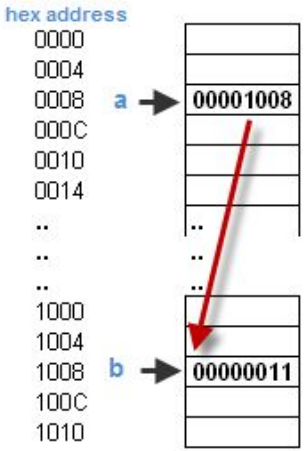
Язык C++

Структуры, объединения, указатели, массивы и строки

Указатель

- Указатель (pointer) – переменная, диапазон значений которой состоит из адресов ячеек памяти и специального значения – нулевого адреса
- Указатель «указывает» хранящимся внутреннего адресом на ячейку памяти, к которой с его помощью можно обратиться
- Значение нулевого адреса используется только для обозначения того, что указатель в данный момент не указывает ни на какую ячейку памяти



Операторы & и *

Унарный оператор **&** выдает адрес объекта

Унарный оператор * есть оператор *косвенного доступа*

Указатели. Операторы & и *

```
int x = 1;
int y = 2;
int z[10];
int* ip;        /* ip - указатель на int */

ip  = &x;       /* теперь ip указывает на x */
y   = *ip;      /* y теперь равен 1 */
*ip = 0;        /* x теперь равен 0 */
ip  = &z[0];    /* ip теперь указывает на z[0] */
```

Указатели

```
int main() {
    int i = 10;
    int j = 12;
    long l = 128L;
    float f = 129.1;

    std::cout << &i << std::endl;
    std::cout << &j << std::endl;
    std::cout << &l << std::endl;
    std::cout << &f << std::endl;

    return 0;
}
```

Указатели

```
int main() {
    bool b = true;
    long l = 128L;
    std::cout << sizeof(b) << std::endl;
    std::cout << sizeof(l) << std::endl;

    bool* pb = &b;
    long* pl = &l;
    std::cout << sizeof(pb) << std::endl;
    std::cout << sizeof(pl) << std::endl;

   return 0;
}
```

Использование указателей в качестве аргументов функций

```
void swap(int x, int y) {
  int temp;
  temp = x;
  x = y;
  y = temp;
}
```

Использование указателей в качестве аргументов функций

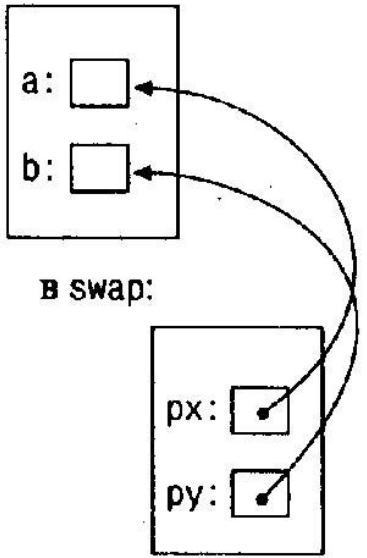
```
int main() {
    int a = 1;
    int b = 2;

    printf("a = %d, b = %d\n", a, b);
    Swap(a, b);
    printf("a = %d, b = %d\n", a, b);
}
```

```
a = 1, b = 2
a = 1, b = 2
```

Использование указателей в качестве аргументов функций

```
void Swap(int* px, int* py) {
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
}
```



a = 1, b = 2
a = 2, b = 1

Массив

- Конечное множество однотипных элементов
- Размер множества не меняется
- Индексация с 0
- Многомерные массивы

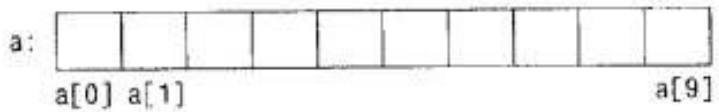
Массив

```
int main() {
    int arr[10];
    int arr2[] = {1, 2, 3, 4, 5};
    int arr3[3] = {1, 2, 3};
    int arr4[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    printf("%d\n", arr2[0]);
    printf("%d\n", arr4[1][2]);
}
```

Связь массивов и указателей

- Определим массив
  int a[10];
- Определим указатель
  int *pa;
- Присвоим указатель адресу первого элемента массива
  pa = &a[0];
- Получим значение первого элемента массива через указатель
  int x = *pa;

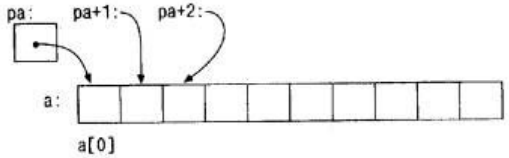


Связь массивов и указателей

- Получим указатель на следующий элемент массива
  *(pa + 1)

- Получим указатель на произвольный элемент массива
  *(pa + i) – это эквивалентно a[i]

- Компилятор преобразует ссылку на массив в указатель на начало массива, следовательно:
  - Имя массива является указательным выражением
  - Записи pa = &a[0] и pa = a эквивалентны
  - Записи a[i], *(a + i), *(pa + i) и pa[i] эквивалентны
  - Массив можно объявлять, как указатель, а потом пользоваться им, как массивом



NULL vs nullptr

```
void func(int*) {
    std::cout << "int func(int*)\n";
}
void func(int) {
    std::cout << "int func(int)\n";
}

int main() {
    func(nullptr);
    func(0);
    func(NULL);  // Compile-time error: call to 'func' is ambiguous

    return 0;
}
```

Строки

- Массив символов
- Заключается в “”
- Escape character
- Null-terminated string

Строки и указатели

```
int main() {
   printf("здравствуй, мир\n");

   char* first_string;
   first_string = "now is the time";

   char second_string[] = "now is the time";

   char* third_string   = "now is the time";
}
```

