Этапы развития программирования

- «Блюдо спагетти»
- Процедурное программирование
- Структурное программирование
- Объектное программирование
- Объектно-ориентированное программирование

Предпосылки для появления процедурного программирования

- Необходимость оперирования высокоуровневыми операциями предметной области
- Необходимость накопления алгоритмического материала (библиотек)

Определение функции

```
#include <stdio.h>

double max (double x, double y)
{
        return x > y? x : y;
}

int main (void)
{
        double result = max (10.3, 12.1);
        printf ("max = %lg\n", result);
        return 0;
}
```

Переменные в функциях

```
#include <stdio.h>

void f (double x)
{
        double y = 10.;
        printf ("inside call (step1): x=%lg, y=%lg\n", x, y);
        x = 10.;
        printf ("inside call (step2): x=%lg, y=%lg\n", x, y);
}

int main (void)
{
        double x = 5., y = 5.;
        printf ("before call: x=%lg, y=%lg\n", x, y);
        f (x);
        printf ("after call: x=%lg, y=%lg\n", x, y);
        return 0;
}
```

```
before call: x=5, y=5
inside call (step1): x=5, y=10
inside call (step2): x=10, y=10
after call: x=5, y=5
```

Формальные и фактические параметры

Формальные параметры – это переменные, которые объявляются в заголовке функции

Фактические параметры – это значения, которыми инициализируются формальные параметры при вызове функции (передача по значению)

Указатели в параметрах функций

```
#include <stdio.h>

void max (double x, double y, double * result)
{
        *result =  x > y? x : y;
}

int main (void)
{
        double result;
        max (10.3, 12.1, &result);
        printf ("max = %lg\n", result);
        return 0;
}
```

Массивы в параметрах функций

```
void initArray (double value, double * array, unsigned size)
{
        unsigned i;
        for (i = 0; i < size; ++i)
                array [i] = value;
}

int main (void){
        double arr [10];
        initArray (3.5, arr, 10);
        return 0;
}
```

Прототипы функций

```
#include <stdio.h>

double max (double /*x*/, double /*y*/);

int main (void)
{
        printf ("max = %lg\n", max (10.3, 12.1));
        return 0;
}

double max (double x, double y)
{
        return x > y? x : y;
}
```

Терминология

Описание (declaration) имени –
показывает, ЧТО это имя собой представляет (как его можно использовать). Для функции это прототип.

Определение (definition) имени –
показывает, КАК это имя устроено (для функции это тело). Определение всегда является описанием, но не наоборот.

Глобальные переменные

Любое имя видно в том блоке ({}), где оно определено, а также во всех вложенных блоках. Вне блока имя не доступно.

```
#include <stdio.h>

int glob = 0;                    //глобальная переменная

void f (void)
{
        glob = 10;
}
int  main (void)
{
        f();
        printf ("glob == %d", glob);          //напечатается 10
        return 0;
}
```

Области видимости

Об областях видимости можно говорить только относительно какой-либо структурной единицы

Глобальные имена – (относительно функции) имена, которые видны за пределами функции (глобальные переменные, имена функций)

Локальные имена – (относительно функции) имена, которые видны только в данной функции (локальные переменные – переменные, которые объявляются внутри функций, а также формальные параметры)

Статические переменные

```
#include <stdio.h>
int f (void)
{
        static int result = 0; //выполнится только один раз!
        return result++;
}

int main (void)
{
        int i;
        for (i = 0; i < 5; ++i)
                printf (“f () == %d”, f ());
        return 0;
}
```

```
f () == 0
f () == 1
f () == 2
f () == 3
f () == 4
```

Области существования переменных

Статические переменные – создаются и инициализируются до вызова main, и уничтожаются после выхода из main (глобальные переменные, а также локальные, объявленные как static)

Автоматические переменные – создаются, когда программа доходит до места их объявления, уничтожаются при выходе из блока (составной инструкции или тела функции), где объявлены (локальные переменные, объявленные без static)

