Programmer som Data - Assignment 9

Bastjan Rosgaard Sejberg, Søren Kastrup, Weihao Chen Nyholm-Andersen

November 2023

10.1

(A)

- **ADD**
  Pops the two top values in stack, adds them together, finally pushing the result back onto the stack. To do this, we untag the two values, then add them and tag them again, thus making the stack 1 smaller, as one of the values gets "absorbed" during the process.

- **CSTI**
  introduces a new integer value/constant, increasing the size of the stack, incrementing the stack pointer and tagging the new value, then put it at the top of the stack.

- **NIL**
  A nil reference could possibly be used for something like an empty list or just a null pointer or akin. NIL and CSTI 0's critical difference is their interpretation: NIL is not tagged, thus will not be accounted for in regards to garbage collection. Conversely, CSTI 0 is indeed tagged and will therefore be accounted for in garbage collection and susceptible to a clean-up.

- **IFZERO**
  A conditional jump instruction, as seen in Assembly: It pops the top value on the stack, checks if it's an integer of 0 or just NIL, then (if true) jumps to the specified target address or will just go to the next instruction.

- **CONS**
  Holds two values or two pointers to values, making memory objects, also known as a pair or a cons cell. What isn't explicitly stated in the code is that the first value is CAR and second is CDR.

- **CAR**
  As aforementioned in CONS, CAR is the first element/value in a cons cell. The abbreviation means Contents of the Address part of Register. To retrieve this value, we check if the value at the top of the stack is a valid cons cell, then replaces the top of the stack with the CAR of the cons cell. If it isn't valid (like being a nil reference), it will in this case print "Cannot take car of null" and return -1.

- **SETCAR**
  Updates the first value (CAR) of a cons cell by popping the new element we want to replace CAR with, then update the cons cell at the top of the stack.

(B)

- *Color:*

  Color is defined as ((hdr)&3). It is a bitwise operation using the *AND* operator. The "3" is translated into binary, thus being "11", which is the first 2 values of the header, being "gg", which is the color. Therefore, it reads the current color. To explain concisely, the bitwise "AND" operator compares and sees what 1's they have gotten in common, then returns the result.

- *Length:*

  Length is defined as (((hdr)¿¿2)&0x003FFFFF). It is a bitwise operation, wherein it shifts the bits in the header to the right, thus ignoring the color bits. The *0x003FFFF* is a hexadecimal. Translated to binary, it is 22 digits of 1's, reading the length bits, then returns the length.

- *Paint:*

  Paint is defined as (((hdr)&($\tilde{3}$))—(color)). It takes the garbage collection bits and negates them, thus removing the color it has been tagged as. Afterwards, it executes the bitwise OR operator with the color. Since the values now are negated, they would be "0", thus it just replaces the bits with the color instead.

(C)

Allocate is exclusively being called in CONS, as it allocates the cons cell. There is no other interaction.

(D)

In allocate: If there is nothing left in the freelist, it performs collect to clean up and free up space

```
1
2         ...
3             if (attempt==1)
4           collect(s, sp);
5           } while (attempt++ == 1);
6           printf("Out of memory\n");
7           exit(1);
8         ...
```

Listing 1: listmachine.c

10.2

Code

```
// mark - Gives a block on the heap a different colour
void mark(word *block, int color) {
  block[0] = Paint(block[0], color);
}

// mark phase - Marks blocks on the heap reachable from the stack black.
void markPhase(int s[], int sp) {
  for (int i = 0; i < sp; i++)
    if (!IsInt(s[i]))
      mark((word *) s[i], Black);
}

// sweep phase - Scans entire heap, paints black blocks white and white blocks blue
//               and then creates a linked list of all blue blocks with a reference
//               to the start of the linked list to freelist variable.
void sweepPhase() {
  word *first, *next, *prev = 0;

  // Loops through the entire heap
  for (word* heapPtr = heap; heapPtr < afterHeap; heapPtr = next) {
    next = heapPtr + Length(heapPtr[0]) + 1; // reference to the next block

    // If this block is black, paint it white and continue
    if (Color(heapPtr[0]) == Black)
      mark(heapPtr, White);

    // If this block is white, paint it blue and link it together with the previous
    else if (Color(heapPtr[0]) == White){
      mark(heapPtr, Blue);
      heapPtr[1] = 0; // reference to the next block, default 0 as existence is yet unknown

      // If the current block is not first in the linked list,
      // provide the previous block a reference to this one
      if (prev != 0) {
        prev[1] = (word) heapPtr;
        prev = heapPtr;
      } else // First block in the linked list
        prev = first = heapPtr;
    }
  }

  // Provide the freelist variable a reference to the first free block in the linked list
  freelist = first;
}

void collect(int s[], int sp) {
  markPhase(s, sp);
  heapStatistics();
  sweepPhase();
  heapStatistics();
}
```

