|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **tType** | **Storage Size** | **Value Range** | **printf** | **scanf** |
| char | 1 byte | 0 to 255  (ASCII A : 65 a : 97 0: 48) | %c | %c |
| int | 4 bytes | -2,147,483,648 to 2,147,483,647 | %d | %d |
| float | 4 bytes | 1.2E-38 to 3.4E+38 (Precision to 6 decimal places) | %f | %f |
| double | 8 bytes | 2.3E-308 to 1.7E+308 (Precision to 15 decimal places) | %lf | %f |
| Int/float/double/char | | Pointers declared as int \*x | %p (if printf x)  ^prints address not value.  %d (if printf \*x) | NA |

**note:** int, float, double can be assigned values of other types apart from char. ie. int can be assigned a float/double, but no decimal will be taken in. Double can be assigned int, as x.00.

|  |  |
| --- | --- |
| **Header** | **Commonly used functions** |
| <stdio.h>  ~ Standard input output | printf( ) -- display output  scanf(“%d”, &x) -- input into the address of variable x |
| <math.h>  ~ Mathematics | ceil( x ) -- Get smallest integral value that exceeds x  (Round up)  floor( x ) -- Get largest integral value less than x  (Round Down)  fabs ( x ) -- Compute absolute value of x  pow ( x, y ) -- Compute x raised to the power y  sqrt( x ) -- Compute the square root of x |
| <stdlib.h>  ~ Standard library | rand ( ) -- Generate pseudo-random number  srand ( ) -- Seed pseudo-random number  to get a specific range of random numbers, use rand()%a + b where range  = b < rand < a  Alternative: rand ()%(max-min) + min  min=< Rand < max |
| <time.h>  ~ Time | time ( ) -- current time  time(NULL) -- use in srand(time(NULL)) |

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| = | Assign right to left |
| == | Equivalent |
| += | Add right to left and assign to left |
| -= | Subtract right from left and assign to left |
| /= | Divide left by right and assign to left |
| %= | Modulus left by right and assign left |
| \*= | Multiply left by right and assign left |
| a < x < b | Left to right. a < x is a Boolean if true = 1, false = 0. Then 0/1 < b. |
| x++ | Return value of x before incrementing x. |
| ++x | Increments x before returning value of x. |
| ++(x+y) | Gives syntax error, does not work. |
| a>0 || b>0 | If a> 0, condition is true, so will not check for b>0 |
| a>0 && b>0 | True only if both a>0 and b>0 |
| x = (3 + 2 && 5 ) || y++ | X = 1. Y will not increment. If 1st condition (3+2 && 5) != 0, treated as true (1), and 2nd condition will not run for a || statement. So y will not be incremented. |
| x -= y + z | x = x – (y + z) = x – y – z |
| x /= y %= z | Right to left. y %= z 1st. Then x /= resultant. |

|  |  |
| --- | --- |
| **Properties of Conditional Operators ( <, <=, >, >= )** | |
| **Incorrect way** | **Correct way** |
| num = 10;  1 < num < 5  ≡ (1 < num ) < 5  ≡ (true) < 5  ≡ (1) < 5  ≡ true | num = 10;  1< num && num<5  ≡ (1 < 10) && (10 < 5)  ≡ (true) && (false)  ≡ (false) |

|  |  |
| --- | --- |
| **if and else if statements** | **Switch statements** |
| if ( condition ){  /\* Do Something\*/ ;  }  else if ( another condition){  /\* Do Something\*/ ;  }  else{  /\* Do something\*/ ;  } | switch (variable to test){  case 1: /\*Do Something\*/ ; **break;**  case 2: /\*can assign somthing\*/ ; **break;**  case 3: /\*can print somthing\*/ ; **break;**  case 4: /\*can leave blank too\*/ ; **break;**  case 5: /\* can call function\*/ ; **break;**  case6:/\* can be if/else/switch too \*/; **break;**  .......... ................ .........  case n: /\*Do Something\*/ ; break;  default: /\*Do Something\*/ ; break;  }  *N can only be (+/-)int / char and cannot be inequality*  ***If break is not used, it will run every single case from the case which the variable is true. (ie if true on case2, and no breaks used until default, will run all the codes until it reaches default)*** |
| **if and else statement (conditional version)** | **for loop** |
| (condition) ? /\*if true\*/ : /\*if false\*/;  Example:  biggerNum = (num1 > num2) ? num1: num2 ; | for ( starting value; condition; changes){  /\*Do Something\*/ ;  /\* repeat when the condition is true\*/ ;  }  *Usually used when you know how many loops to run.* |
| **while loop** | **do-while loop** |
| While (condition){  /\*Do Something here\*/ ;  /\*repeat when the condition is true\*/ ;  } | do{  /\*Do something first\*/ ;  /\*Repeat if condition is true\*/ ;  /\*Guarantee one loop\*/ ;  } while (condition) ; |
| **Factoria: input, counter, factoria = 1** | **Dissect Number: num; counter; i; arr[20]** |
| if (input = 0){  factoria = 1;  }  else{  for (i=2; i <= input; i++){  factoria \*= counter;  }  } | while (num > 0){  num/10;  counter++;  }  for ( i = 0; i < counter; i++){  arr[ i ] = num%10;  num/10;  } |

