|  |  |
| --- | --- |
| **.data 0x10000000**  key: .asciiz "Enter the key: "  value: .asciiz "Enter the value: "  endline: .asciiz "\n"  node: .asciiz "Node: "  keyvalue: .asciiz " Key value: "  char: .asciiz " Char: "  nextnode: .asciiz " Next node: "  start: .asciiz "Start"  endstring: .asciiz "End"  **.text**  **main:**  addi $sp, $sp, -12  add $s0, $zero, $zero  lui $s1, 0x1001  add $s2, $zero, $zero  j mainmethod  li $v0, 10  addi $sp, $sp, 12  syscall  **mainmethod:**  add $t0, $zero, $zero  add $t2, $zero, $zero  li $v0, 4  la $a0, key  syscall  li $v0, 5  syscall  add $t4, $zero, $v0  li $v0, 4  la $a0, endline  syscall  addi $t5, $zero, -1  **mainwhile:**  beq $t4, $t5, aftermainwhile  addi $sp, $sp, -16  sw $ra, 12($sp)  sw $t0, 8($sp)  sw $t2, 4($sp)  sw $t4, 0($sp)  addiu $a0, $zero, 108  jal mymalloc  add $t1, $zero, $v0  lw $t4, 0($sp)  lw $t2, 4($sp)  lw $t0, 8($sp)  lw $ra, 12($sp)  addi $sp, $sp, 16  bne $t0, $zero, next  add $t0, $zero, $t1  **next:**  sw $t4, 0($t1)  li $v0, 4  la $a0, value  syscall  li $v0, 8  addiu $t8, $t1, 4  addu $a0, $zero, $t8  li $a1, 100  syscall  li $v0, 4  la $a0, endline  syscall  sw $t2, 104($t1)  li $v0, 4  la $a0, key  syscall  li $v0, 5  syscall  add $t4, $zero, $v0  li $v0, 4  la $a0, endline  syscall  add $t2, $zero, $t1  addi $t5, $zero, -1  j mainwhile  **aftermainwhile:**  sw $zero, 104($t1)  add $a0, $zero, $t0  addi $sp, $sp, -4  sw $ra, 0($sp)  jal mergesort  add $a0, $zero, $v0  jal printout  lw $ra, 0($sp)  addi $sp, $sp, 4  jr $ra  **mergesort:**  bne $a0, $zero, secondcheck  add $v0, $zero, $a0  jr $ra  **secondcheck:**  lw $t0, 104($a0)  bne $t0, $zero, mergesortini  add $v0, $zero, $a0  jr $ra  **mergesortini:**  add $t1, $zero, $a0  add $t2, $zero, $t0  **mergesortwhile:**  beq $t2, $zero, aftermergesortwhile  lw $t0, 104($t2)  beq $t0, $zero, aftermergesortwhile  lw $a0, 104($a0)  lw $t0, 104($a0)  lw $t2, 104($t0)  j mergesortwhile  **aftermergesortwhile:**  lw $t2, 104($a0)  sw $zero, 104($a0)  addi $sp, $sp, -8  sw $ra, 4($sp)  sw $t2, 0($sp)  add $a0, $zero, $t1  jal mergesort  lw $a0, 0($sp)  sw $v0, 0($sp)  jal mergesort  lw $a0, 0($sp)  addi $sp, $sp, 4  add $a1, $zero, $v0  jal merge  lw $ra, 0($sp)  addi $sp, $sp, 4  jr $ra  **merge:**  bne $a0, $zero, secondifcheck  add $v0, $zero, $a0  jr $ra  secondifcheck:  bne $a1, $zero, check  add $v0, $zero, $a1  jr $ra  **check:**  lw $t1, 0($a0)  lw $t2, 0($a1)  slt $t0, $t1, $t2  beq $t0, $zero, elsecheck  add $t3, $zero, $a0  addi $sp, $sp, -8  sw $ra, 4($sp)  sw $t3, 0($sp)  lw $a0, 104($a0)  jal merge  lw $t3, 0($sp)  lw $ra, 4($sp)  addi $sp, $sp, 8  sw $v0, 104($t3)  j end  **elsecheck:**  add $t3, $zero, $a1  addi $sp, $sp, -8  sw $ra, 4($sp)  sw $t3, 0($sp)  lw $a1, 104($a1)  jal merge  lw $t3, 0($sp)  lw $ra, 4($sp)  addi $sp, $sp, 8  sw $v0, 104($t3)  **end:**  add $v0, $zero, $t3  jr $ra  **printout:**  add $t0, $zero, $a0  li $v0, 4  la $a0, start  syscall  li $v0, 4  la $a0, endline  syscall  **printwhile:**  beq $t0, $zero, afterprintwhile  li $v0, 4  la $a0, node  syscall  li $v0, 1  add $a0, $zero, $t0  syscall  li $v0, 4  la $a0, keyvalue  syscall  li $v0, 1  lw $a0, 0($t0)  syscall  li $v0, 4  la $a0, char  syscall  li $v0, 11  lw $a0, 4($t0)  syscall  li $v0, 4  la $a0, nextnode  syscall  li $v0, 1  lw $a0, 104($t0)  syscall  li $v0, 