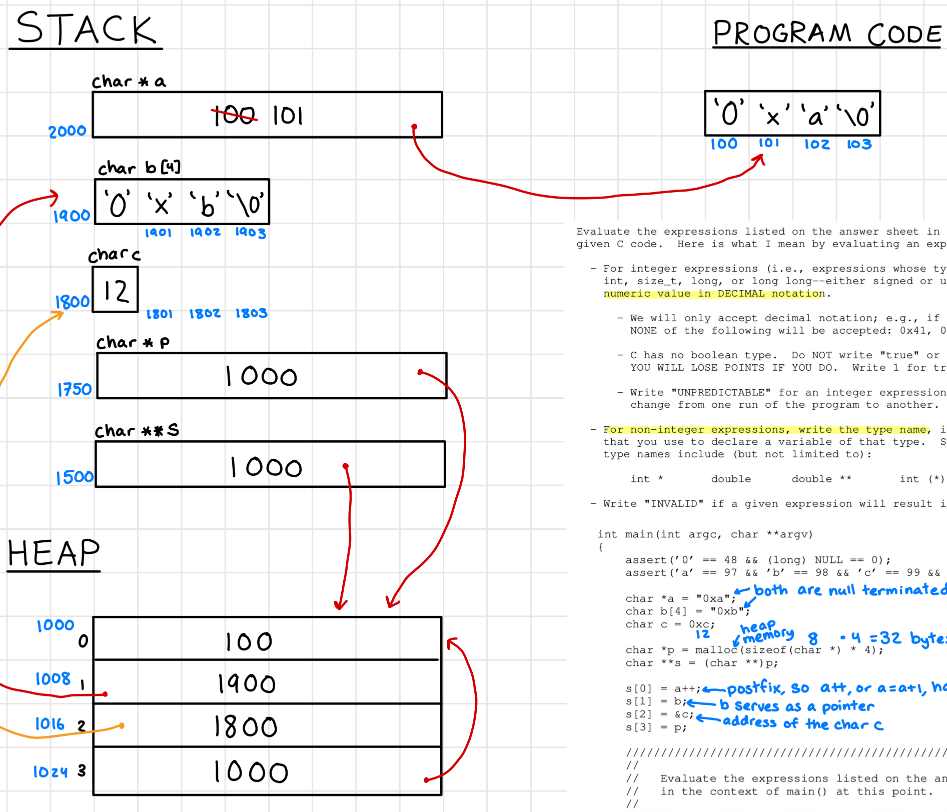


string literals cannot be changed. So these are stored in 'program code' part of memory

Evaluate the expressions listed on the answer sheet in the context of the given C code. Here is what I mean by evaluating an expression:

- For integer expressions (i.e., expressions whose types are char, short, int, size_t, long, or long long--either signed or unsigned), write the numeric value in DECIMAL notation.

  - We will only accept decimal notation; e.g., if the answer is 65, NONE of the following will be accepted: 0x41, 01000001, 'A', 2^6+1.

  - C has no boolean type. Do NOT write "true" or "false". YOU WILL LOSE POINTS IF YOU DO. Write 1 for true and 0 for false.

  - Write "UNPREDICTABLE" for an integer expression whose value can change from one run of the program to another.

- For non-integer expressions, write the type name, in the format that you use to declare a variable of that type. Some example type names include (but not limited to):

```
    int *       double       double **       int (*)(int *, int *)
```

- Write "INVALID" if a given expression will result in a compiler error.

```
int main(int argc, char **argv)
{
    assert('0' == 48 && (long) NULL == 0);
    assert('a' == 97 && 'b' == 98 && 'c' == 99 && 'x' == 120);

    char *a = "0xa";   ← both are null terminated
    char b[4] = "0xb"; ↙
    char c = 0xc;
             12      heap memory   8   • 4 =32 bytes allocated, returns its address
    char *p = malloc(sizeof(char *) * 4);
    char **s = (char **)p;

    s[0] = a++;  ← postfix, so a++, or a=a+1, happens after s[0] is set to a
    s[1] = b;    ← b serves as a pointer
    s[2] = &c;   ← address of the char c
    s[3] = p;

    ////////////////////////////////////////////////////////////////
    //                                                            //
    //   Evaluate the expressions listed on the answer sheet     //
    //   in the context of main() at this point.                 //
    //                                                            //
    ////////////////////////////////////////////////////////////////

    free(s);

    return 0;
}
```

[1]

(1.1) *a points to 'x', then get value of 'x' = $\boxed{120}$

(1.2) b + 1 $\boxed{\text{char *}}$, b is acting as a pointer's first element. so b points to 1900. b+1 holds mem. address 1901, so it's of type char *