|  |  |  |
| --- | --- | --- |
| **GCD**: num1, num2, lower, i, GCD | **LCM**: num1, num2, i, lower, LCM | |
| lower = (num1 > num2) ? num2 : num1;  for (i = lower; i >= 1; i- - ){  if ( num1%i == num2%i && num1%i==0){  GCD = i;  break;  }  } | lower = (num1 > num2) ? num2 : num1;  for (i=lower; i>=1; i- - ){  if (num1%i == num2%i && num1%i==0 ){  LCM = num1\*num2/i;  break;  }  } | |
| **PrimeTest**: num, i | **Swap numbers**: num1, num2, temp | |
| for(i = 2; i < num/2; i++){  if (num%i = = 0){  /\* num is not prime\*/  break; or return 0;  }  }  return 1; /\*num is prime\*/ | **Normal** | **With Pointers** |
| temp = num1;  num1 = num2;  num2 = temp; | void swapFunc(int \*num1, int \*num2){  int temp;  temp = \*num1;  \*num1 = \*num2;  \*num2 = temp;  }  When using the function:  swapFunc(**&**num1, **&**num2); |

|  |  |  |
| --- | --- | --- |
| **Properties of function/array** | **Examples** | **Reasoning** |
| **Passing arrays to functions** can be int[] or int \* | ie. Void scanarray(int arr[], size) or void scanarray(int \*, size) | Because when passing array to a function, it is passing the address of the 1st value in the array, which is what a pointer does as well. |
| A function does not need to have a parameter to be valid. | ie. int testing() |  |
| Declaring more values than function size only results in a warning. | ie. int arr[4] = {0,1,2,3,4} | gives a warning, prog can still be compiled. |
| Initialising an array. | Arr[6] = {0}  Arr[6] = {1, 2}  Arr[] = {1,2} | Initialises all elements in array to 0  Only 1st 2 elements initialised, others are given a random value. |
| Initialising 2d array to 0 | Arr[][MAXCOL] = {{0}} |  |
| Declaring 2d arrays, must mention numCols.  Arr[numRows][numCols] | ie. Arr[][3]  Arr[6][2]  Arr[3][MAX\_COL] | System needs the numCols to create a 2d array in memory. |

|  |  |
| --- | --- |
| **For loop to insert values into array** | **For loop to print array** |
| int main(void) {  int numbers[6], i;  printf(“Enter… : “);  for (i = 0; i < 6; i++)  scanf("%d", &numbers[i]);  ...  } | void printArray(int arr[], int size) {  int i;  for (i = 0; i < size; i++)  printf("%d ", arr[i]);  printf("\n");  } |
| **Sum array** | **Passing from 1 array to another** |
| int sum(int arr[], int size) {  int i, sum = 0;  for (i = 0; i < size; i++)  sum += arr[i];  return sum;  } | int j = 0;  for (i=0; i<size; i++){  array2[j] = array[i];  j++;  } |