4  la $a0, endline  syscall  lw $t0, 104($t0)  j printwhile  **afterprintwhile:**  li $v0, 4  la $a0, endstring  syscall  jr $ra  **mymalloc:**  beq $s0, $zero, else  add $t0, $zero, $s0  lw $s0, 104($s0)  j endmalloc  **else:**  add $t0, $s1, $s2  sw $zero, 104($t0)  addi $s2, $s2, 108  **endmalloc:**  add $v0, $zero, $t0  jr $ra  **free:**  sw $s0, 4($a0)  add $s0, $zero, $a0  jr $ra | struct node {  int key;  char value[100];  struct node\* next;  };  int main(){  struct node\* list =0  struct node\* current;  struct node\* prev =0;  int key;  askuserforinput(key);  while(key!=-1){  current = (struct node\*) mymalloc(108);  if(**!list**){  list = current;  }  current->key=key;  askuserforinput(current->value);  current->next=prev;  askuserforinput(key);  prev=current;  }  current->next=0;  **list =** mergesort(list);  printout(list);  }  struct node \*mergesort(struct node\* head){  struct node\* head\_one;  struct node\* head\_two;  if((head == NULL) || (head->next == NULL)){  return head;  }  head\_one = head;  head\_two = head->next;  while((head\_two != NULL) && (head\_two->NEXT == NULL)){  head = head->next;  head\_two = head->next->next;  }  head\_two =head->next;  head->next=NULL;  return merge(mergesort(head\_one), mergesort(head\_two));  }  struct node\* merge(struct node\* head\_one, struct node\* head\_two){  struct node\*=head\_three;  if(head\_one == NULL){  return head\_two;  }  if(head\_two == NULL){  return head\_one;  }  if(head\_one->key < head\_two->key){  head\_three=head\_one;  head\_three->next=merge(head\_one->next, head\_two);  }  else{  head\_three=head\_two;  head\_three->next=merge(head\_one, head\_two->next);  }  return head\_three;  } |
| ***#*** *$s0 bevat free\_list*  ***#*** *$s1 bevat heap*  ***#*** *$s2 bevat offset\_heap*  **allocate**:  beq $s0, $zero, else  add $t0, $zero, $s0  lw $s0, 4($s0)  j end  **else**:  add $t0, $s1, $s2  sw $zero, 4($t0)  addi $s2, $s2, 8  **end**:  add $v0, $zero, $t0  jr $ra  **free**:  sw $s0, 4($a0)  add $s0, $zero, $a0  jr $ra | struct chunk {  int value;  struct chunk\* next\_chunk;  };  struct chunk\* free\_list = 0;  struct chunk\* heap;  unsigned int offset\_heap;  struct chunk \* allocate(){  struct chunk\* c;  if(free\_list != 0) {  c=free\_list;  free\_list=free\_list->next\_chunk;  }  else {  c=heap+offset\_heap;  offset\_heap+=sizeof(struct chunk);  }  return c;  }  void free(struct chunk\* c){  c->next\_chunk=free\_list;  free\_list = c;  } |
| **.data 0x10000000**  this: .asciiz “This=˽ “  size: .asciiz “˽size=˽ “  next: .asciiz “˽next˽chunk=˽”  endline: .asciiz “\n”  end: .asciiz “end”  **.globl main**  **.text**  **main**:  lui $t0, 0x1001  li $t1, 10  addi $t2, $t0, 20  sw $t1, 0($t0)  sw $t2, 4($t0)  li $t1, 6  add $t2, $t0, 36  sw $t1, 20($t0)  sw $t2, 24($t0)  move $a0, $t0  jal printlist  li $v0, 10  syscall  **printlist**:  add $t0, $zero, $a0  **printwhile**:  beq $t0, $zero, afterprintwhile  li $v0, 4  la $a0, this  syscall  li $v0, 1  add $a0, $zero, $t0  syscall  li $v0, 4  la $a0, size  syscall  li $v0, 1  lw $a0, 0($t0)  syscall  li $v0, 4  la $a0, next  syscall  li $v0, 1  lw $a0, 4($t0)  syscall  li $v0, 4  la $a0, endline  syscall  lw $t0, 4($t0)  j printwhile  **afterprintwhile:**  li $v0, 4  la $a0, end  syscall  li $v0, 4  la $a0, endline  syscall  jr $ra | struct malloc\_chunk{  unsigned int size;  struct malloc\_chunk\* next\_chunk;  };  void print\_list(struct malloc\_chunk\* list){  while(list!=0){  printf(“This=%p, ˽size=%d, ˽next˽chunk˽=%p˽->”,list, list->size, list->next\_chunk);  }  printf(“end”);  } |
