Основы алгоритмизации и построение структурных схем программ

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Цели и задачи лекции

Цель лекции – изучить основные термины и понятия, используемые в процесссе изучения курса.

Определения

Существует несколько определений понятия алгоритма. Приведем два самых распространенных.

***Алгоритм** – последовательность чётко определенных действий, выполнение которых ведёт к решению задачи. Алгоритм, записанный на языке машины, есть программа решения задачи.*

***Алгоритм** – это совокупность действий, приводящих к достижению результата за конечное число шагов.*

Вообще говоря, первое определение не передает полноты смысла понятия алгоритм. Используемое слово "последовательность" сужает данное понятие, т.к. действия не обязательно должны следовать друг за другом – они могут повторяться или содержать условие.

Определения

Свойства алгоритмов:

***Дискретность** (от лат. discretus — разделенный, прерывистый) – это разбиение алгоритма на ряд отдельных законченных действий (шагов).*

***Детерминированность** (от лат. determinate — определенность, точность) - любое действие алгоритма должно быть строго и недвусмысленно определено в каждом случае. Например, алгоритм проезда к другу, если к остановке подходят автобусы разных маршрутов, то в алгоритме должен быть указан конкретный номер маршрута 5. Кроме того, необходимо указать точное количество остановок, которое надо проехать, скажем, три.*

Определения

Свойства алгоритмов:

***Конечность** – каждое действие в отдельности и алгоритм в целом должны иметь возможность завершения.*

***Массовость** – один и тот же алгоритм можно использовать с разными исходными данными.*

***Результативность** – алгоритм должен приводить к достоверному решению.*

Переменные и константы, типы данных

*Фундаментальные объекты данных, с которыми работает программа, – это **переменные и константы**. Используемые в программе переменные перечисляются в объявлениях или декларациях, в которых указывается их тип, а также иногда их начальные значения.*

*С именами переменных связывается **тип данных**, который контролируется компилятором и для которого выделяется определенное количество байтов памяти*

Переменные и константы

Имена переменных *должны начинаться с буквы (латинского алфавита) или символа подчеркивания (например, _aza), за которым могут следовать любые комбинации букв в любом регистре (заглавные или строчные), символы подчеркивания или цифры 0–9.*

В языке C имеется различие между заглавными и строчными буквами. Поэтому переменная World будет отличаться от переменной world и т.п. При этом в определении переменной не разрешается символ пробела (пробелов) и некоторые другие символы, например, $.

Типы данных

Стандарт C89 определяет пять базовых типов данных:

int *– целочисленный тип, целое число;*

float *– вещественное число одинарной точности с плавающей точкой;*

double *– вещественное число двойной точности с плавающей точкой;*

char *– символьный тип для определения одного символа;*

void *– тип без значения.*

Типы данных

Кроме того, существуют модификаторы, которые могут применяться к этим базовым типам. Ряд компиляторов может поддерживать еще и логический тип ***_Bool.*** *Тип* ***void*** *служит для объявления функции, не возвращающей значения, или для создания универсального указателя (pointer).*

Объект типа ***char*** *всегда занимает 1 байт памяти. Размеры объектов других типов, как правило, зависят от среды программирования и операционной системы.*

Приведем модификаторы базовых типов данных. К ним относятся следующие спецификаторы, предшествующие им в тексте программы:

signed, unsigned, long, short

Типы данных

*Базовый тип **int** может быть модифицирован каждым из перечисленных спецификаторов. Тип **char** модифицируется с помощью **unsigned** и **signed**, тип **double** – с помощью **long**.*

Типы данных

*Базовый тип **int** может быть модифицирован каждым из перечисленных спецификаторов. Тип **char** модифицируется с помощью **unsigned** и **signed**, тип **double** – с помощью **long**.*

Типы данных языка С		
Тип данных	Типичный размер в битах	Минимально допустимый диапазон значений
char	8 (или 1 байт)	от –128 до 127
unsigned char	8	от 0 до 255
signed char	8	от –128 до 127

