|  |  |
| --- | --- |
| **string.h**  Header file containing the different functions for C-style string manipulations.  Copy ‘sourceStr’ to ‘destStr’  strcpy(destStr, sourceStr)  Copy up to ‘n’ chars from ‘sourceStr’ to ‘destStr’  strncpy(destStr, sourceStr, n)  Concatenate (append) the string ‘sourceStr’ to ‘destStr’  strcat(destStr, sourceStr)  Concatenates ‘n’ chars from ‘sourceStr’ to ‘destStr’  strncat(destStr, sourceStr, n)  NULL if ‘c’ does not exist in ‘sourceStr’  strchr(sourceStr, c)  Length of the ‘sourceStr’ excluding ‘\0’  size\_t strlen(sourceStr)  Compares and returns 0 if ‘str1’ = ‘str2’  int strcmp(str1, str2)  Get  ‘n’ chars from stdin – including spaces, tabs  fgets(str, n, stdin) | /\* String functions \*/  char test1[50] = “C Programming”, test2[50];  // Try this  test2 = test1;  // Need to do the following to copy C strings  strcpy(test2, test1);  // test2 will be “C Programming”  // test1 = “C Programming”, test2 = “”  strncpy(test2, test1, 6);  // test2 will be “C Prog”;  // test2 = “C Programming”  strcat(test2, “ – CS 1713”);  // test2 will now be “C Programming – CS 1713”  // test2 = “C Programming”  strncat(test2, “ – CS 1713”, 5);  // test2 will now be “C Programming – CS”  // test1 = “C Programming”, test2 = “CS 1713”  printf(“%d”, (int)strlen(test1)); // 13  printf(“%d”, (int)strlen(test2)); // 7  // test1 = “C Programming”, test2 = “CS 1713”  if(strcmp(test2, “CS 1713”) == 0)  // True and hence will be executed  if(strcmp(test1, test2) == 0)  // Not equal, false and hence will not run  // String input from user - “C Programming”  char test[20];  printf(“Enter course name : ”);  scanf(“%s”, test);  printf(“%s”, test);  // Output  “Enter course name : C Programming  C”  char test[20];  printf(“Enter course name : ”);  fgets(test, 20, stdin);  printf(“%s”, test);  // Output  “Enter course name : C Programming  C Programming  ” |
| **ctype.h**  Header file containing the different functions for working with characters. | char myString[30] = "Hey9! Go"; |
| ***isalpha(c)*** -- Returns true if c is alphabetic: a-z or A-Z. | isalpha('A'); // Returns true  isalpha(myString[0]); // Returns true because 'H' is alphabetic  isalpha(myString[3]); // Returns false because '9' is not alphabetic |
| ***isdigit(c)*** -- Returns true if c is a numeric digit: 0-9. | isdigit(myString[3]); // Returns true because '9' is numeric  isdigit(myString[4]); // Returns false because ! is not numeric |
| ***isalnum(c)*** -- Returns true if c is alphabetic or a numeric digit. Thus, returns true if either isalpha or isdigit would return true. | isalnum('A'); // Returns true  isalnum(myString[3]); // Returns true because '9' is numeric |
| ***isspace(c)*** -- Returns true if character c is a whitespace. | isspace(myString[5]); // Returns true because that character is a space ' '.  isspace(myString[0]); // Returns false because 'H' is not whitespace. |
| ***islower(c)*** -- Returns true if character c is a lowercase letter a-z. | islower(myString[0]); // Returns false because 'H' is not lowercase.  islower(myString[1]); // Returns true because 'e' is lowercase.  islower(myString[3]); // Returns false because '9' is not a lowercase letter. |
| ***isupper(c)*** -- Returns true if character c is an uppercase letter A-Z. | isupper(myString[0]); // Returns true because 'H' is uppercase.  isupper(myString[1]); // Returns false because 'e' is not uppercase.  isupper(myString[3]); // Returns false because '9' is not an uppercase letter. |
| ***isblank(c)*** -- Returns true if character c is a blank character. Blank characters include spaces and tabs. | isblank(myString[5]); // Returns true because that character is a space ' '.  isblank(myString[0]); // Returns false because 'H' is not blank. |
