C++ course

name: Виктория Александровна
e-mail: kewtree1408@gmail.com

source: https://bitbucket.org/Kewtree/coursecpp/src
login: coursecpp
password: 12345

General Plans

1. Inception
2. Statements
3. Functions
4. Pointers
5. Massives
6. Structures
7. Files
8. OOP: encapsulation
9. OOP: inheritance
10. OOP: polimorph
11. Test
12. Game over

Functions

1. Repeat (functions)
2. Default arguments
3. Local and global variables (:: statement)
4. Memory class (auto, register, static, extern, dynamic)
5. Inline-functions
6. Recursion
7. Overload functions
8. Pointers (introduction)

Repeat: functions

Функция (в программировании) — это проименованная часть программы, которая может вызываться из других частей программы столько раз, сколько необходимо.

Функция (в математике)— это «закон», по которому каждому элементу одного множества (называемому ***областью определения***) ставится в соответствие некоторый элемент другого множества (называемого ***областью значений***).

Функция (в философии) — обязанность, круг деятельности.

Repeat: functions

```
int main() {
   ...
   return 0;
}

void print_hello() {
   cout << "Hello, world\n";
}

float min(float a, float b){
   return ((a>b)?a:b);
}
```

Repeat: functions

<возвращаемый тип> <название функции> (список входных параметров через ',') {
    return <возвращаемое значение>;
}

Возвращаемое значение обязательно совпадает с возвращаемым типом.

Repeat: definitions

Основные определения:

1. Объявление функции (прототип, сигнатура) - сообщение компилятору ее имени, типа входных параметров и возвращаемого значения.

```
int main();
void print_hello();
float min(float, float);
```

2. Определение функции (реализация) - объявление вместе с телом функции

```
int main() { return 0; }
void print_hello() { cout << "Hello, world\n"; }
float min(float a, float b){ return ((a>b)?a:b); }
```

Repeat: functions

Существует три способа объявления функций:

1. Разместить прототип функции в заголовочный файл (*.h), а затем включить этот файл в исходный текст программы через директиву #include.
2. Записать прототип функции в тот файл, где она используется.
3. Определить функцию до того, как она будет впервые вызвана.

Default arguments

C++ допускает при вызове функции опускать некоторые параметры. Достигается это указанием только в прототипе функции значений аргументов по умолчанию.

```
void print (int a = 5, float b = 9.8);
```

Правило: все параметры справа от аргумента по умолчанию также должны иметь значение по умолчанию.

```
int func (int x, int y = 5, int z = 10);
int func (int x, int y = 5, int z);
```

Default arguments

Example:

skip.cpp -- вызов функции с параметром по умолчанию и без него

Local & global variables

Каждая переменная характеризуется *областью действия*, *областью видимости* и *временем жизни*.
1. *Область действия* -- область программы, в которой переменная доступна для использования.
2. Переменная находится в *области видимости*, если к ней можно получить доступ быть может с помощью операции разрешения видимости (::), в том случае, если переменная непосредственно невидима.
3. *Время жизни* -- интервал выполнения программы, в течение которого эта переменная существует

Local & global variables

Локальные переменные доступны внутри функции, используются и существуют только внутри этой функции.
Глобальные переменные доступны на протяжении всей программы и могут использоваться в любом месте.

Операция разрешения области видимости -- ::

`Example:`

`local_global.cpp --` отличие глобальных и локальных переменных

Memory class

1. Auto -- автоматические
2. Register -- регистровые
3. Static -- статические
4. Extern -- внешние
5. Dynamic -- динамические

Memory class

Example:

static.cpp -- подсчет среднего значения через статическую переменную

count.cpp -- подсчет количества вызовов функции

Inline-functions

Встраиваемые функции.
Зачем?
+ Для увеличения скорости работы программы.

- Увеличения времени компиляции.
- Увеличение размеров машинного кода.

Inline-functions

Example:

inline.cpp -- пример работы на сложении и вычитании переменных.

Recursion

"Чтобы понять рекурсию, нужно сперва понять рекурсию" (c)

Рекурсия - вызов функции из самой себя.

```
void f(){
    f();
}
```

+ функциональность
+ при программировании параллельных процессов
+ улучшается читаемость

Recursion

Example:

factorial2.cpp -- выполнение рекурсии на примере факториала (хвостовая рекурсия)

Overload functions

Перегрузка позволяет иметь несколько одноименных функций, выполняющих схожие операции над аргументами разных типов.

Overload functions

Example:

overload.cpp -- вывод трех различных линий на экран

Pointers

Указатели. Можно ли без них обойтись?

Зачем они нужны?

1! Эффективное распределение памяти

2! Увеличение производительности (времени работы программы)

