**Escape Sequences**

|  |  |  |
| --- | --- | --- |
| Character | Sequence | ASCII |
| alarm | \a | 7 |
| backspace | \b | 8 |
| form feed | \f | 12 |
| newline | \n | 10 |
| carriage return | \r | 13 |
| horizontal tab | \t | 9 |
| vertical tab | \v | 11 |
| backslash | \\ | 92 |
| single quote | \' | 39 |
| double quote | \" | 34 |
| question mark | \? | 63 |

**Input Format**

|  |  |  |
| --- | --- | --- |
| Specifier | Input Text is a | Destination Type |
| %c | character | char |
| %d | decimal | int, short |
| %f | floating-point | float |
| %lf | floating-point | double |

**DeMorgan Law**

adult = !child && !senior;

adult = !(child || senior);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SAMPLE CODE**

#include <stdio.h>

int main(void)

{

const float pi = 3.14159f;

float radius;

float area

printf("Enter radius : ");

scanf("%f", &radius);

area = pi \* radius \* radius;

printf("Area = %f\n", area);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**UNARY OPERANDS**

int main(void)

{

int age = 19;

printf("Prefix: %d\n", ++age); //20

printf(" %d\n", age); //20

printf("Postfix: %d\n", age++); //20

printf(" %d\n", ++age); //22

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CASTING**

int minutes = 45;

hours = **(float)**minutes / 60;

printf("= %.2lf hours\n", hours); //0.75

**OPTIONAL/ALTERNATIVE PATH**

if (likePizza == 1) {

printf("I like pizza\n");

} else {

printf("I else pizza\n");}

**CASE-BY-CASE**

switch (choice) {

case 'A' :

case 'a' :

cost = 1.50;

break;

default:

choice = '?';

cost = 0.0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CONDITIONAL EXPRESSION**

int minutes;

char s;

printf("How many minutes left ? ");

scanf("%d", &minutes);

s = minutes > 1 ? 's' : ' ';

printf("%d minute%c left\n", minutes, s);

return 0;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ITERATION CONSTRUCTS**

**WHILE**

slices = 4;

while (slices > 0) {

slices--;

printf("Gulp! Slices left %d\n", slices);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DO WHILE**

slices = 4;

do {

slices--;

printf("Gulp! Slices left %d\n", slices);

} while (slices > 0); //3 2 1 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**FOR**

for (slices = 4; slices > 0; --slices)

printf("Gulp! Slices left %d\n", slices - 1);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Magic Numbers**

const double PI = 3.14159;

//ou

#define PI 3.14159

//antes do main

int main(void){ …………….. }

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ERRORS**

**SINTACTIC ERRORS** – missing semi-colon, undeclared variable, parentheses missing, return missing.

**SEMANTIC ERRORS** – uninitialized variable, value truncation, = instead of ==, dangling else, infinite it, of by one loop

­­­­­­­**CHARACTER STRINGS**

const int NCHAR = 9;

char name[NCHAR + 1] = "I am Andre";

printf("%s", name);

**STRUCTURES**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h> //looks in system directory

#define SIZE 2 // with “” current directory

#define nums 4

/////////////////////////////////////////////////

struct Student

{

char firstName[51]; // [1 Mark]

char lastName[51]; // [1 Mark]

char studentId[10]; // [1 Mark]

};

struct SenecaStudent

{

struct Student student;

float grades[nums];

//getGrades;;

};

/////////////////////////////////////////////////

void setSenecaStudent(struct SenecaStudent \*);

void setStudentId(char \*);

float getGrades(struct SenecaStudent );

void setGrades(float ar[], int);

float getAverageGrade(struct SenecaStudent[], int);

/////////////////////////////////////////////////

int main(void)

{

struct SenecaStudent students[SIZE] = {

{ { "Susan", "Wong", "123456789" },{ 78.0f, 67.0f, 82.0f, 81.0f } },{ { { '\0' },{ '\0' },{ '\0' } }, 0 }};

float avgGrade;

setSenecaStudent(&students[1]);

avgGrade = getAverageGrade(students, SIZE);

printf("The average grade for the %d Seneca Students is: %.2f", SIZE, avgGrade);

return 0;

}

/////////////////////////////////////////////////

void setSenecaStudent(struct SenecaStudent\* pStudent)

{

printf("Enter First Name:");

scanf(" %40[^\n]", pStudent->student.firstName);

printf("Enter Last Name:");

scanf(" %40[^\n]", pStudent->student.lastName);

setStudentId(pStudent->student.studentId);

setGrades(pStudent->grades, nums);