Listing 2: listmachine.c

Results

```
[soer4769@soerthinkpad Fun]$ ./listmachine ex30.out 1000
1000 999 998 997 996 995 994 993 992 991 990 989 988 987 986 985 984 983 982 981 980 979 978 977 976 975 974 973 972
 971 970 969 968 967 966 965 964 963 962 961 960 959 958 957 956 955 954 953 952 951 950 949 948 947 946 945 944 943
 942 941 940 939 938 937 936 935 934 933 932 931 930 929 928 927 926 925 924 923 922 921 920 919 918 917 916 915 914
 913 912 911 910 909 908 907 906 905 904 903 902 901 900 899 898 897 896 895 894 893 892 891 890 889 888 887 886 885
 884 883 882 881 880 879 878 877 876 875 874 873 872 871 870 869 868 867 866 865 864 863 862 861 860 859 858 857 856
 855 854 853 852 851 850 849 848 847 846 845 844 843 842 841 840 839 838 837 836 835 834 833 832 831 830 829 828 827
 826 825 824 823 822 821 820 819 818 817 816 815 814 813 812 811 810 809 808 807 806 805 804 803 802 801 800 799 798
 797 796 795 794 793 792 791 790 789 788 787 786 785 784 783 782 781 780 779 778 777 776 775 774 773 772 771 770 769
 768 767 766 765 764 763 762 761 760 759 758 757 756 755 754 753 752 751 750 749 748 747 746 745 744 743 742 741 740
 739 738 737 736 735 734 733 732 731 730 729 728 727 726 725 724 723 722 721 720 719 718 717 716 715 714 713 712 711
 710 709 708 707 706 705 704 703 702 701 700 699 698 697 696 695 694 693 692 691 690 689 688 687 686 685 684 683 682
 681 680 679 678 677 676 675 674 673 672 671 670 669 668 Heap: 333 blocks (666 words); of which 0 free (0 words, lar
gest 0 words); 1 orphans
Heap: 333 blocks (666 words); of which 332 free (664 words, largest 2 words); 1 orphans
667 666 665 664 663 662 661 660 659 658 657 656 655 654 653 652 651 650 649 648 647 646 645 644 643 642 641 640 639
638 637 636 635 634 633 632 631 630 629 628 627 626 625 624 623 622 621 620 619 618 617 616 615 614 613 612 611 610
609 608 607 606 605 604 603 602 601 600 599 598 597 596 595 594 593 592 591 590 589 588 587 586 585 584 583 582 581
580 579 578 577 576 575 574 573 572 571 570 569 568 567 566 565 564 563 562 561 560 559 558 557 556 555 554 553 552
551 550 549 548 547 546 545 544 543 542 541 540 539 538 537 536 535 534 533 532 531 530 529 528 527 526 525 524 523
522 521 520 519 518 517 516 515 514 513 512 511 510 509 508 507 506 505 504 503 502 501 500 499 498 497 496 495 494
493 492 491 490 489 488 487 486 485 484 483 482 481 480 479 478 477 476 475 474 473 472 471 470 469 468 467 466 465
464 463 462 461 460 459 458 457 456 455 454 453 452 451 450 449 448 447 446 445 444 443 442 441 440 439 438 437 436
435 434 433 432 431 430 429 428 427 426 425 424 423 422 421 420 419 418 417 416 415 414 413 412 411 410 409 408 407
406 405 404 403 402 401 400 399 398 397 396 395 394 393 392 391 390 389 388 387 386 385 384 383 382 381 380 379 378
377 376 375 374 373 372 371 370 369 368 367 366 365 364 363 362 361 360 359 358 357 356 355 354 353 352 351 350 349
348 347 346 345 344 343 342 341 340 339 338 337 336 Heap: 333 blocks (666 words); of which 0 free (0 words, largest
0 words); 1 orphans
Heap: 333 blocks (666 words); of which 332 free (664 words, largest 2 words); 1 orphans
335 334 333 332 331 330 329 328 327 326 325 324 323 322 321 320 319 318 317 316 315 314 313 312 311 310 309 308 307
306 305 304 303 302 301 300 299 298 297 296 295 294 293 292 291 290 289 288 287 286 285 284 283 282 281 280 279 278
277 276 275 274 273 272 271 270 269 268 267 266 265 264 263 262 261 260 259 258 257 256 255 254 253 252 251 250 249
248 247 246 245 244 243 242 241 240 239 238 237 236 235 234 233 232 231 230 229 228 227 226 225 224 223 222 221 220
219 218 217 216 215 214 213 212 211 210 209 208 207 206 205 204 203 202 201 200 199 198 197 196 195 194 193 192 191
190 189 188 187 186 185 184 183 182 181 180 179 178 177 176 175 174 173 172 171 170 169 168 167 166 165 164 163 162
161 160 159 158 157 156 155 154 153 152 151 150 149 148 147 146 145 144 143 142 141 140 139 138 137 136 135 134 133
132 131 130 129 128 127 126 125 124 123 122 121 120 119 118 117 116 115 114 113 112 111 110 109 108 107 106 105 104
103 102 101 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 6
6 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28
27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 Heap: 333 blocks (666 words); of which 0 free (0 w
ords, largest 0 words); 1 orphans
Heap: 333 blocks (666 words); of which 332 free (664 words, largest 2 words); 1 orphans
3 2 1
Used   0.000 cpu seconds
[soer4769@soerthinkpad Fun]$ ./listmachine ex35.out 5
33 33 Heap: 333 blocks (666 words); of which 0 free (0 words, largest 0 words); 1 orphans
Heap: 333 blocks (666 words); of which 330 free (660 words, largest 2 words); 1 orphans
44 44
Used   0.000 cpu seconds
[soer4769@soerthinkpad Fun]$ ./listmachine ex36.out 5
1 Heap: 333 blocks (666 words); of which 0 free (0 words, largest 0 words); 1 orphans
Heap: 333 blocks (666 words); of which 331 free (662 words, largest 2 words); 1 orphans
1
Used   0.000 cpu seconds
```