| ***#*** *$s0 bevat free\_list*  ***#*** *$s1 bevat heap*  ***#*** *$s2 bevat top of heap*  **mymalloc**:  addi $sp, $sp, -12  sw $s0, 8($sp)  sw $s1, 4($sp)  sw $s2, 0($sp)  lui $t5, 0x1000  lw $s0, 0($t5)  lw $s1, 4($t5)  lw $s2, 8($t5)  addiu $t4, $a0, 11  lui $t5, 0xffff  ori $t5, $t5, 0xfffc  and $t3, $t4, $t5  add $t1, $zero, $s0  add $t2, $zero, $s0  **mallocwhile**:  beq $t1, $zero, aftermallocwhile  lw $t4, 0($t1)  slt $t5, $t4, $t3  beq $t5, $zero, aftermallocwhile  add $t2, $zero, $t1  lw $t1, 4($t1)  j mallocwhile  **aftermallocwhile**:  beq $t1, $zero, mallocelse  lw $t4, 4($t1)  sw $t4, 4($t2)  lw $t5, 0($t1)  ori $t4, $t5, 1  sw $t4, 0($t1)  addiu $v0, $t1, 8  j endmalloc  **mallocelse**:  add $t0, $zero, $s2  ori $t4, $t3, 1  sw $t4, 0($t0)  add $s1, $s1, $t3  addiu $v0, $t0, 8  **endmalloc:**  lui $t5, 0x1000  sw $s0, 0($t5)  sw $s1, 4($t5)  sw $s2, 8($t5)  lw $s2, 0($sp)  lw $s1, 4($sp)  lw $s0, 8($sp)  addi $sp, $sp, 12  jr $ra  **myfree**:  addi $sp, $sp, -4  sw $s0, 0($sp)  lui $t5, 0x1001  lw $s0, 0($t5)  addiu $t0, $a0, -8  lw $t4, 0($t0)  andi $t5, $t4, 1  beq $t5, $zero, endfree  lui $t5, 0xffff  or $t5, $t5, 0xfffc  and $t4, $t4, $t5  add $t1, $zero, $s0  add $t2, $zero, $zero  **freewhile**:  beq $t1, $zero, afterfreewhile  lw $t4, 0($t1)  slt $t5, $t4, $t3  beq $t5, $zero, afterfreewhile  add $t2, $zero, $t1  lw $t1, 4($t1)  j freewhile  **afterfreewhile**:  bne $t2, $zero, endif  sw $t1, 4($t0)  add $s0, $zero, $t0  j endfree  **endif:**  lw $t4, 4($t3)  sw $t4, 4($t0)  sw $t0, 4($t3)  jr $ra  **endfree:**  lui $t5, 0x1001  sw $s0, 0($0)  lw $s0, 0($sp)  addi $sp, $sp, 4  jr $ra | #include <stdio.h>  #include <unistd.h>  struct malloc\_chunk{  unsigned int size;  struct malloc\_chunk \* nextfree;  };  static struct malloc\_chunk \* freelist=0;  static unsigned in top\_heap = 0x10010000;  void \*mymalloc(unsigned int size){  struct malloc\_chunk\* toreturn;  struct malloc\_chunk\* search;  struct malloc\_chunk\* prevsearch;  unsigned int realsize;  realsize = (size+sizeof(struct malloc\_chunk)+3)&0xfffffffc  search = prevsearch = freelist;  while(search && search->size < realsize){  prevsearch = search;  search = search->nextfree;  }  if(search){  prevsearch->nextfree=search->nextfree;  search->size |=1;  return ((void \*) search)+sizeof(struct malloc\_chunk);  }  else{  toreturn=(struct malloc\_chunk \*) top\_heap;  toreturn->size = realsize | 1;  top\_heap+=realsize;  return toreturn +sizeof(struct malloc\_chunk);  }  }  void myfree( void \*tofree){  struct malloc\_chunk\* freechunk;  struct malloc\_chunk\* search;  struct malloc\_chunk\* prevsearch;  unsigned int realsize;  freechunk=(struct malloc\_chunk \*)(tofree-sizeof(struct malloc\_chunk));  if(freechunk->size &1){  realsize = freechunk->size & 0xfffffffc;  search freelist;  prevsearch=0;  while(search&&search->size<realsize){  prevsearch=search;  search=search->nextfree;  }  if(!prevsearch){  freechunk->nextfree=search;  freelist=freechunk;  return;  }  freechunk->nextfree=prevsearch->nextfree  prevsearch->nextfree=freechunk;  return;  }  } |