Типы данных

Типы данных языка С		
Тип данных	Типичный размер в битах	Минимально допустимый диапазон значений
int	16 или 32	от –32768 до 32767
unsigned int	16 или 32	от 0 до 65535
signed int	16 или 32	от –32767 до 32767

Типы данных

Типы данных языка С		
Тип данных	Типичный размер в битах	Минимально допустимый диапазон значений
short int	16	от –32767 до 32767
unsigned short int	16	от 0 до 65535
signed short int	16	от –32767 до 32767

Типы данных

Типы данных языка С		
Тип данных	Типичный размер в битах	Минимально допустимый диапазон значений
long int	32	от –2147483648 до 2147483647
long long int	64	от $-(2^{63}-1)$ до $(2^{63}-1)$ для С99
signed long int	32	от –2147483647 до 2147483647
unsigned long int	32	от 0 до 4294967295
unsigned long long int	64	от 0 до $(2^{64}-1)$ для С99

Типы данных

Типы данных языка С		
Тип данных	Типичный размер в битах	Минимально допустимый диапазон значений
float	32	от 1E–37 до 1E+37 (с точностью не менее 6 значащих десятичных цифр)
double	64	от 1E–37 до 1E+37 (с точностью не менее 10 значащих десятичных цифр)
long double	80	от 1E–37 до 1E+37 (с точностью не менее 10 значащих десятичных цифр)

Пример программы

Пример. *Напишите программу ввода символа, строки, действительных и целых чисел. Действительные числа сложите, целые числа перемножьте. Для действительных чисел использовать типы* ***float*** *и* ***double.***

```
#include <stdio.h>
#include <conio.h>
int main (void) {
  // Объявления
char ch, str[79+1];// С учетом одного места для символа
'\0'
  int x, y, z;
  float a, b, c;
  double A, B, C;
```

Пример программы

```
// Выполнение программы
printf("\n\t Enter a symbol: ");
ch = getchar();
printf("\t The symbol is: %c\n", ch);
_flushall();
printf("\n\t Enter a string: ");
gets_s(str, 79);
printf("\t The string is: %s\n", str);
a = 2.42F; b = 3.58F;
c = a + b;
printf("\n\t The sum %1.2f and %1.2f (as float) is
equal: %1.4f\n", a, b, c);
```

