Practical no. 2:

Design and Implement single pass macro processor to handle recursive and nested calls

|  |  |
| --- | --- |
| Program: | for args in macro\_args: |
| inp\_file = open('src\_code.txt') |  |
|  |  |
| macro\_start\_flag = 0 |  |
| macro\_name\_flag = 0 |  |
| MDT = dict() |  |
| MNT = dict() |  |
| ALA = dict() |  |
| macro\_name = '' | macro\_ala = [] |
| mdt\_index = 0 | macro\_call = line.split()[0] |
| mnt\_index = 0 | if macro\_call in list(MDT.keys()): |
| ala\_index = 0 | call\_count += 1 |
| macro\_def = [] | arguments = line.split()[1:] |
| macro\_args = [] | for arg in arguments: |
| call\_count = 0 | arg = arg.upper() |
|  | if len(arg) < 8: |
| # reading through input and creating MDT table | additional\_b = 8 - len(arg) |
| for line in inp\_file: | for b in range(additional\_b): |
| if len(line.split()) > 0: | arg += 'b' |
| line = line.replace('\n', '') | macro\_ala.append(arg) |
| if line.split()[0] == 'MACRO': | ala\_entry = { |
| macro\_name\_flag = 0 | macro\_call+'\_'+str(call\_count): |
| macro\_start\_flag = 1 | macro\_ala |
| pass | } |
| elif line.split()[0] == 'MEND': | ALA.update(ala\_entry) |
| mdt\_index += 1 |  |
| macro\_line = (mdt\_index, line) | # creating MNT table |
| macro\_def.append(macro\_line) | for entry in MDT: |
| mdt\_entry = { | mnt\_index += 1 |
| macro\_name: macro\_def | mnt\_entry = { |
| } | entry: (mnt\_index, MDT[entry][0][0]) |
| MDT.update(mdt\_entry) | } |
| macro\_def = [] | MNT.update(mnt\_entry) |
| macro\_args = [] |  |
| macro\_start\_flag = 0 | # adding values in ALA |
| else: | ALA\_final = ALA.copy() |
| if macro\_start\_flag != 0: | for calls in ALA: |
| if macro\_name\_flag == 0: | macro\_ala = [] |
| macro\_name = line.split()[0] | macro\_call = calls.split('\_')[0] |
| mdt\_ent = { | given = ALA[calls] |
| macro\_name: None | for line in MDT[macro\_call]: |
| } | if macro\_call not in line[1] and 'MEND' not in |
| MDT.update(mdt\_ent) | line[1]: |
| macro\_name\_flag = 1 | for code in line[1].split(): |
| mdt\_index += 1 | if '#' in code: |
| for code in line.split(): | index = int(code.replace('#', '')) - 1 |
| # sustituting '#indx' for arguments | macro\_ala.append(given[index]) |
| if '&' in code: | next\_call = line[1].split()[0] |
| if code not in [arg[0] for arg in | call\_count += 1 |
| macro\_args]: | next\_call += '\_' + str(call\_count) |
| arg\_sub = | ala\_entry = { |
| '#'+str(len(macro\_args)+1) | next\_call: macro\_ala |
| macro\_args.append((code, | } |
| arg\_sub)) | ALA\_final.update(ala\_entry) |
| else: |  |

|  |  |
| --- | --- |
| # printing output  print('\nMacro Definition Table (MDT)') print('Index\tContents')  for entry in MDT:  for lines in MDT[entry]: print(lines[0], '\t', lines[1])  print('\nMacro Name Table (MNT)') print('Index\tMacro Name\tMDT Index') | for entry in MNT:  print(MNT[entry][0], '\t', entry,'\t\t', MNT[entry][1])  print('\nArgument List Array (ALA)') print('Index\tArgument')  for entry in ALA\_final:  for arg in ALA\_final[entry]: ala\_index += 1 print(ala\_index, '\t', arg) |
| Input source Program: MACRO  ADD1 &arg L 1 &arg  A 1 =F'1'  ST 1 &arg MEND  MACRO  ADDS &arg1 &arg2 &arg3 ADD1 &arg1  ADD1 &arg2 ADD1 &arg3 MEND  ADDS data1 data2 data3 | |
| Output: | |