| ***isxdigit(c)*** -- Returns true if c is a hexadecimal digit: 0-9, a-f, A-F. | isxdigit(myString[3]); // Returns true because '9' is a hexadecimal digit.  isxdigit(myString[1]); // Returns true because 'e' is a hexadecimal digit.  isxdigit(myString[6]); // Returns false because 'G' is not a hexadecimal digit. |
| ***ispunct(c)*** -- Returns true if c is a punctuation character. Punctuation characters include: !"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~ | ispunct(myString[4]); // Returns true because '!' is a punctuation character.  ispunct(myString[6]); // Returns false because 'G' is not a punctuation character. |
| ***isprint(c)*** -- Returns true if c is a printable character. Printable characters include alphanumeric, punctuation, and space characters. | isprint(myString[0]); // Returns true because 'H' is a alphabetic.  isprint(myString[4]); // Returns true because '!' is punctuation.  isprint(myString[5]); // Returns true because that character is a space ' '.  isprint('\0'); // Returns false because the null character is not printable. |
| ***iscntrl(c)*** -- Returns true if c is a control character. Control characters are all characters that are not printable. | iscntrl(myString[0]); // Returns false because 'H' is a not a control character  iscntrl(myString[5]); // Returns false because space is a not a control character  iscntrl('\0'); // Returns true because the null character is a control character |
| ***toupper(c)*** -- If c is a lowercase alphabetic character (a-z), returns the uppercase version (A-Z). If c is not a lowercase alphabetic character, just returns c. | toupper(myString[0]); // Returns 'H' (no change)  toupper(myString[1]); // Returns 'E' ('e' converted to 'E')  toupper(myString[3]); // Returns '9' (no change)  toupper(myString[5]); // Returns ' ' (no change) |
| ***tolower(c)*** -- If c is an uppercase alphabetic character (A-Z), returns the lowercase version (a-z). If c is not an uppercase alphabetic character, just returns c. | tolower(myString[0]); // Returns 'h' ('H' converted to 'h')  tolower(myString[1]); // Returns 'e' (no change)  tolower(myString[3]); // Returns '9' (no change)  tolower(myString[5]); // Returns ' ' (no change) |
| **Structs**  Merge data of same type - Array  Merge data of different types – Structs  typedef struct StructTypeName\_struct {  type item1;  type item2;  ...  type itemN;  } StructTypeName;  ...  StructTypeName myVar;    myVar.item1 = ...  OR  struct StructTypeName {  type item1;  type item2;  ...  type itemN;  }StructTypeName;  ...  StructTypeName myVar;    myVar.item1 = ... | /\* Student information – name, age, gpa \*/  int N = 5;  char name[5][10];  int age[5];  double gpa[5];  /\* Do the same as above using struct \*/  #include<stdio.h>  #include<string.h>  typedef struct Student\_struct  {  char name[10];  int age;  double gpa;  } Student;  int main(int argc, char \*argv[])  {  Student students[5];  int i=0, N=5;  for(i=0; i<N; i++)  {  printf(“Enter Student %d Information\n”,i+1);  printf(“Enter name : ”);  scanf(“%s”, students[i].name);  printf(“Enter age : ”);  scanf(“%d”, &students[i].age);  printf(“Enter GPA : ”);  scanf(“%lf”, &students[i].gpa);  }  for(i=0; i<N; i++)  {  printf(“Student %d Information\n”,i+1);  printf(“Name : %s”, students[i].name);  printf(“Age : %d”, students[i].age);  printf(“GPA : %lf”, students[i].gpa);  }  return 0;  } |