3. Более глубокое понимание устройства переменных и объектов в C++

4. Возврат из функции более одного значения

Pointers

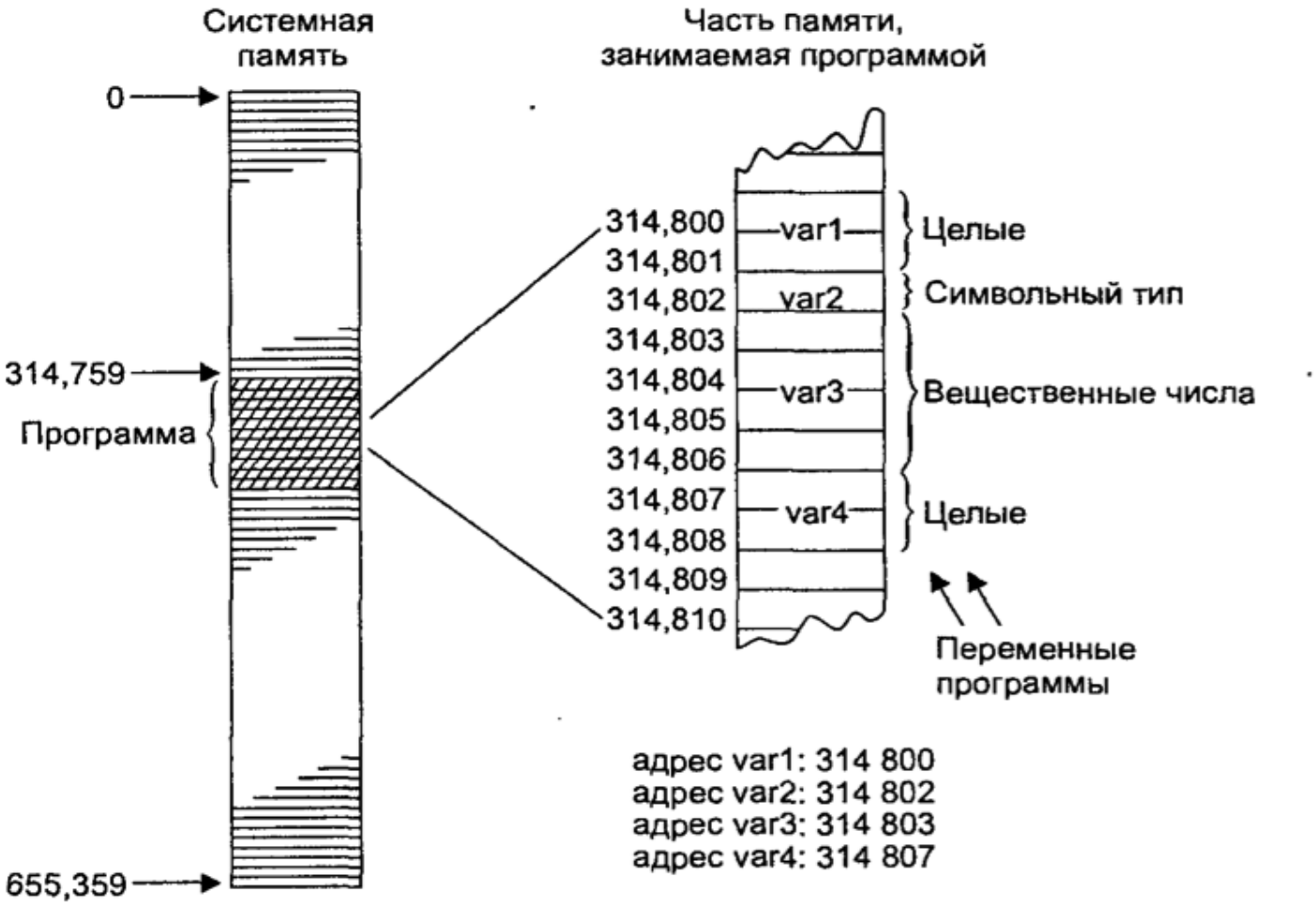

Рис. 10.1. Адреса в памяти

Pointers

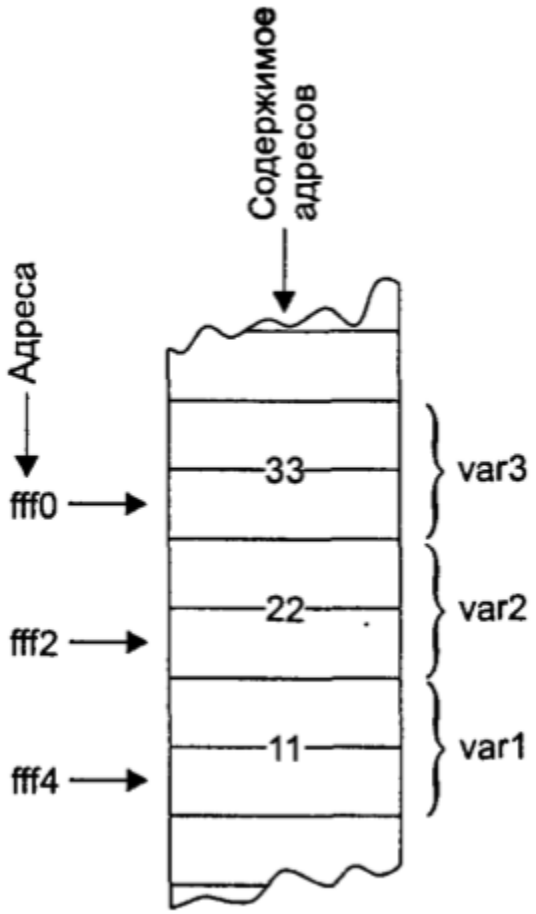

Рис. 10.2. Адреса и содержимое переменных

Pointers: *, &

* -- получение значения по известному адресу

& -- получение адреса по известному значению

Hometask

1. Написать свой калькулятор со всеми известными функциями. Использовать прототипы, функции, перегрузку.
2. Написать функцию вычисления степени с параметром по умолчанию == 2.
3. Написать функцию для чисел фибоначчи (обычную и рекурсивную).
4. Написать функции вычисления периметра для треугольника, квадрата, прямоугольника, окружности. Использовать для выбора диалог.
5. Показать использование оператора :: области видимости на своем придуманном примере.