(1.3) c + 2 C = 12, so C+2 = $\boxed{14}$

(1.4) (long) argv[argc] (long) 0 = $\boxed{0}$
↑ always NULL

(1.5) sizeof(b) Char array of 4, so $\boxed{4}$

(1.6) strlen(b + 1) b+1 points to 'x' in "0xb", so strlen("xb") = $\boxed{2}$

(1.7) sizeof(a) == sizeof(b + 2) Sizeof (char *) $\underset{\text{true}}{==}$ sizeof (char *) = $\boxed{1}$
b+2 is acting as a pointer to a char because pointer arithmetic is decaying it to a char *. If we did sizeof(b), it would still be 4.

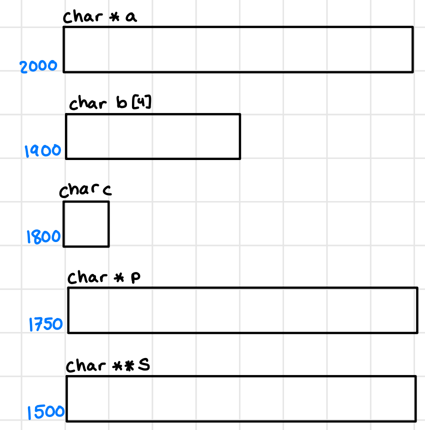
$\boxed{'0'}\boxed{'\backslash 0'}$ $\boxed{'1'}\boxed{'\backslash 0'}$

(1.9) *"0" <= *("1" + 1) '0' <= '\0', 48 <= 0 $\xrightarrow{\text{false}}$ $\boxed{0}$

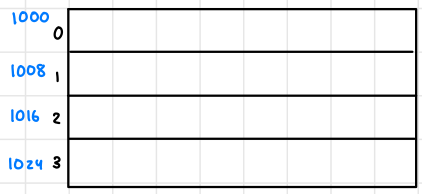
(1.10) s[1][0] <= s[0][1] '0' <= 'x' 48 <= 120 $\xrightarrow{\text{true}}$ $\boxed{1}$

(1.11) s[2][3] $\boxed{\text{unpredictable}}$, because we don't know what's at memory address 1803

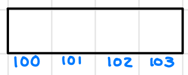
STACK



HEAP



PROGRAM CODE



Evaluate the expressions listed on the answer sheet in the context of the given C code. Here is what I mean by evaluating an expression:

- For integer expressions (i.e., expressions whose types are char, short, int, size_t, long, or long long--either signed or unsigned), write the numeric value in DECIMAL notation.
    - We will only accept decimal notation; e.g., if the answer is 65, NONE of the following will be accepted: 0x41, 01000001, 'A', 2^6+1.
    - C has no boolean type. Do NOT write "true" or "false". YOU WILL LOSE POINTS IF YOU DO. Write 1 for true and 0 for false.
    - Write "UNPREDICTABLE" for an integer expression whose value can change from one run of the program to another.
- For non-integer expressions, write the type name, in the format that you use to declare a variable of that type. Some example type names include (but not limited to):

      int *   double   double **   int (*)(int *, int *)

- Write "INVALID" if a given expression will result in a compiler error.

```
int main(int argc, char **argv)
{
    assert('0' == 48 && (long) NULL == 0);
    assert('a' == 97 && 'b' == 98 && 'c' == 99 && 'x' == 120);

    char *a = "0xa";
    char b[4] = "0xb";
    char c = 0xc;

    char *p = malloc(sizeof(char *) * 4);
    char **s = (char **)p;

    s[0] = a++;
    s[1] = b;
    s[2] = &c;
    s[3] = p;

    /////////////////////////////////////////////////////////////////
    //                                                             //
    //   Evaluate the expressions listed on the answer sheet      //
    //   in the context of main() at this point.                  //
    //                                                             //
    /////////////////////////////////////////////////////////////////

    free(s);

    return 0;
}
```

```
[1]

(1.1)    *a
--------------------------------------------------------
(1.2)    b + 1
--------------------------------------------------------
(1.3)    c + 2
--------------------------------------------------------
(1.4)    (long) argv[argc]
--------------------------------------------------------

(1.5)    sizeof(b)
--------------------------------------------------------
(1.6)    strlen(b + 1)
--------------------------------------------------------
(1.7)    sizeof(a) == sizeof(b + 2)
--------------------------------------------------------


(1.9)    *"0" <= *("1" + 1)
--------------------------------------------------------
(1.10)   s[1][0] <= s[0][1]
--------------------------------------------------------
(1.11)   s[2][3]
--------------------------------------------------------
```