|  |  |
| --- | --- |
| **Creating infinity loop** | **Generating list of random numbers** |
| while (1){  /\*Do something\*/  if(condition)  break;  else if(another condition)  return value;  }  while(1) ≡ while(true) ≡ infinity loop  Useful when you want repetition and do different things depending on a lot of different conditions.  So you can’t really put one single condition inside the while( ). | #include<stdlib.h>  #include<time.h>  int generate\_random(int arr[], int size){  int i;  srand(time(NULL));  for(i=0; i<size; i++){  // insert rand number btwn b and a into array  arr[i] = rand()%a + b;  }  }  Remember to put srand(time(NULL)) function OUTSIDE THE LOOP.  Range of random numbers example:   * **34 to 77**: rand()%44 + 34   rand()%44 will generate num from 0 to 43   * **340 to 450**: rand()%111 + 340   rand()%111 will generate num from 0 to 110 |
|  | **Inserting/printing into 2d arrays** |
|  | void scan\_array(int arr[]**[max\_col]**, int rowsize, int colsize){  int i, j;  for (i=0; i<rowsize; i++){  for(j=0; j<colsize; j++){  scanf(“%d”, &array[i][j]);  OR  printf(“%d”, array[i][j]);  }  }  } |

|  |  |
| --- | --- |
| **Pointers in functions** | **Initialising pointers.** |
| void figure\_me\_out(int \*, int, int \*);  int main(void){  int a=10, b=20, c=30;  figure\_me\_out(&a, b, &c);  printf("%d %d %d\n", a, b, c);  return 0;  }  void figure\_me\_out(int \*i, int j, int \*k){  int c = \*k;  (\*i)++; j += 2; c -= 10;  } *Answer: a = 11, b = 20, c = 30*  *Cos only the pointer i is changed in the new function, the other pointer was assigned to a variable before changing the variable, meaning original pointer is untouched.* | Int a= 2, b=3;  Int \*a\_p, \*b\_p;  a\_p = &a;  b\_p = &b  \*a\_p = 1 + \*b\_p => this means a = 1 + b  a\_p = 1 + b\_p => this means address of a = 1 + address of b. Cannot use this.  \*(a\_p)++ => a++  a\_p++ => increment in pointer address. Used to move from 1 element in an array to the next. Can be used. |
| **Generating non-repeating random numbers** | |
| #include <stdio.h>  #include <time.h>  // seed time ONCE  srand(time(NULL));  // generate 10 random numbers  for(d = 0; d <10; d++){  // Assign random number into array first  arr[d] = rand()%A + B;  // loop through filled array only. This loop will not occur for the first element since (when d=0, e=0, e is NOT < d )  for(e=0; e<d; j++){  if(arr[d] == arr[e]){    // if it is repeated, d - -, number at index d will be re-assigned  d - - ;  break;  }  }  } | |
| **Multiple Conditions testing (Main cause of unwanted infinity loops)** | |
| If you want to loop until ONE condition is met:  while( num1 != 10 && num2 != 20 && num3 > 30 ){  /\* loop will break when (num1 == 10) OR (num2 == 20) OR (num3 **<=** 30)\*/  }  while(1){ // alternative method  if(num1 == 10 || num2 == 20 || num3 <= 30)  break;  } | |

|  |
| --- |
| If you want to loop until ALL conditions are met:  while(num1 != 10 || num2 != 20 || num3 > 30){  /\* loop will break only when (num1 == 10) AND (num2 == 20) AND (num3 **<=** 30)\*/  }  while(1){ // alternative method  if(num1 == 10 && num2 == 20 && num3 <= 30)  break;  } |

**FILE IO**

|  |  |
| --- | --- |
| Syntax | Meaning |
| fopen(X,Y) | To open a file ( to read/ write / append etc), X is file name, Y is action taken with file ( read / write / append etc)  *Returns NULL if error* |
| fclose(X) | To close the file after done using, X is file name  *Returns end of file (EOF) if error* |
| fscanf(X,Y,Z) | Equivalent of scanf but scanf is input from stdin, X is pointer for input(file name/stdin), Y is same as scanf (“%d or %f” etc), Z is address to scan to (&num1 / &value etc)  *Returns negative value if error, else returns number of chars stored* |
| fprintf(X,Y,Z) | Equivalent of printf but printf is pointer to stdout, X is pointer for output( file name/stdout), Y is same as printf(“blah blah blah %d\n”), Z is variable to print same as printf (num1, value etc)  *Returns EOF if error, else returns numbers of chars written.* |
| feof() | Used to check for EOF.  *Returns non-zero value (true) when EOF , otherwise returns 0*  *It returns true only after EOF is read, not when EOF is reached, can result in extra iterations if used in loops* |
| ferror() | Used to check for error  *Returns non-zero if error, otherwise returns 0.* |
| fputc(x,y)  fputs(x,y) | Writes char x to file y.  Writes string x to file y with newline character ‘\n’ |
| fgetc(x)  fgets(x, y, z) | Reads x from file y.  Reads x (pointer to 1st char in the string array to store what is read) up to y-1 characters, or ‘\n’ reached from file z. |