}

/////////////////////////////////////////////////

void setStudentId(char \*id)

{

printf("Enter the student id:");

scanf(" %9[^\n]", id); // [1 Mark]

}

/////////////////////////////////////////////////

void setGrades(float ar[], int size)

{

//write the loop [2 marks]

for (int i = 0; i < size; i++)

{

printf("Enter a grade : ");

//[1 mark]

scanf("%f", &ar[i]);

}

}

/////////////////////////////////////////////////

float getGrades(struct SenecaStudent pStudent)

{

float result = 0.0f;

for (int i=0; i < nums; i++)

{result += pStudent.grades[i]; }

return result;

}

/////////////////////////////////////////////////

float getAverageGrade(const struct SenecaStudent pStudent[], int size)

{

float result = 0;

result += getGrades(pStudent[0]);

result += getGrades(pStudent[1]);

return result / 2;

}

**SPECIAL FUNCTIONS**

int yes(void)

{

char lChar = 'x';

char lChar2 = 'x';

int response = -1;

scanf(" %c%c", &lChar, &lChar2);

while (!((lChar2 == '\n') && ((lChar == 'Y') || (lChar == 'y') || (lChar == 'n') || (lChar == 'N'))))

{

clearKeyboard();

printf("<Only (Y)es or (N)o: ");

scanf(" %c%c", &lChar, &lChar2);

}

if ((lChar == 'Y') || (lChar == 'y')) {

response = 1;}

else if ((lChar == 'N') || (lChar == 'n')) {

response = 0;}

return response;

}

void sortContacts(struct Contact[], int);

void sortContacts(struct Contact aContact[], int quant)

{

int i, j;

struct Contact temp;

for (i = quant - 1; i > 0; i--) {

for (j = 0; j < i; j++) {

if (strcmp (aContact[j].numbers.cell, aContact[j + 1].numbers.cell) > 0) {

temp = aContact[j];

aContact[j] = aContact[j + 1];

aContact[j + 1] = temp;

}}}

return;

}

math.c

printf("%f\n",sqrt(10.0)); //3.16

printf("%f\n",exp(4.0)); //54.59

printf("%f\n",log(4.0)); //1.38

printf("%f\n",log10(100.0)); //2

printf("%f\n",fabs(-5.2)); //5.2

printf("%f\n",ceil(4.5)); //5.0

printf("%f\n",floor(-4.5)); //-5.0

printf("%f\n",pow(4.0,.5)); //2

printf("%f\n",fmod(4.5,2.0)); //0.5

printf("%f\n",sin(0.0)); //0.00

printf("%f\n",cos(0.0)); /1.00

printf("%f\n",tan(0.0)); //0.00

RANDOM NUMBERS

#include <stdlib.h>

#include <stdio.h>

#include <time.h> // prototype for time(NULL)

int main(void){

int i;

srand(time(NULL));

for (i = 0; i < 10 ; i++)

printf("Random number %d is %d\n", i+1, rand());

return 0;}

int main(void) //10 números de 6 a 100, sempre os mesmos

{

int i, n, a = 6, b = 100;

for (i = 0; i < 10 ; i++) {

n = a + rand() % (b + 1 - a);

printf("Random number %d is %d\n", i+1, n);

}

return 0; }

PASS BY VALUE X PASS BY ADDRESS

int main(void)

{

struct Student harry = { 975, 2, {50.0f, 50.0f}};

set(&harry);

display(harry);}

void set(struct Student\* st)

{ struct Student harry = {306, 2, {78.9, 91.6}};

\*st = harry;}

A variable that holds an address is called a *pointer*. Each pointer type has a special value called its null value. int \*p = NULL;

Some general guidelines for defining a module include:

1. the module is easy to upgrade
2. the module contains a readable amount of code
3. the module may be used as part of the solution to some other problem

For a structured design, we stipulate that

1. each module has one entry point and one exit point
2. each module is highly cohesive
3. each module exhibits low coupling

*Flags* are variables that determine whether an iteration continues or stops.  A flag is either true or false.  Flags helps ensure that no paths cross one another.  By introducing a flag, we avoid the jump and multiple exit, obtain a flow chart where no path crosses any other and hence an improved design.

*ATENCAO (\*n).name é igual a n->name*

A *buffer* is a small region of memory that holds data temporarily and provides intermediate storage between a device and a program.

**strlen()** - returns the number of characters in a character **stringstrcpy()** - copies one character string to another

**strcmp()** - compares one character string to another

**strcat()** - concatenates one character string to another