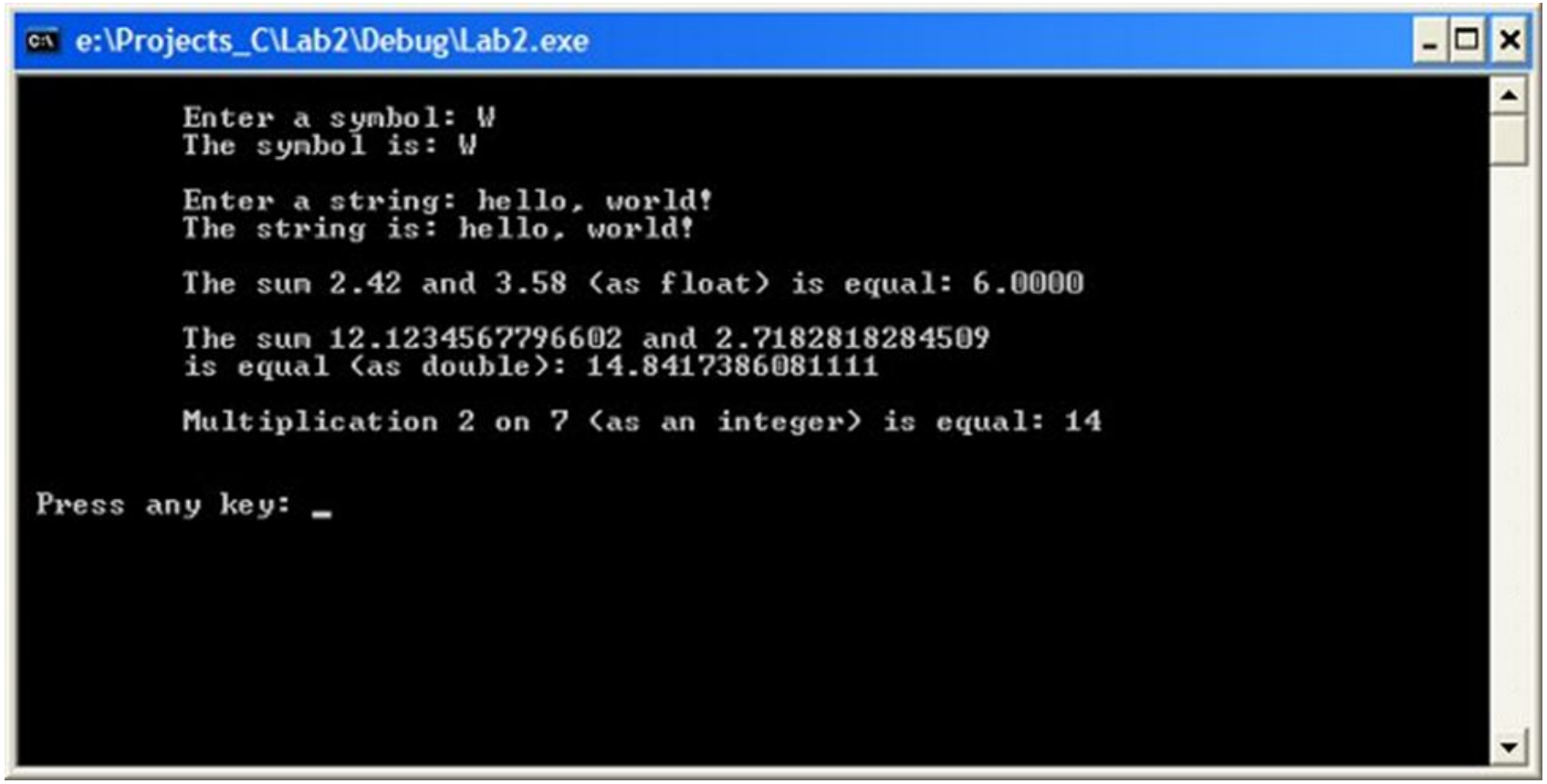
Пример программы

```
A = 12.1234567796602;
B =  2.71828182845O9;
C = A + B;
printf("\n\t The sum %1.13f and %1.13f \n\t is equal (as
double): %1.13f\n", A, B,C);
x = 2; y = 7;
z = x*y;
printf("\n\t Multiplication %d on %i (as an integer) is
equal: %d\n", x, y, z);
```

Пример программы

```
    printf("\n\n Press any key: ");
    _getch();
    return 0;
}
```

Результаты работы программы



Описание функции printf()

Прототип функции **`printf()`** имеет вид:

```
int printf(const char *format, ?);
```

Функция `printf()` записывает в стандартный *поток* `stdout` (стандартный выходной *поток* данных) значения аргументов из заданного списка аргументов в соответствии со строкой форматирования, адресуемой параметром `format`. Строка форматирования состоит из элементов двух типов. К элементам первого типа относятся символы, которые выводятся на экран. Элементы второго типа содержат спецификации формата, определяющий способ отображения аргументов. Спецификация формата начинается символом процента, за которым следует код формата.

Спецификаторы формата функции printf()

Код	Формат
%c	Символ
%d	Десятичное целое число со знаком
%i	Десятичное целое число со знаком
%e	Экспоненциальное представление числа (в виде мантиссы и порядка, e — на нижнем регистре)
%E	Экспоненциальное представление числа (в виде мантиссы и порядка, E — на

Спецификаторы формата функции printf()

Код	Формат
%F	Десятичное число с плавающей точкой (только стандарт C99; если применяется к бесконечности или нечисловому значению, то выдает надписи INF, *INFINITY*(бесконечность) или *NAN* — Not A Number на верхнем регистре. Спецификатор %f выводит их эквиваленты на нижнем регистре)
%g	Использует более короткий из форматов %e или %f
%G	Использует более короткий из форматов %E или %F