**Note:** fscanf(stdin, “%d”, &num) is same as scanf(“%d”, &num). fprintf(stdout, “num = %d”, num) is same as printf(“num = %d”, num).

|  |  |
| --- | --- |
| To scanf from a file | To printf to a file |
| int scanPrices(float arr[]) {  FILE \*infile; // pointer to your file.  int size, i;  // open file for reading  infile = fopen("prices.in", "r");  … whatever you want to scan with the file ie.  fscanf(infile, "%d", &size);  // close the file ( not needed but good practice)  fclose(infile);  return size;  } | void printResult(float total\_price) {  FILE \*outfile;  // open file for writing  outfile = fopen("prices.out", "w");  … whatever you want ot print in the file ie.  fprintf(outfile, "Total price = $%.2f\n", total\_price);    fclose(outfile);  } |
| To check if a file is opened properly |  |
| int scanPrices(float arr[]) {  FILE \*infile;  int size, i;  if ((infile = fopen("prices.in", "r")) == NULL) {  printf("Cannot open file \"prices.in\"\n");  exit(1);  }  . . .  }  *exit() is similar to return(), 0 means a good run.*  *can use to trace where program terminates (i don’t know how tho ☹)*  *requires <stdlib.h>* |  |

|  |  |
| --- | --- |
| Modes when opening files | Meaning |
| “r” | Open file for reading (file must already exist) |
| “w” | Open file for writing (new file will be created if does not exist, otherwise overwrite old file) |
| “a” | Open for appending (new file will be created if does not exist) |
| “r+” | Open for reading and writing, starting at beginning |
| “w+” | Open for reading and writing (truncate if file exists) |
| “a+” | Open for reading and writing (append if file exists) |

**Only important ones are “r” and “w”**

**Strings & chars**

|  |  |
| --- | --- |
| **Headers** | **Functions** |
| **<**ctype.h>  ~character type | isalphanum(x) -- check if x is alphanumerical ( x is a char)  isalpha(x) -- check if x is alphabet  isdigit(x) -- check if x is a digit  isspace(x) -- check if x is white-space ( “ ”, “\n” included)  ispunct(x) -- check if x is punctuation  islower(x) -- check if x is lowercase  isupper(x) -- check if x is uppercase  tolower(x) -- converts x to lowercase  toupper(x) -- converts x to uppercase  *for tolower, toupper, x is not changed, they convert x to lower/upper for comparison in that instance only. x will remain as it is.* |
| **<**string.h> | strlen(str) --to count the length of str  strcat(str1, str2) --append str2 to the end of str1  strncat(str1, str2, n) --append n chars from str2 to the end of str1  strcpy(str1, str2) --copy str2 into str1 (replaces whatever is in str1)  *if str2 is longer than str1, strcpy will cause it to copy into unknown memory locations*  strncpy(str1, str2, n) -- copy n chars from str2 into str1  strcmp(str1, str2) -- compares the ASCII values of corresponding chars in str1 and str2. Returns negative integer is str1 is lesser than str2, returns positive integer is str1 is greater, returns 0 if equal  strncmp(str1, str2, n) -- compares for first n chars in str1 and str2  strchr(str1, char) -- Returns pointer to first occurrence of char in str1.  strrchr(str1,char) -- Returns pointer to last occurrence of char in str1  strstr(str1, str2) -- Returns pointer to first occurrence of str2 in str1  strtok(str1, n) -- Breaks str1 into ‘tokens’ split by delimiter ‘n’  atoi(str) -- converts str into int  *only for digits until 1st non digit char* |