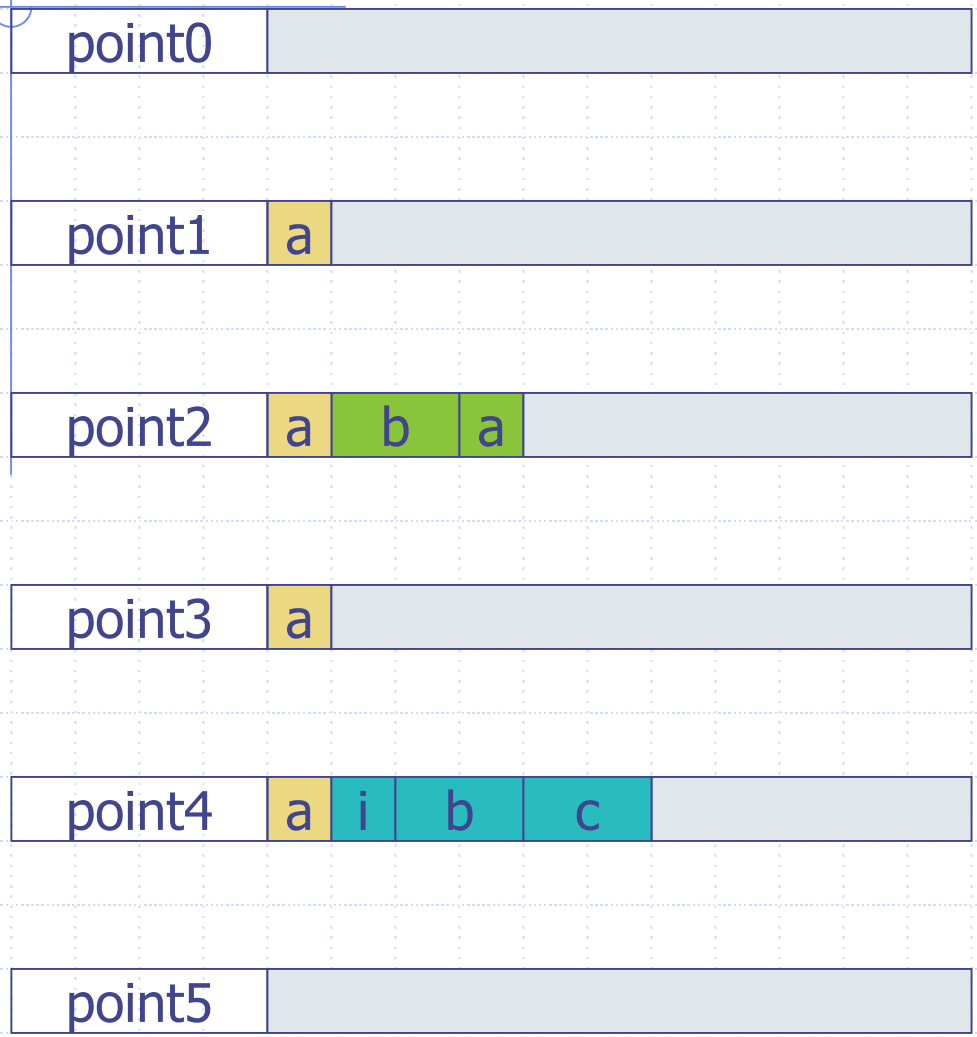
Области памяти

- Область программного кода
- Стек (stack) – автоматические переменные
- Область статических данных
- Куча (heap) – динамически выделяемая память

Стек

Стек (stack, LIFO – Last In First Out) – контейнер данных, для которого определяется понятие вершины,
а также операции добавления элемента в вершину и удаления элемента из вершины