Строки и указатели. Длина строки

```
unsigned StringLenght(char* str) {
    unsigned result = 0;

    while (*str != '\0') {
        str++;
        result++;
    }

    return result;
}
```

Строки и указатели. Сравнение

```
int StringCompare(char* first, char* second);

int main() {
   printf(
       "%d\n",
       StringCompare("hello world", "hello world")
   );
}
```

Строки и указатели. Сравнение

```
void Test(char* first, char* second) {
    int r = stringCompare(first, second);

    printf(
        "[%s] %c [%s]\n",
        first,
        r == 0 ? '==' : r > 0 ? '>' : '<',
        second
    );
}
```

Строки и указатели. Сравнение

```
int main() {
    Test("", "");
    Test("ab", "a");
    Test("a", "ab");
    Test("abc", "");
    Test("", "abc");
    Test("abc", "Abc");
    Test("Abc", "abc");
    Test("abc", "abc");
}
```

Строки и указатели. Сравнение

```
int StringCompare(char* first, char* second) {
    int i = 0;
    while(first[i] != '\0' && second[i] != '\0'){
       if(first[i] != second[i])
          return first[i] < second[i] ? -1 : 1;
       i++;
    }

    return first[i] == second[i] ? 0 : first[i] < second[i] ? -1 : 1;
}
```

Строки и указатели. Сравнение

```
int StringCompare(char* first, char* second) {
    while(*first && (*first == *second)){
        first++;
        second++;
    }

    return *first - *second;
}
```

Структура

- это одна или несколько переменных (возможно, различных типов), которые для удобства работы с ними сгруппированы под одним именем.

;

Структура

```
struct Point
{
  int x;
  int y;
} pt1, pt2, pt3;

Point p4;
```

- Объявление структуры определяет тип
- Перечисленные в структуре переменные называются *элементами (членами)*

Структура

```
struct Point {
  int x;
  int y;
} ;

int main(int argc, char* argv[]) {
  Point pt;
  Point max_point = {200,300};

  pt.x  = 200;
  pt.y = 250;

  return 0;
}
```

Вложенные структуры

```
struct Rect  {
  Point pt1;
  Point pt2;
};

int main(int argc, char* argv[]) {
  Point pt;
  pt.x  = 200;
  pt.y = 250;

  Rect rec;
  rec.pt1 = pt;
  rec.pt2.x = 1;
  rec.pt2.y = 2;
  return 0;
}
```

Операции над структурами

- Копирования
- Присваивания
- Взятие адреса
- Доступ к элементам

Операции со структурами

```
Point make_point(int x, int y) {
  Point result;
  result.x = x;
  result.y = y;

  return result;
}

int main(int argc, char* argv[]) {
  Point pt = make_point(239, 1);

  return 0;
}
```

Операции со структурами

```
Point AddPoint(
      Point p1,
      Point p2
) {
  p1.x += p2.x;
  p1.y += p2.y;

  return p1;
}
```

Массивы структур

```
struct Record {
  char name[10];
  char surname[10];
  long phone;
};

Record phonebook[200];
```

Указатели на структуры

```
Record* FindRecord(
    long phone,
    Record* records,
    int count
) {
  for(int i=0; i < count; ++i) {
      if(records[i].phone == phone)
          return &records[i];
  }

  return  nullptr;
}
```

Указатели на структуры

```
Record* key = FindRecord("22345", phonebook, 10);

if(key != nullptr) {
  std::printf("Name: %s  Sername: %s", key->name, key->surname);
}
```

Если ***p*** – указатель на структуру, то ***p->элемент структуры*** ее отдельный элемент

Объединения

- это переменная, которая может содержать (в разные моменты времени) объекты различных типов и размеров. Все требования относительно размеров и выравнивания выполняет компилятор. Объединения позволяют хранить разнородные данные в одной и той же области памяти без включения в программу машинно-зависимой информации.

Объединение

```
union Name {
  struct {
     char name[13];
     char code[3];
  };

  struct {
      int32_t i1;
      int32_t i2;
      int32_t i3;
      int32_t i4;
  };
};
```

Объединение

```
bool NameCompare(const Name& a, const Name& b) {
   return (std::strcmp(a.name, b.name) == 0 && std::strcmp(a.code, b.code)) ;
}

bool IntCompare(const Name& a, const Name& b) {
   return (a.i1 == b.i1 && a.i2 == b.i2 && a.i3 == b.i3 && a.i4 == b.i4);
}
```

Объединение

```
int main() {
   Name a = {.name = "0123456789AB", .code = "12"};
   Name b = {.name = "0123456789AB", .code = "10"};

   const uint64_t retry = 100000000000;

   // …

   return 0;
}
```

Объединение

```
std::chrono::system_clock::time_point begin = std::chrono::system_clock::now();
for(int i = 0; i < retry;  ++i)
     NameCompare(a, b);

std::chrono::system_clock::time_point end = std::chrono::system_clock::now();

std::cout << std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count()
          << std::endl;
```

Объединение

```
struct Triangle {
  Point a;
  Point b;
  Point c;
};

struct Rect {
  Point left_top;
  Point right_top;
  Point left_bottom;
};
```

Объединение

```
struct Circle {
  Point center;
  float r;
};

enum FigureType{
  Triangle,
  Rect,
  Circle,
};
```

Объединение

```
union FigureU {
  Triangle triangle;
  Rect     rect;
  Circle   circle;
};

struct Figure {
  FigureType type;
  FigureU   fig;
};
```