*Note: Chars are declared in ‘ ’, Strings are declared in “ ”. Char can be treated like an integer(print %d, +, -). In these cases, ASCII value of char is used / changed*

|  |  |
| --- | --- |
| **Other Syntax** | **Meaning** |
| %s | How to represent strings in printf / scanf  Printf : prints until ‘\0’  Scanf : scans until whitespace encountered. |
| putc(x), putchar(x) | Writes x to stdout, similar to printf |
| getc(), getchar() | Reads a char from stdin |
| ungetc(x, y) | Pushes back the last char from stream y(can be stdin or a file) that has been stored in variable x (var or arr) |
| puts(x) | Writes x to stdout with newline char ‘\n’ |
| gets(x) | Reads string x from stdin  *don’t use due to security error.*  *Use fgets(str, size+1, stdin); or scanf(“%s”, str)*  *However, scanf reads only until whitespace, while fgets reads until size or \n, so fgets is better to use.* |

|  |  |
| --- | --- |
| **Initializing Strings** | **Adding \0 for user input** |
| char str[x] = “pudding”  OR  char str[x] = {‘p’, ‘u’, ‘d’, ‘d’, ‘i’, ‘n’, ‘g’, ‘\0’}  x = max length of string array.  *using “ ”adds \0 at the end automatically.*  *Usually not needed as we take strings frm user input.*  **Array of strings**  Char str[x][y] = {“banana”, “cow”, “pineapple”};  x = max number of strings  y = max length of each string. | fgets(str, size, stdin);  length = strlen(str);  if (str[length-1] == ‘\n’){  str[length-1] = ‘\0’;  }  *Because fgets takes in the ‘\n’ char as well before it terminates.*  **Array method**  i = 0;  scanf(“%c”, &arr[i]);  while (str[i] != ‘\n’){  i++;  scanf(“%c”, &arr[i]);  }  arr[i] = ‘\0’; |
| **Counting strlen without using function.** |  |
| int mystrlen(char \*p) {  int count = 0;  while (\*p++) {  count++;  }  return count;  } |  |

Good luck with recursion, can’t help there. Too many kinds of code, depends on your own thinking liao

|  |  |
| --- | --- |
| **Getting sum of digits using recursion** | **Summing 2 digits with recursion** |
| int getSum(int num){  int sum = 0, r;      if (num != 0){        r = num%10;        sum = sum + r;        getSum(num / 10);      }      return sum;  } | int add(int n1, int n2, int carry){  int sum;    if(n1 == 0){  return 0;  }else{  Sum = n1%10 + n2%10 + carry;  if(sum>9){  return sum%10 + 10\*add(n1/10, n2/10, 1);  } else{  return sum + 10\*add(n1/10, n2/10, 0);  }  }  }  From Pyp, to sum numbers like pri sch method  ie 4830  + 1234  6064 |

|  |  |
| --- | --- |
| **Factorial** | **Counting Occurances** |
| int factorial(int n) {  if (n == 0)  return 1;  else  return n \* factorial(n-1);  } | int countValue(int value, int arr[], int size) {  if (size == 0)  return 0;  else {  if(value == arr[size-1]){  return 1 + countValue(value, arr, size-1);  }else {  return countValue(value,arr,size-1);  }  }  } |
| **Auxiliary function** | |
| *To use if you need to add extra parameters to the recusive function*  ie for counting occurances, if want to count from the starting value.  //this is the auxiliary function, to keep the function call in the main function the same  int countValue(int value, int arr[], int size) {  return countValue\_recur(value, arr, 0, size);  }  int countValue\_recu(int value, int arr[], int start, int size) {  if (start == size)  return 0;  else  return (value == arr[start]) +  countValue(value, arr, start+1, size);  } | |