Стек



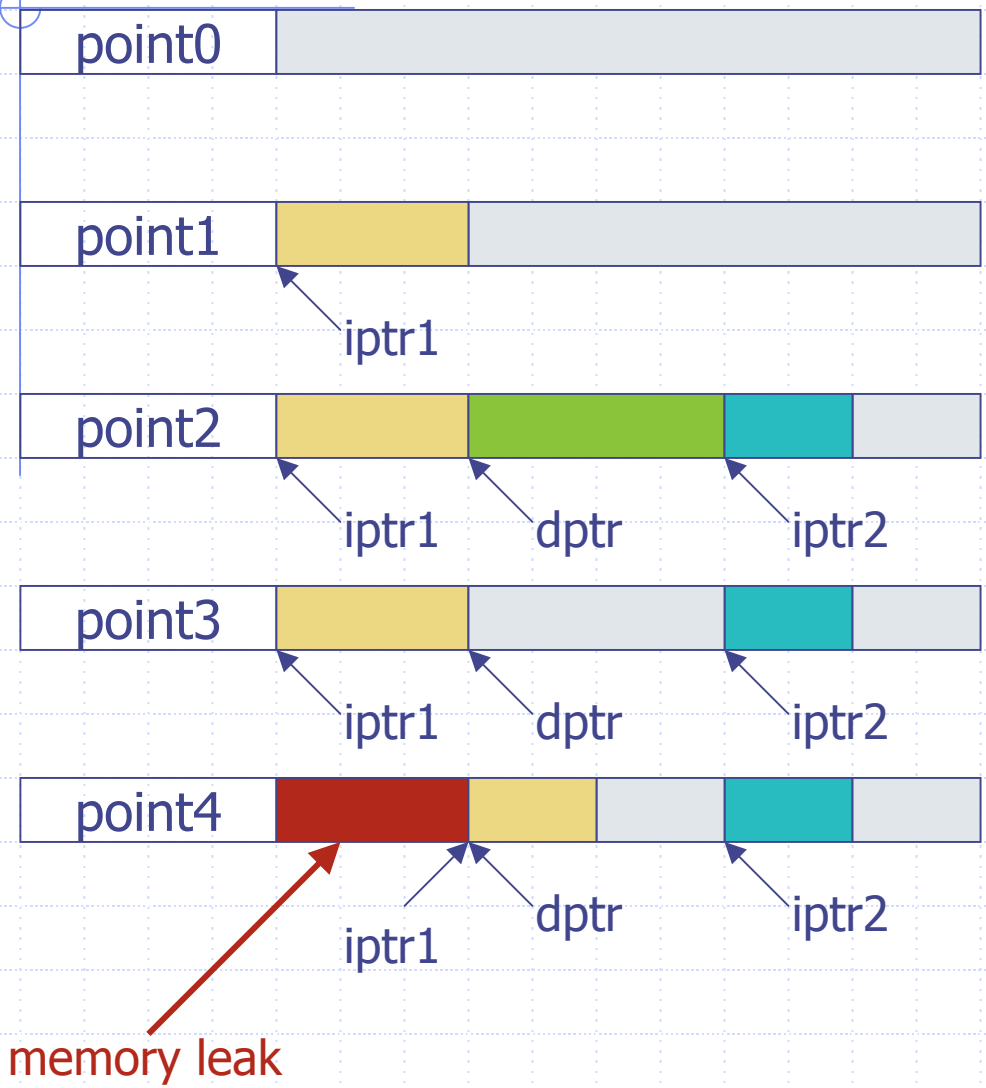

```
void f (int i)
{
        double b, c;
        //point4
}
int main (void)
{
        //point0
        int a;
        //point1
        {
                double b;
                int a;
                //point2
        }
        //point3
        f (a);
}
//point5
```

Область статических данных

Выделение памяти под статические переменные и их инициализация происходит до вызова main, освобождение памяти происходит после выхода из нее

Объем статических данных известен во время компиляции и не изменяется во время исполнения программы

Динамическая память (куча)




```
int main (void)
{
        int *iptr1=0, *iptr2=0;
        double *dptr=0;
        //point0
        iptr1 = new int [3];
        //point1
        dptr= new double [2];
        iptr2 = new int [2];
        //point2
        delete [] dptr;
        //point3
        iptr1 = new int [2];
        //point4
}
```