Refer to the file listmachine.c in the folder ***Exercise_10.2****.*

10.3

Code

```
// mark - Gives a block on the heap a different colour
void mark(word *block, int color) {
  block[0] = Paint(block[0], color);
}

// mark phase - Marks blocks on the heap reachable from the stack black.
void markPhase(int s[], int sp) {
  for (int i = 0; i < sp; i++)
    if (!IsInt(s[i]))
      mark((word *) s[i], Black);
}

// sweep phase - Scans entire heap, paints black blocks white and white blocks blue
//               and then creates a linked list of all blue blocks with a reference
//               to the start of the linked list to freelist variable.
//               This method also joins together adjecent dead blocks into a single
//               dead block.
void sweepPhase() {
  word *first, *next, *prev = 0;

  // Loops through the entire heap
  for (word* heapPtr = heap; heapPtr < afterHeap; heapPtr = next) {
    next = heapPtr + Length(heapPtr[0]) + 1; // reference to the next block

    // If this block is black, paint it white and continue
    if (Color(heapPtr[0]) == Black)
      mark(heapPtr, White);

    // If this block is white, paint it blue and join it with adjecent blocks
    else if (Color(heapPtr[0]) == White){
      mark(heapPtr, Blue);
      heapPtr[1] = 0; // reference to the next block, default 0 as existence is yet unknown

      // Gets a reference to the next reachable adjecent white block on the heap
      // when one the is found, join them together an continue to the next one
      while (next < afterHeap && Color(next[0]) == White) {
        heapPtr[0] = mkheader(0, Length(heapPtr[0]) + Length(next[0]) + 1, Blue);
        next += Length(next[0]) + 1;
      }

      // If the current block is not first in the linked list,
      // provide the previous block a reference to this one
      if (prev != 0) {
        prev[1] = (word) heapPtr;
        prev = heapPtr;
      } else // First block in the linked list
        prev = first = heapPtr;
    }
  }

  // Provide the freelist variable a reference to the first free block in the linked list
  freelist = first;
}
```