**Sorting and Searching**

|  |  |
| --- | --- |
| **Linear Search** | **Binary Search (better)** |
| // Return index if found, otherwise return -1  int linearSearch(int arr[], int size, int key) {  int i;  for (i=0; i<size; i++){  if (key == arr[i])  return i;  }  return -1; // indicate key not found  } | ***Requires arr to be sorted.***  // Return index if found; otherwise return -1  int binarySearch(int arr[], int size, int key) {  int low = 0, high = size - 1, mid = (low + high)/2;  //Halves the range with each check.  while ((low <= high) && (arr[mid] != key)) {  if (key < arr[mid])  high = mid - 1;  else if (key > arr[mid])  low = mid + 1;  mid = (low + high)/2;  }  //will only occur if mid != key.  if (low > high)  return -1;  else  return mid;  } |
| **Selection Sort** | **Enchanced Bubble Sort (easiest to use/ understand)** |
| // sorting in increasing order  void selectionSort(int arr[], int size) {  int i, start, min\_index, temp;    // each iteration of the for loop is one pass  // find the index of minimum element  for (start = 0; start < size-1; start++) {  min\_index = start;  for (i = start+1; i < size; i++){  if (arr[i] < arr[min\_index]) {  min\_index = i;  }  }  // swap minimum element with element at start index  temp = arr[start];  arr[start] = arr[min\_index];  arr[min\_index] = temp;  }  } | //sorting in increasing order  void bubbleSort(int arr[], int size) {  int i, limit, temp;  int swapEncountered;  // limit is where the inner loop variable i should end  //each iteration brings the largest value to the ‘top’  for (limit = size-2; limit >= 0; limit--) {  swapEncountered = 0;  for (i=0; i<=limit; i++) {  if (arr[i] > arr[i+1]) {  // swap arr[i] with arr[i+1]  temp = arr[i];  arr[i] = arr[i+1];  arr[i+1] = temp;  swapEncountered = 1;  }  }  if(!swapEncountered){  break;  }  }  }  *Remove all the swapEncountered for a normal Bubble sort.*  *swapEncountered is a flag which breaks the outer for loop once no swap has been made inside the inner for loop, meaning the arr is already sorted.* |
| **Insertion Sort** |  |
| // To sort arr in increasing order.  void insertionSort(int arr[], int size) {  int i, j, temp;  for (i=1; i<size; i++) {  temp = arr[i];  j = i-1;  //finds where to slot arr[i] into the values before it, stopping when arr[i] > arr[j] or arr[i] is the smallest value.  while ((j>=0) && (temp<arr[j])) {  arr[j+1] = arr[j];  j--;  }  arr[j+1] = temp;  }  } |  |

**STRUCTURES**

|  |  |
| --- | --- |
| **Declaring Structures** | **Declaring structure inside a structure** |
| typedef struct {  char name[12];  int age;  char gender;  } player\_t;  int main(void){  int a, b;  //to initialize a structure  player\_t player1 = {“Brusco”, 23, ‘M’};  *Usually, will take in user input for structure values* | typedef struct {  int day, month, year;  } date\_t;  typedef struct {  char matric[10];  date\_t birthday;  } student\_t;  int main(void){  student\_t john = {"A0123456Y", {15, 9, 1990}}; |
| **Scanf and Printf from structure** | |
| scanf("%s %d %c", player1.name, &player1.age, &player1.gender);  printf("player1: name = %s; age = %d; gender = %c\n", player1.name, player1.age, player1.gender);  *(.) dot operator is use to access structure. So for struc inside a struc, it is* ***john.date.day*** *etc.* | |
| **Structure to function** | |
| void print\_player(char [], player\_t); //variable type player\_t in function prototype.  int main(void){  // passing structure to the function  Print\_player(“player1”, player1);  player2 = scan\_player();  }  //declare a variable of player\_t inside the function  Void print\_player(char header[], player\_t player){  printf("%s: name = %s; age = %d; gender = %c\n", header, player.name, player.age, player.gender);  }  *You can return a structure from a function as well. ie function :*  player\_t scan\_player() {  player\_t player;  printf("Enter name, age and gender: ");  scanf("%s %d %c", player.name, &player.age, &player.gender);  return player;  } | |
| **Array of structures** | |
| typedef struct {  char code[8];  int enrolment;  } module\_t;  int main(void) {  module\_t modules[10]; //array of 10 structures  int num\_modules;  *to access the array, use modules[i].code / modules[i].enrolment*  *mainly to aid in sorting 2 arrays, since with a structure, only need to sort the array of structures. ie using modules[i].* | |

|  |
| --- |
| **Structures and pointers** |
| *To change values of structures in functions, need to use arrays, similar to variables if cannot return the values. For array of structures, works the same way as array of integers/characters/strings, so array can be changed inside the function.*  int main(void) {  change\_name\_and\_age(&player1); //passing the address of the structure variable is enough  void change\_name\_and\_age(player\_t \*player\_ptr) {  strcpy((\*player\_ptr).name, "Alexandra");  (\*player\_ptr).age = 25;  }  *(\*player\_ptr).age can be shortened with the -> operator to player\_ptr->age*  *Remember the () for the pointers in structure.*  void change\_name\_and\_age(player\_t \*player\_ptr) {  strcpy(player\_ptr->name, "Alexandra");  player\_ptr->age = 25;  } |