Спецификаторы формата функции printf()

Код	Формат
%o	Восьмеричное число без знака
%s	Символьная строка
%x	Шестнадцатеричное без знака (строчные буквы)
%X	Шестнадцатеричное без знака (прописные буквы)
%p	Выводит указатель

Спецификаторы формата функции printf()

Код	Формат
%n	Соответствующий аргумент должен быть указателем на целое число. (Этот спецификатор указывает, что в целочисленной переменной, на которую указывает ассоциированный с данным спецификатором указатель, будет храниться число символов, выведенных к моменту обработки спецификации %n)
%%	Выводит знак процента

Описание функции getchar ()

Прототип функции `getchar()` имеет следующий вид:

```
int getchar(void);
```

Функция `getchar()` возвращает из стандартного потока `stdin` (входного потока данных) следующий символ. При чтении символа предполагается, что символ имеет тип `unsigned char`, который потом преобразуется в *целый*. При достижении конца файла, как и при обнаружении ошибки, *функция* `getchar()` возвращает *значение* `EOF` (*End* Of File – *конец файла*).

Описание функции gets ()

*Прототип функции **gets** имеет следующий вид:*

```
char *gets(char *str);
```

*Функция gets() читает символы (включая пробелы) из стандартного потока stdin и помещает их в массив символов, адресуемый указателем *str (далее это массив символов). Символы читаются до тех пор, пока не встретится разделитель строк или значение EOF. Для реализации EOF на клавиатуре следует набрать одновременно Ctrl+Z. Вместо разделителя строк в конец строки вставляется нулевой символ, свидетельствующий о ее завершении. Следует учесть, что нет способа ограничить количество символов, которое прочитает функция gets(). Поэтому массив, адресуемый указателем *str, может переполниться, и тогда программа выдаст непредсказуемые результаты.*

Описание функции scanf()

Прототип функции **`scanf()`** имеет следующий вид:

```
int scanf(const char *format, ?);
```

Функция `scanf()` представляет собой функцию для ввода данных общего назначения, которая читает *поток* `stdin` и сохраняет информацию в переменных, перечисленных в списке аргументов. Если в строке форматирования встретится разделитель, то *функция* `scanf()` пропустит один или несколько разделителей во входном потоке. Под разделителем, или пробельным символом, подразумевают *пробел*, символ табуляции `\t` или разделитель строк `\n`. Все переменные должны передаваться посредством своих адресов, например, с помощью символа `&`. Управляющая строка, задаваемая параметром `format`, состоит из символов трех категорий: спецификаторов формата, пробельных символов, символов, отличных от пробельных

Описание функции scanf()

Пример использования функции:

```
printf("\t Enter a real number a: ");
scanf_s("%f", &a);

printf("\t Enter a real number b: ");
scanf_s("%f", &b);

c = a + b
```

Спецификаторы формата функции scanf()

Код	Формат
%c	Читает один символ
%d	Читает десятичное целое число
%i	Читает целое число в любом формате (десятичное, восьмеричное или шестнадцатеричное)
%u	Читает десятичное целое число типа short int
%e	Читает число с плавающей точкой (и в экспоненциальной форме)
%E	Аналогично коду %e

Спецификаторы формата функции scanf()

Код	Формат
%f	Читает число с плавающей точкой
%lf	Читает десятичное число с плавающей точкой типа double
%F	Аналогично коду %f (для стандарта C99)
%g	Читает число с плавающей точкой.
%G	Аналогично коду %g
%o	Читает восьмеричное число
%x	Читает шестнадцатеричное число
%X	Аналогично коду %x

Спецификаторы формата функции scanf()

Код	Формат
%s	Читает строку
%p	Читает указатель
%n	Принимает целое значение, равное количеству прочитанных до сих пор символов
%[ ]	Просматривает набор символов
%%	Читает знак процента

Спасибо за внимание