Listing 3: listmachine.c

Results

```
[soer4769@soerthinkpad Fun]$ ./listmachine ex30.out 1000
1000 999 998 997 996 995 994 993 992 991 990 989 988 987 986 985 984 983 982 981 980 979 978 977 976 975 974 973 972
 971 970 969 968 967 966 965 964 963 962 961 960 959 958 957 956 955 954 953 952 951 950 949 948 947 946 945 944 943
 942 941 940 939 938 937 936 935 934 933 932 931 930 929 928 927 926 925 924 923 922 921 920 919 918 917 916 915 914
 913 912 911 910 909 908 907 906 905 904 903 902 901 900 899 898 897 896 895 894 893 892 891 890 889 888 887 886 885
 884 883 882 881 880 879 878 877 876 875 874 873 872 871 870 869 868 867 866 865 864 863 862 861 860 859 858 857 856
 855 854 853 852 851 850 849 848 847 846 845 844 843 842 841 840 839 838 837 836 835 834 833 832 831 830 829 828 827
 826 825 824 823 822 821 820 819 818 817 816 815 814 813 812 811 810 809 808 807 806 805 804 803 802 801 800 799 798
 797 796 795 794 793 792 791 790 789 788 787 786 785 784 783 782 781 780 779 778 777 776 775 774 773 772 771 770 769
 768 767 766 765 764 763 762 761 760 759 758 757 756 755 754 753 752 751 750 749 748 747 746 745 744 743 742 741 740
 739 738 737 736 735 734 733 732 731 730 729 728 727 726 725 724 723 722 721 720 719 718 717 716 715 714 713 712 711
 710 709 708 707 706 705 704 703 702 701 700 699 698 697 696 695 694 693 692 691 690 689 688 687 686 685 684 683 682
 681 680 679 678 677 676 675 674 673 672 671 670 669 668 Heap: 333 blocks (666 words); of which 0 free (0 words, lar
gest 0 words); 1 orphans
Heap: 2 blocks (997 words); of which 1 free (995 words, largest 995 words); 1 orphans
667 666 665 664 663 662 661 660 659 658 657 656 655 654 653 652 651 650 649 648 647 646 645 644 643 642 641 640 639
638 637 636 635 634 633 632 631 630 629 628 627 626 625 624 623 622 621 620 619 618 617 616 615 614 613 612 611 610
609 608 607 606 605 604 603 602 601 600 599 598 597 596 595 594 593 592 591 590 589 588 587 586 585 584 583 582 581
580 579 578 577 576 575 574 573 572 571 570 569 568 567 566 565 564 563 562 561 560 559 558 557 556 555 554 553 552
551 550 549 548 547 546 545 544 543 542 541 540 539 538 537 536 535 534 533 532 531 530 529 528 527 526 525 524 523
522 521 520 519 518 517 516 515 514 513 512 511 510 509 508 507 506 505 504 503 502 501 500 499 498 497 496 495 494
493 492 491 490 489 488 487 486 485 484 483 482 481 480 479 478 477 476 475 474 473 472 471 470 469 468 467 466 465
464 463 462 461 460 459 458 457 456 455 454 453 452 451 450 449 448 447 446 445 444 443 442 441 440 439 438 437 436
435 434 433 432 431 430 429 428 427 426 425 424 423 422 421 420 419 418 417 416 415 414 413 412 411 410 409 408 407
406 405 404 403 402 401 400 399 398 397 396 395 394 393 392 391 390 389 388 387 386 385 384 383 382 381 380 379 378
377 376 375 374 373 372 371 370 369 368 367 366 365 364 363 362 361 360 359 358 357 356 355 354 353 352 351 350 349
348 347 346 345 344 343 342 341 340 339 338 337 336 Heap: 333 blocks (666 words); of which 0 free (0 words, largest
0 words); 1 orphans
Heap: 3 blocks (996 words); of which 2 free (994 words, largest 992 words); 1 orphans
335 334 333 332 331 330 329 328 327 326 325 324 323 322 321 320 319 318 317 316 315 314 313 312 311 310 309 308 307
306 305 304 303 302 301 300 299 298 297 296 295 294 293 292 291 290 289 288 287 286 285 284 283 282 281 280 279 278
277 276 275 274 273 272 271 270 269 268 267 266 265 264 263 262 261 260 259 258 257 256 255 254 253 252 251 250 249
248 247 246 245 244 243 242 241 240 239 238 237 236 235 234 233 232 231 230 229 228 227 226 225 224 223 222 221 220
219 218 217 216 215 214 213 212 211 210 209 208 207 206 205 204 203 202 201 200 199 198 197 196 195 194 193 192 191
190 189 188 187 186 185 184 183 182 181 180 179 178 177 176 175 174 173 172 171 170 169 168 167 166 165 164 163 162
161 160 159 158 157 156 155 154 153 152 151 150 149 148 147 146 145 144 143 142 141 140 139 138 137 136 135 134 133
132 131 130 129 128 127 126 125 124 123 122 121 120 119 118 117 116 115 114 113 112 111 110 109 108 107 106 105 104
103 102 101 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 6
6 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28
27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 Heap: 333 blocks (666 words); of which 0 free (0 w
ords, largest 0 words); 1 orphans
Heap: 2 blocks (997 words); of which 1 free (995 words, largest 995 words); 1 orphans
3 2 1
Used   0.001 cpu seconds
[soer4769@soerthinkpad Fun]$ ./listmachine ex35.out 5
33 33 Heap: 333 blocks (666 words); of which 0 free (0 words, largest 0 words); 1 orphans
Heap: 4 blocks (995 words); of which 1 free (989 words, largest 989 words); 1 orphans
44 44
Used   0.000 cpu seconds
[soer4769@soerthinkpad Fun]$ ./listmachine ex36.out 5
1 Heap: 333 blocks (666 words); of which 0 free (0 words, largest 0 words); 1 orphans
Heap: 3 blocks (996 words); of which 1 free (992 words, largest 992 words); 1 orphans
1
Used   0.000 cpu seconds
```

Refer to the file listmachine.c in the folder ***Exercise_10.3****.*