| **Functions**   1. Get student information and fill up the entire ‘students’ structure 2. Calculate average grades of all students. 3. Find the best student - with highest GPA. | #include <stdio.h>  #include <string.h>  typedef struct Student\_struct  {  char name[10];  int age;  double gpa;  } Student;  void getStudentInformation(Student s[], int N)  {  int i=0;  for(i=0; i<N; i++)  {  printf(“\nEnter info for Student %d\n”, i+1);  printf(“Enter name : ”);  scanf(“%s”, s[i].name);  printf(“Enter age : ”);  scanf(“%d”, &s[i].age);  printf(“Enter GPA : ”);  scanf(“%lf”, &s[i].gpa);  }  }  double getAverageGPA(Student s[], int N)  {  double sum = 0;  int i=0;  for(i=0; i<N; i++)  {  sum += s[i].gpa;  }  return sum/N;  }  Student getBestStudent(Student s[], int N)  {  double maxGPA = -1;  int i=0, idx = -1;  for(i=0; i<N; i++)  {  if(s[i].gpa >= maxGPA)  {  idx = i;  maxGPA = s[i].gpa;  }  }  return s[idx];  }  int main(int argc, char \*argv[])  {  int N=5;  Student students[5];  getStudentInformation(students, N);  double avg = getAverageGPA(students, N);  printf(“\nStudent average GPA is %lf\n”, avg);  Student bestStudent = getBestStudent(students,N);  printf(“\nStudent with highest GPA is\n”);  printf(“Name : %s\n”, bestStudent.name);  printf(“Age : %d\n”, bestStudent.age);  printf(“GPA : %lf\n”, bestStudent.gpa);  return 0;  } |
| **Separate .h & .c for structs**  Structs are usually written in 2 separate files.  The header file (.h) contains the struct definition and the function prototypes.  The source file (.c) contains the function definitions. It also includes the above .h file.  The main file (which contains the main function), includes the struct .h file in order to use the structure and the functions.  To compile such a code, we need to have both the .c files (struct and main) in the gcc command. E.g.  gcc student.c test.c -o a4 | /\* student.h file \*/  #ifndef STUDENT\_H  #define STUDENT\_H  typedef struct Student\_struct  {  char name[10];  int age;  double gpa;  } Student;  void getStudentInformation(Student[], int);  double getAverageGPA(Student[], int);  Student getBestStudent(Student[], int);  #endif  /\* student.c file \*/  #include<stdio.h>  #include<string.h>  #include"student.h"  void getStudentInformation(Student s[], int N)  {  int i=0;  for(i=0; i<N; i++)  {  printf("\nEnter info for Student %d\n", i+1);  printf("Enter name : ");  scanf("%s", s[i].name);  printf("Enter age : ");  scanf("%d", &s[i].age);  printf("Enter GPA : ");  scanf("%lf", &s[i].gpa);  }  }  double getAverageGPA(Student s[], int N)  {  double sum = 0;  int i=0;  for(i=0; i<N; i++)  {  sum += s[i].gpa;  }  return sum/N;  }  Student getBestStudent(Student s[], int N)  {  double maxGPA = -1;  int i=0, idx = -1;  for(i=0; i<N; i++)  {  if(s[i].gpa >= maxGPA)  {  idx = i;  maxGPA = s[i].gpa;  }  }  return s[idx];  }  /\* test.c file \*/  #include<stdio.h>  #include<string.h>  #include"student.h"  int main(int argc, char \*argv[])  {  int N=5;  Student students[5];  getStudentInformation(students, N);  double avg = getAverageGPA(students, N);  printf("\nStudent average GPA is %lf\n", avg);  Student bestStudent = getBestStudent(students,N);  printf("\nStudent with highest GPA is\n");  printf("Name : %s\n", bestStudent.name);  printf("Age : %d\n", bestStudent.age);  printf("GPA : %lf\n", bestStudent.gpa);  return 0;  } |
| **Weekly Assignment - 4**  Write a program for 2D points   * Create a struct for 2D points (x, y) that can have values of type double * Get 2 points from user * Calculate distance between the points * Find the mid-point of the 2 points * You should have different functions   + Get input from user and save in struct   + Calculate distance between the points   + Create and return a new point which is the mid-point of the input two points | /\* a4.c file \*/  #include<stdio.h>  #include"point.h"  int main(int argc, char \*argv[])  {  int N=2;  Point points[2];  // Call a function which gets N points from user  // Find distance between two points and print it  // Call a function that calculates mid-point for two input points and returns the new point. Print this mid-point here.  return 0;  }  /\* point.h file \*/  /\* point.c file \*/ |